Proyecto Integrador: Instalación y Configuración de un Sistema Operativo Linux

Diseño y Administración de un Entorno de Trabajo en Linux para el Desarrollo de Software

Virtualización con Oracle VM VirtualBox y Debian GNU/Linux 13

Juan Sebastian Tafurth Cortes

Juan Esteban Bustos Chilatra

Facultad de Ingeniería, Universidad Surcolombiana

Ingeniería de Software — Segundo Semestre

Sistemas Operativos

Docente: Claudia Marcela Cano Calderón

2026-2

Tabla de Contenido

Introducción....4

Parte I: Instalación y Configuración del Entorno....4

Instalación de Oracle VM VirtualBox.... 4

Creación de la Máquina Virtual 18

Descarga e Instalación de Debian GNU/Linux 13 21

Instalación del sistema operativo Debian 24

- Selección de lenguaje 25
- Selección de ubicación 26
- Configuración de teclado.... 27
- Instalación sistema base 38

Verificación del Funcionamiento del Sistema: 49

Parte II: Identificación de Características del Sistema 49

Comando: cat /etc/os-releas 50

Comando: lsb_release -a.... 50

Comando: uname -r 51

Propósito: Kernel, hostname, arquitectura y fecha 51

Comando: whoami.... 52

Comando: id.... 52

Comando: uname -m 53

Comando: hostname 53

Comando: hostnamectl 54

Comando: lscpu 54

Comando: free -h 55

Comando: lsblk 55

Comando: lspci 56

Comando: lsusb 56

Tabla de Interpretación de Resultados 57

Conclusiones y Análisis Crítico 62

Reflexiones del Equipo 62

Preguntas de Reflexión 63

Introducción

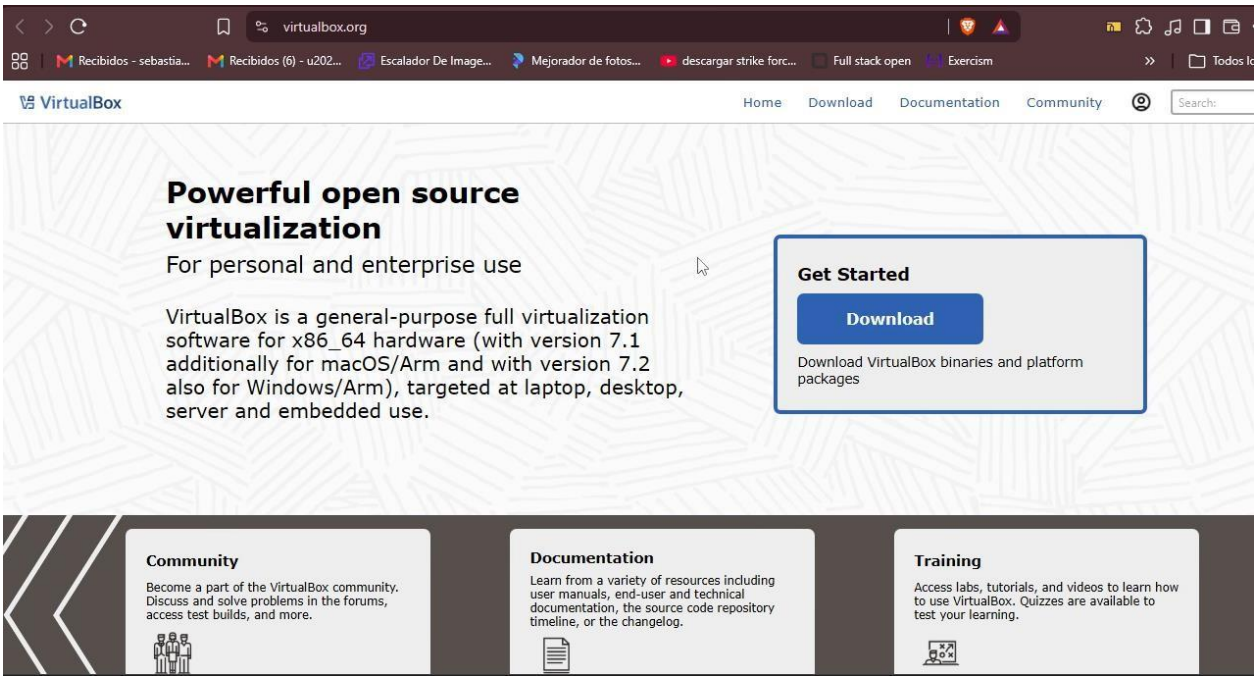
En la presente práctica, correspondiente a las Unidades 1 y 2 del curso de Sistemas Operativos, se llevó a cabo la instalación y configuración de un entorno de virtualización que permite trabajar con un sistema operativo Linux de forma segura y aislada del sistema anfitrión. A través de este proceso, se explora la máquina virtual instalada para identificar los componentes de hardware del sistema y comprender cómo el sistema operativo los reconoce y administra.

Este laboratorio constituye la base práctica sobre la cual se apoyarán las siguientes unidades del curso, abarcando temas de procesos, gestión de memoria, sistemas de archivos y sincronización directamente sobre el sistema instalado. Asimismo, este informe representa el inicio formal de la construcción del Proyecto Integrador del curso, estableciendo el entorno de desarrollo necesario para futuras prácticas.

Parte I: Instalación y Configuración del Entorno Instalación de Oracle VM VirtualBox

Figura 1

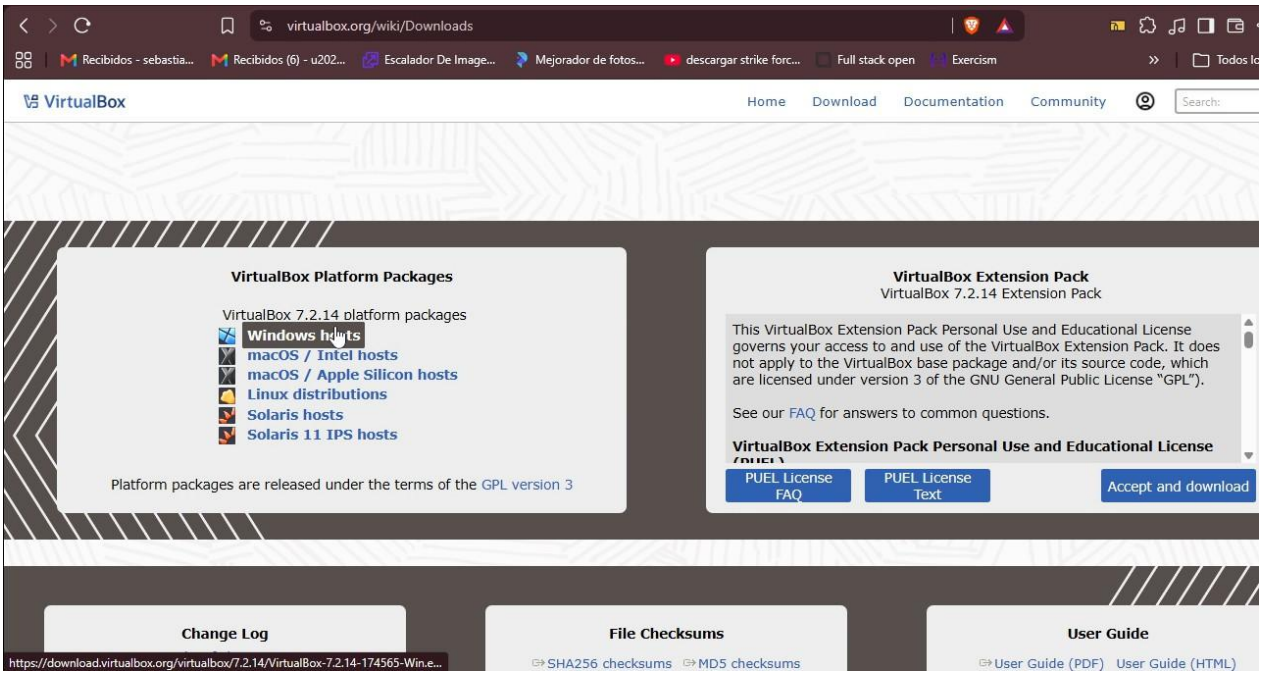
Captura de pantalla - Sitio oficial Virtual Box



Nota. Desde el computador anfitrión, usando el navegador brave, se ingresó al sitio oficial del instalador de Oracle VM VirtualBox.

Figura 2

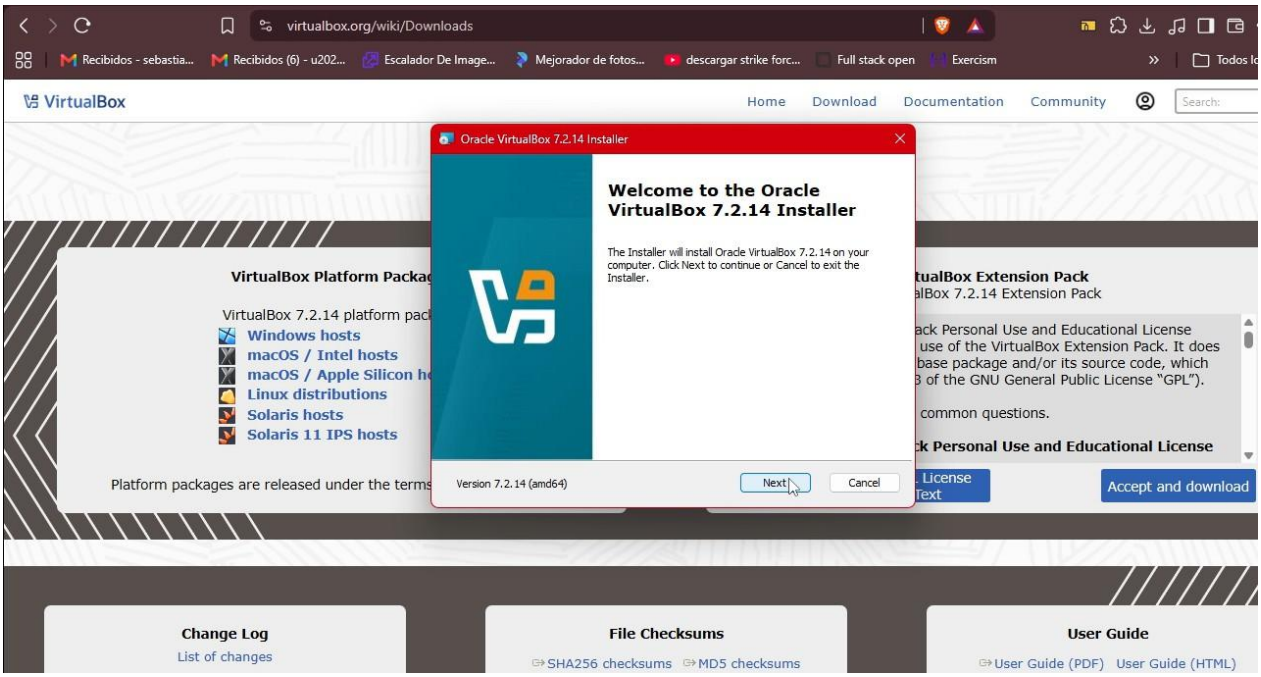
Captura de pantalla - Sitio oficial Virtual Box



Nota. Se seleccionó la versión correspondiente al sistema operativo anfitrión, el cual es Windows 11 y por ende se escogió la opción de instalación Windows Host.

Figura 3

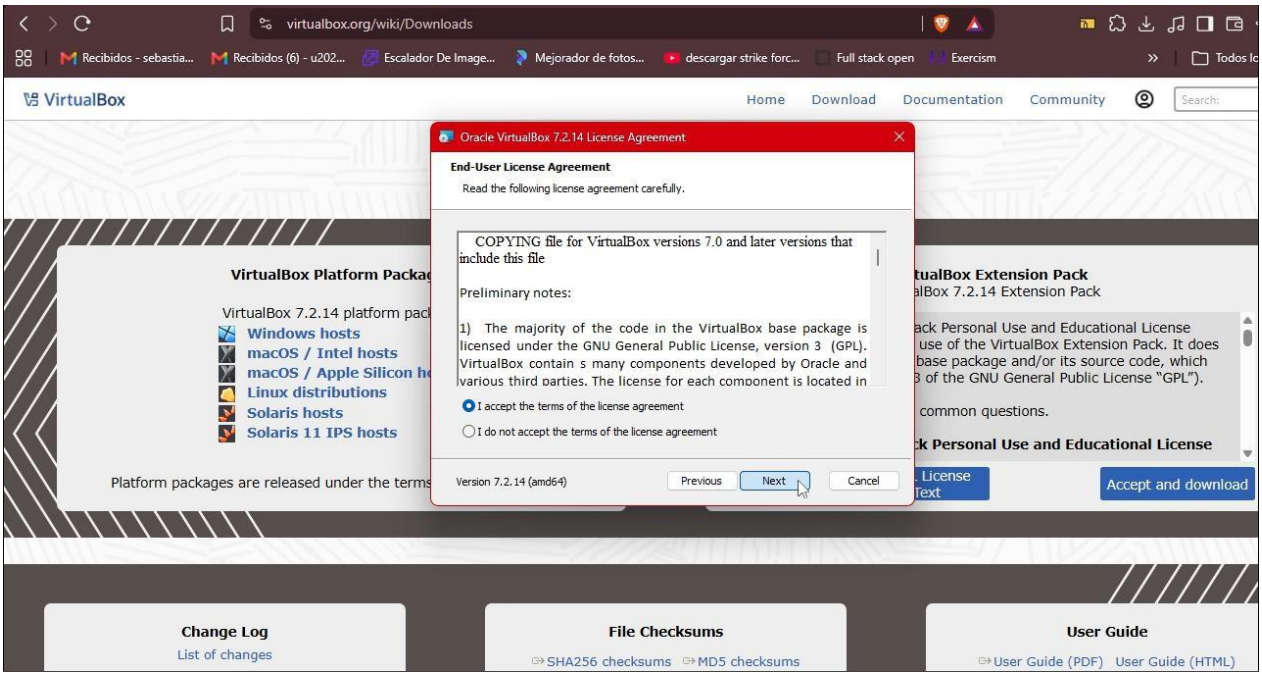
Captura de pantalla – Ventana ejecutable Oracle Virtual box



Nota. Instalado e iniciado el ejecutable de la Oracle VirtualBox.

Figura 4

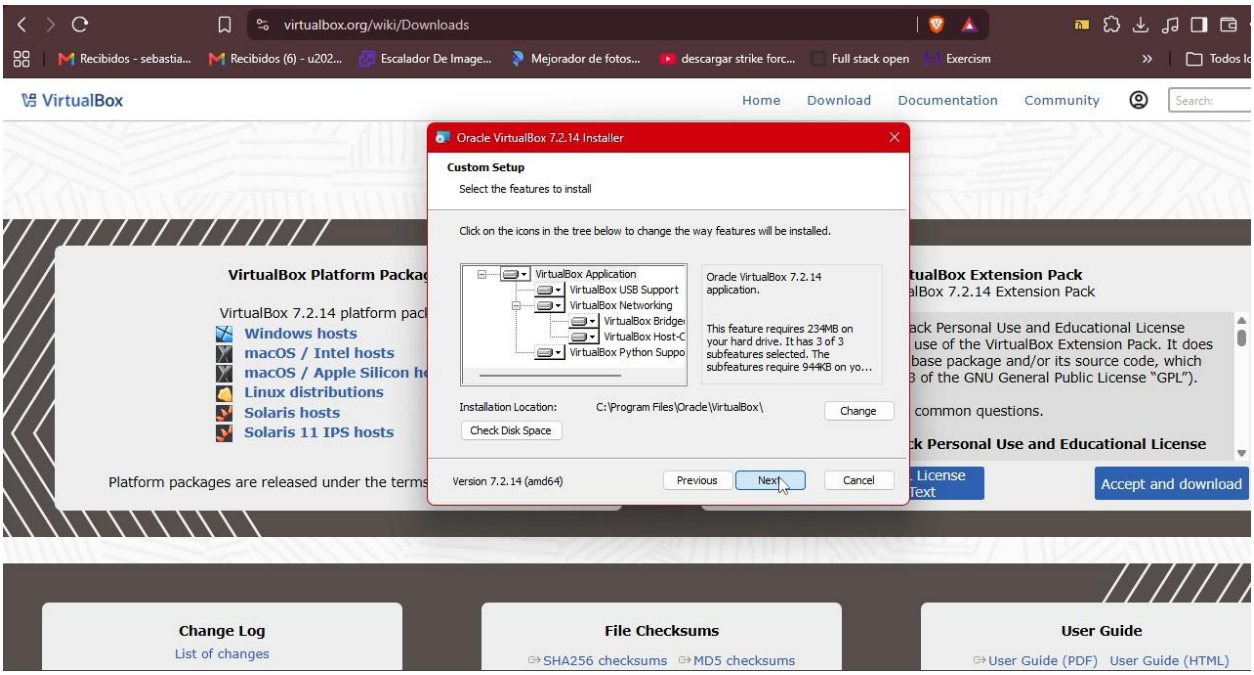
Captura de pantalla – Ventana ejecutable Oracle Virtual box



Nota. Se aceptaron los términos y condiciones del acuerdo de licencia para el uso de la aplicación.

Figura 5

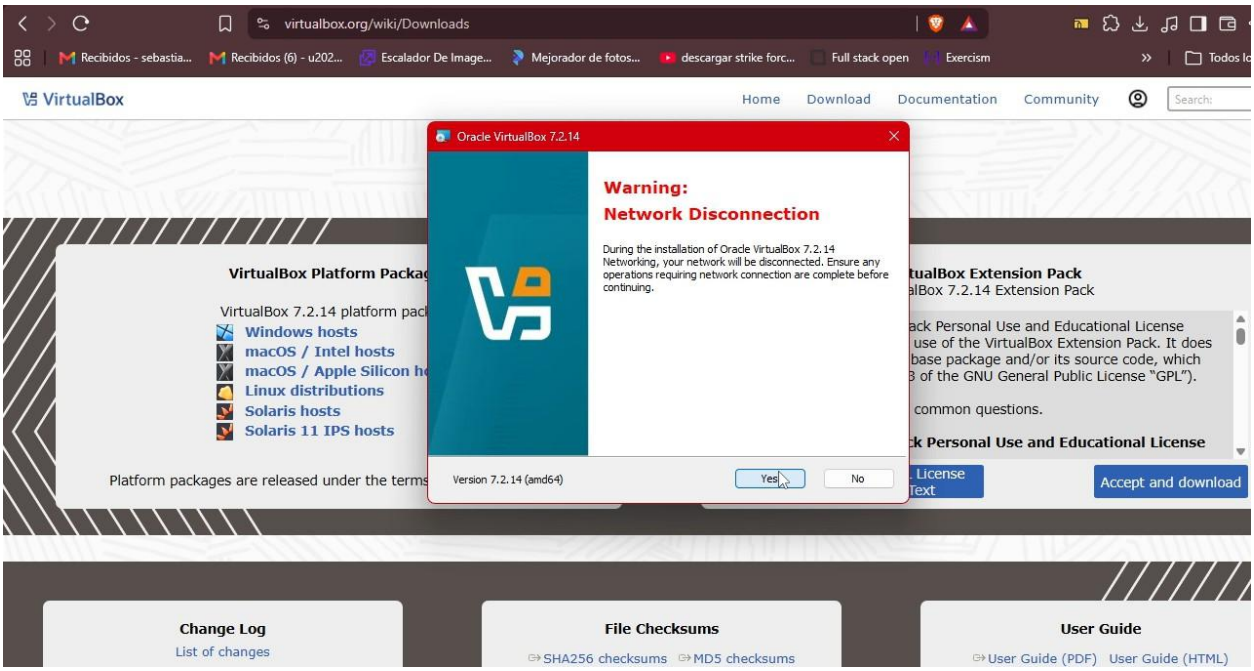
Captura de pantalla – Ventana ejecutable Oracle Virtual box



Nota. Se acepto la dirección de instalación local y todas las carpetas y subcarpetas necesarias para el funcionamiento de la máquina virtual.

Figura 6

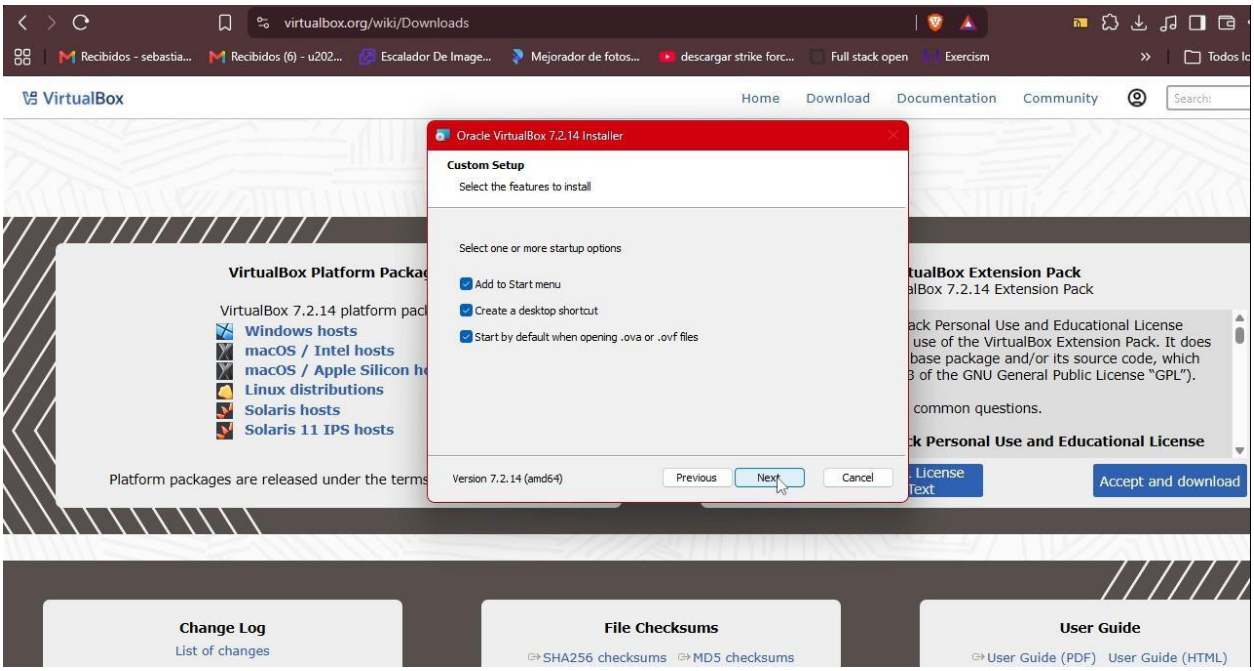
Captura de pantalla – Ventana ejecutable Oracle Virtual box



Nota. Se aceptó la advertencia de la desconexión de red que ocurría al momento de iniciar con la instalación.

Figura 7

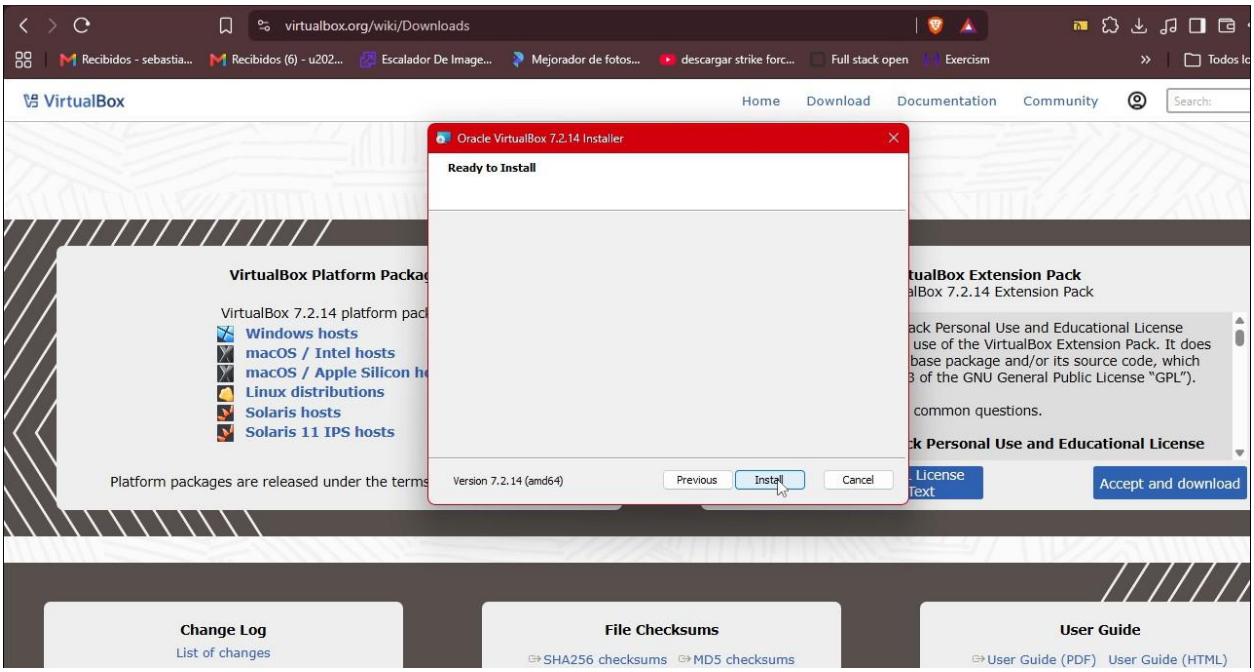
Captura de pantalla – Ventana ejecutable Oracle Virtual box



Nota. Se configuraron los parámetros de configuración personalizada.

Figura 8

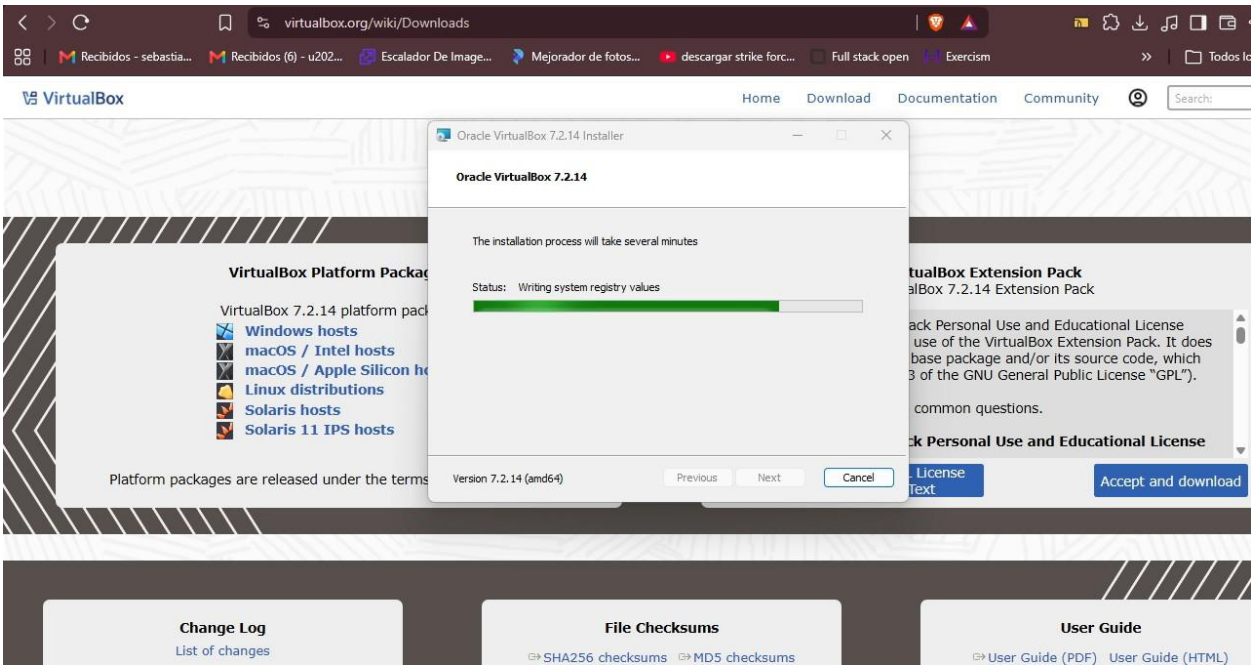
Captura de pantalla – Ventana ejecutable Oracle Virtual box



Nota. Una vez configurados los parámetros de instalación, se procedió a realizar la descarga de la máquina virtual.

Figura 9

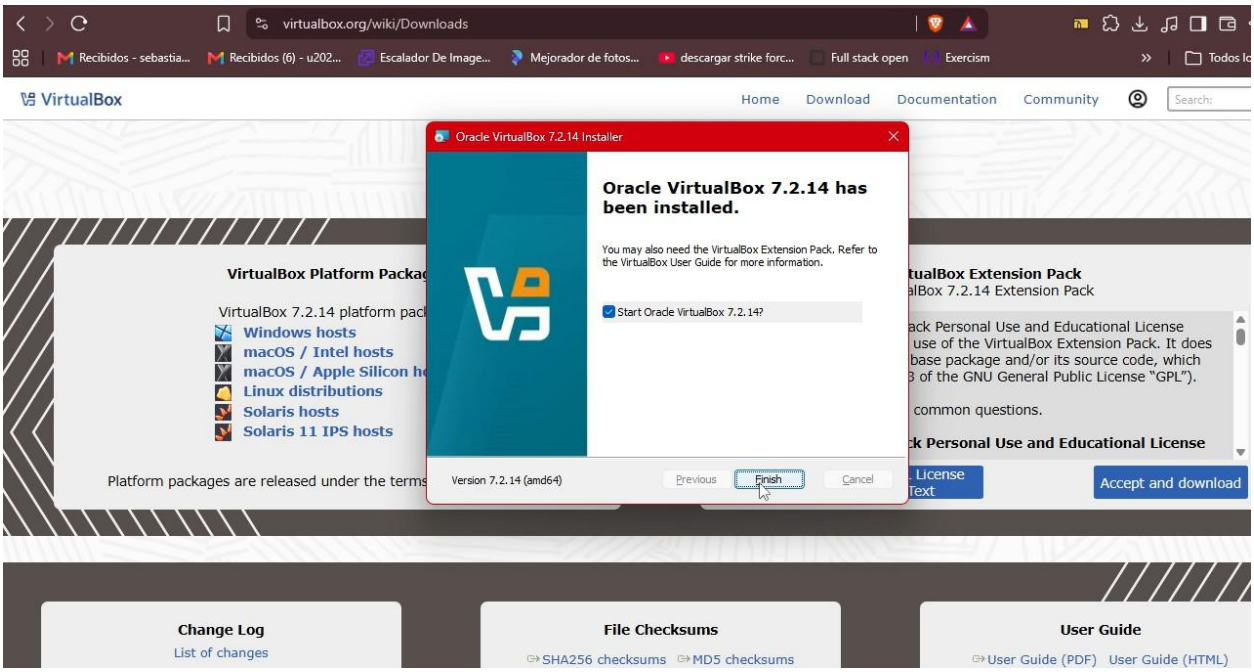
Captura de pantalla – Ventana ejecutable Oracle Virtual box



Nota. Durante la instalación

Figura 10

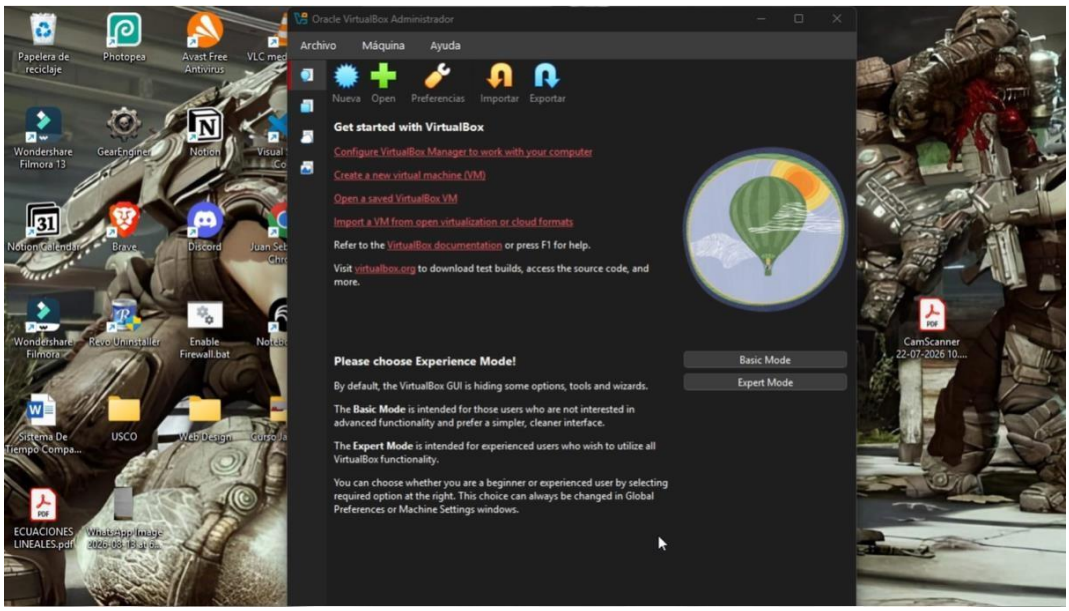
Captura de pantalla – Ventana ejecutable Oracle Virtual box



Nota. Finalización de la instalación

Figura 11

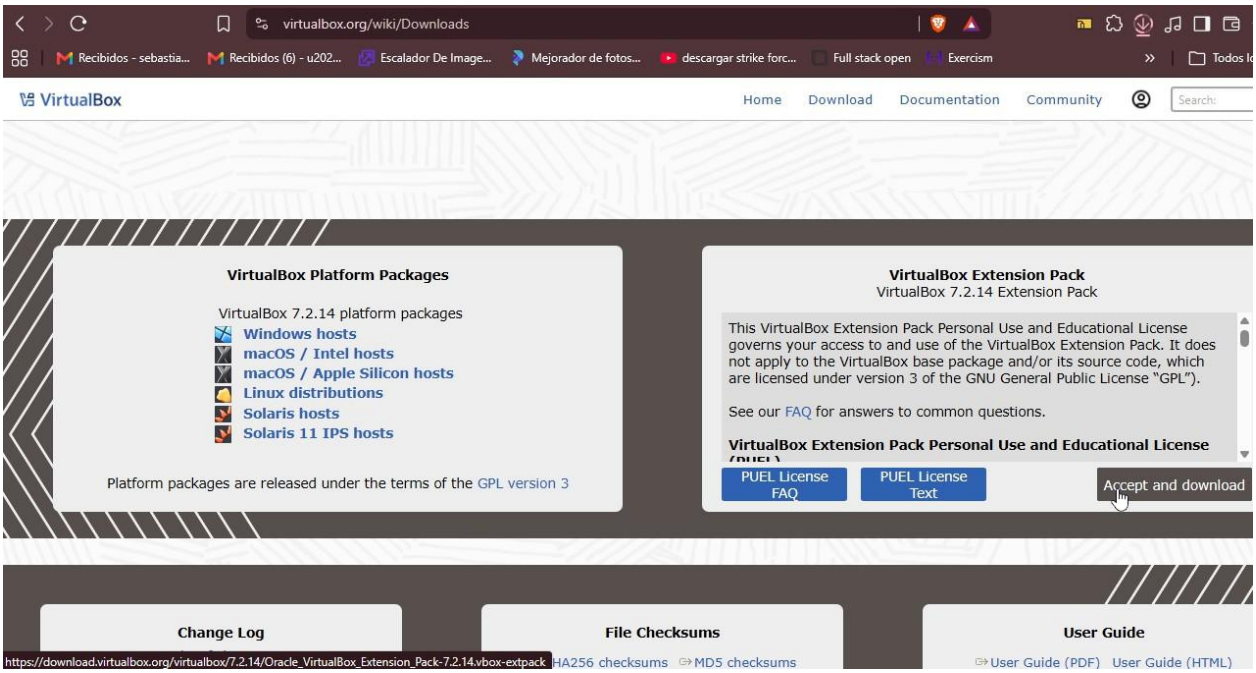
Captura de pantalla – Ventana programa Oracle Virtual box



Nota. Una vez terminada la instalación se procedió a ejecutar la aplicación de Oracle VM Virtual Box, asegurando que la descarga se realizado satisfactoriamente.

Figura 12

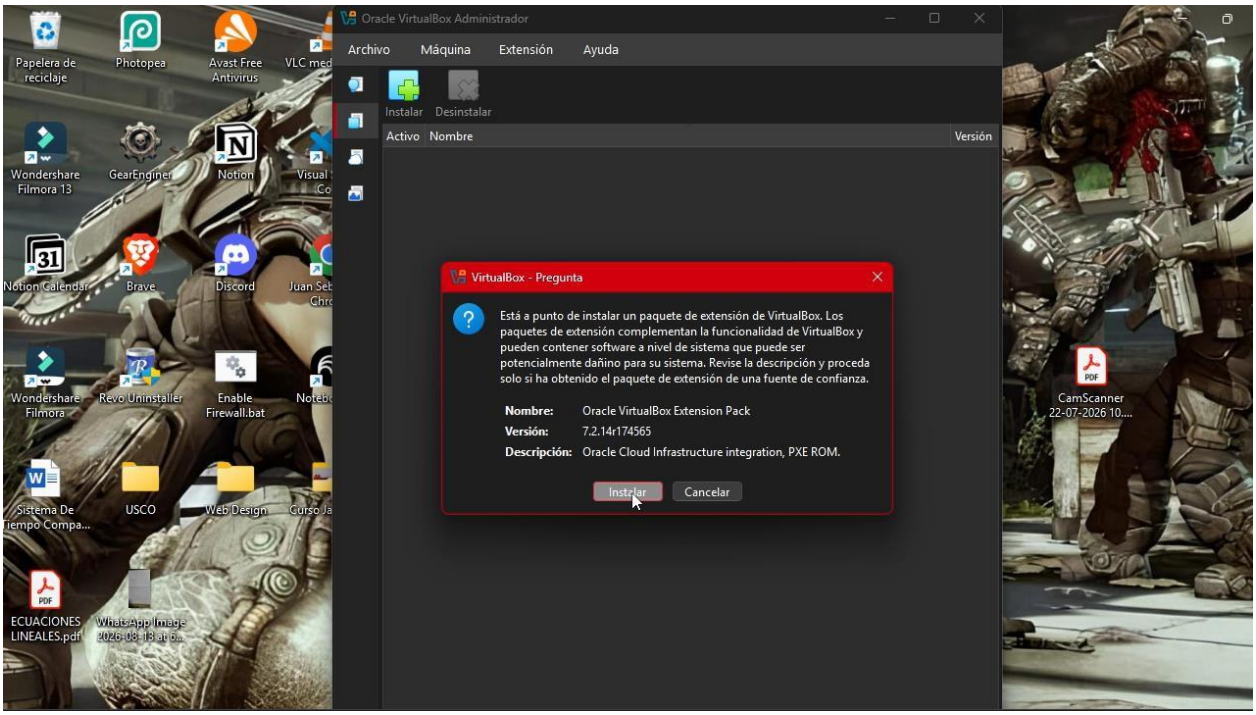
Captura de pantalla – Ventana programa Oracle Virtual box



Nota. Adicionalmente en el sitio oficial del instalador de Oracle Virtual Box se instaló la extensión Pack, que agrega soporte USB 2.0/3.0 y otras funciones.

Figura 13

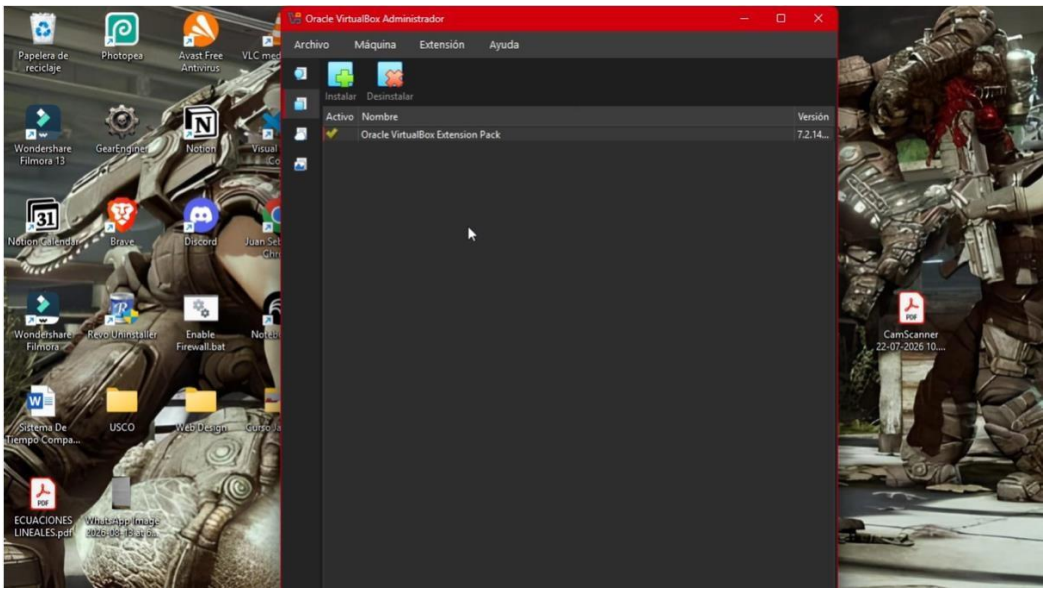
Captura de pantalla – Ventana programa Oracle Virtual box



Nota. Agregando la extensión de contenido al programa de la máquina virtual.

Figura 14

Captura de pantalla – Ventana programa Oracle Virtual box



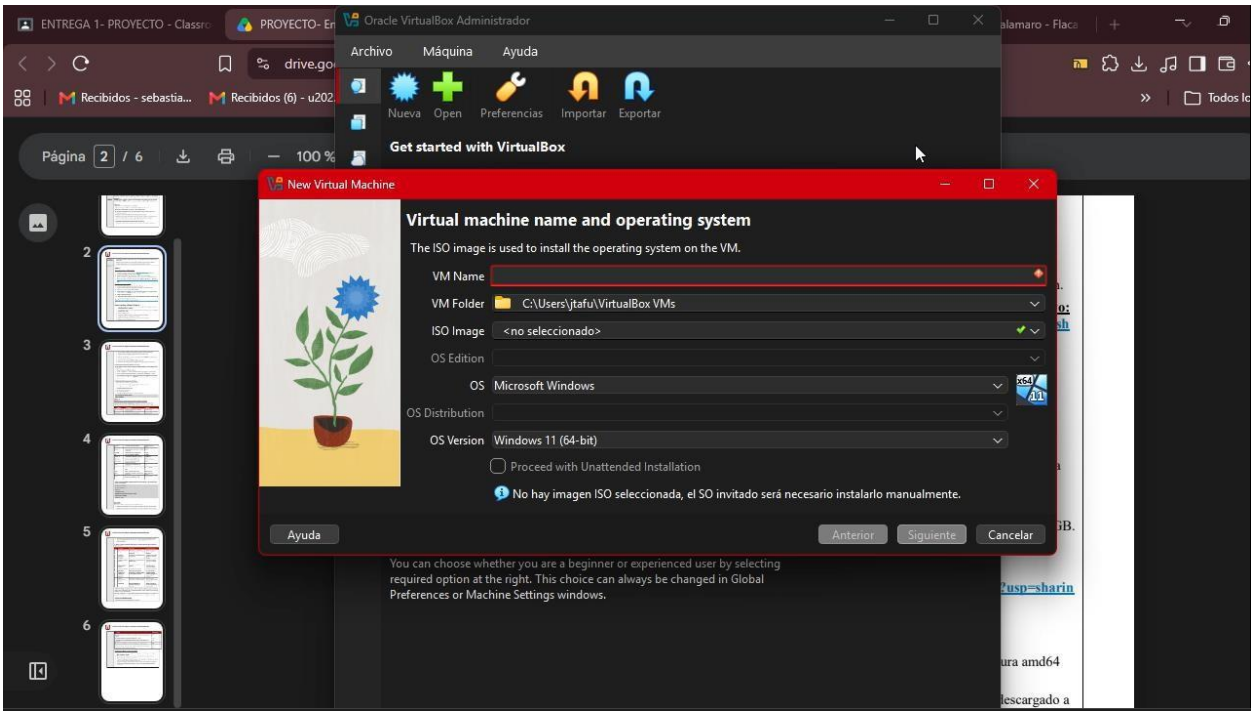
Nota. A continuación, se evidencia la Extension Pack ya agregada.

Creación de la Máquina Virtual

Se procedió a crear una nueva máquina virtual asignando los recursos de hardware requeridos para el correcto funcionamiento del sistema operativo.

Figura 15

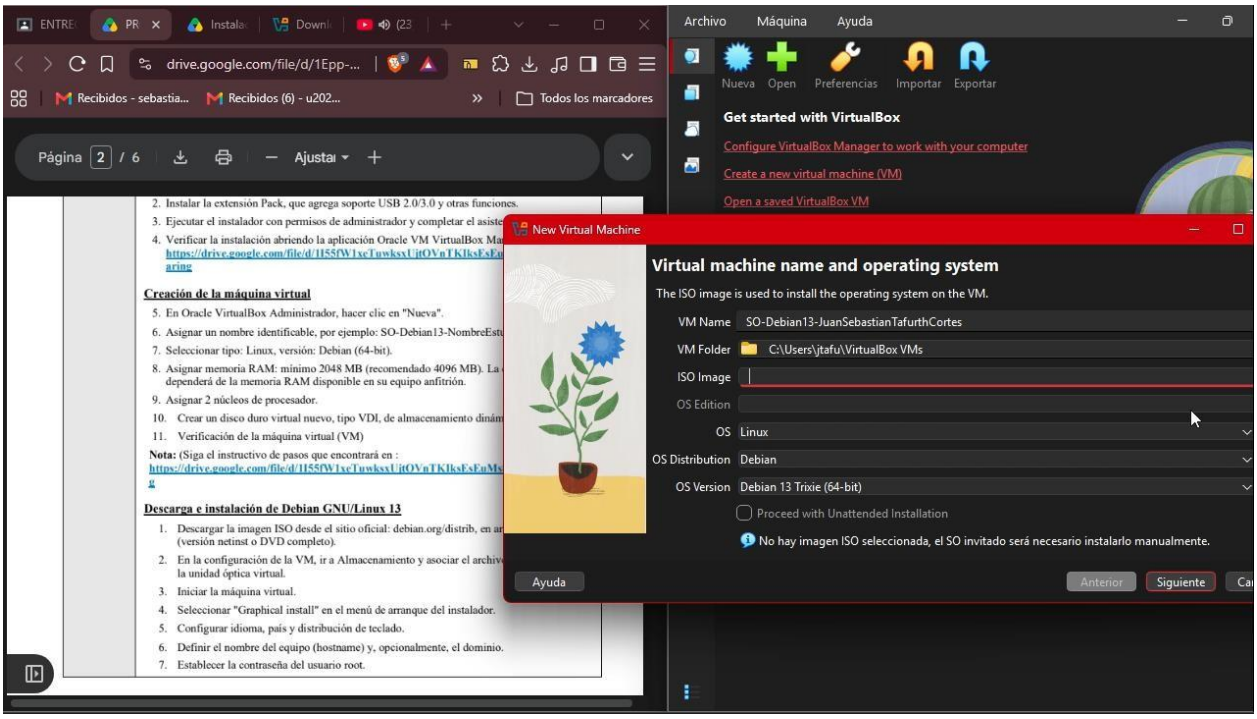
Captura de pantalla – Ventana programa Oracle Virtual box



Nota. Como nombre identificable de la máquina virtual se le asigno SO-Debian13-JuanSebastian, se dejó la dirección predeterminada para la ubicación de la carpeta donde se alojará la VM. El tipo de OS a instalar se insertó “Linux”, la distribución del OS se aplicó la opción “Debian” y por último la versión de la distribución se seleccionó “Debian 13 Trixie (64-bit).

Figura 16

Captura de pantalla – Ventana programa Oracle Virtual box



Nota. A continuación, realizo la asignación de los recursos del computador anfitrión que puede usar la maquia virtual.

- Para la cantidad Memoria RAM se le asigno 3047MB equivalente a 3.04GB.
- Para el número de núcleos del CPU se le asignaron 2 núcleos.
- Para la cantidad de espacio en el disco para ocupar se le asigno 20GB

Figura 17

Captura de pantalla – Ventana programa Oracle Virtual box

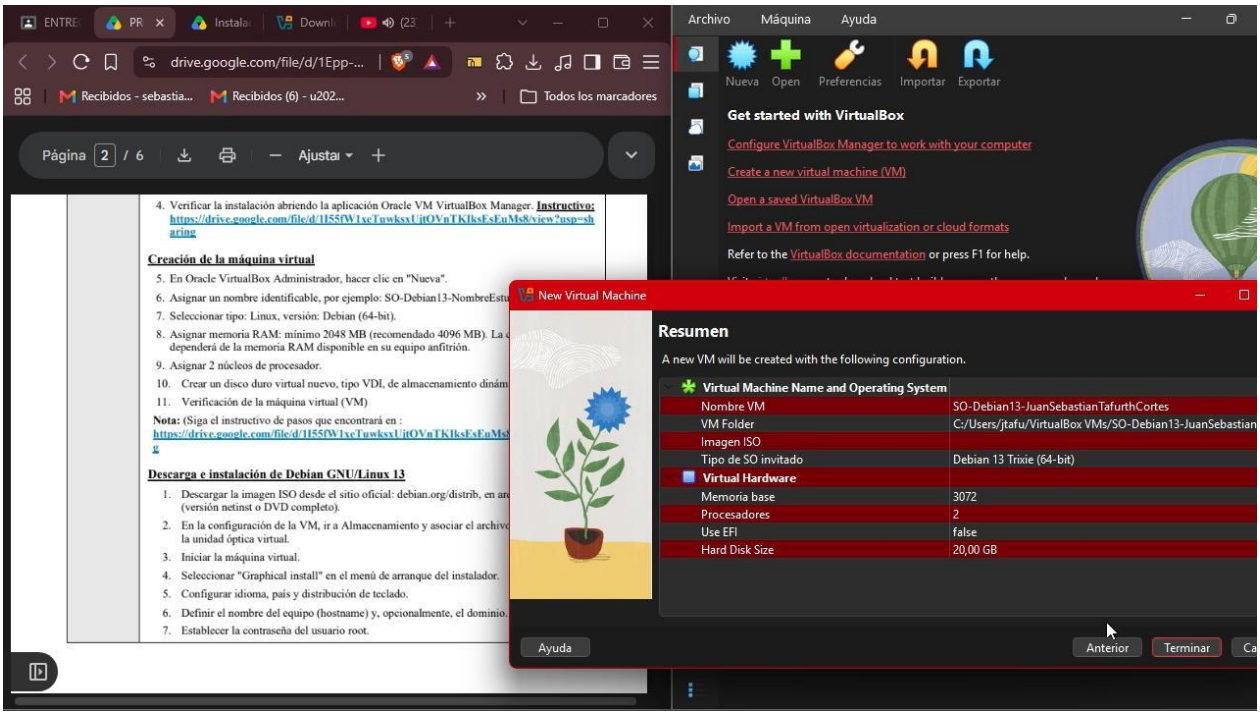
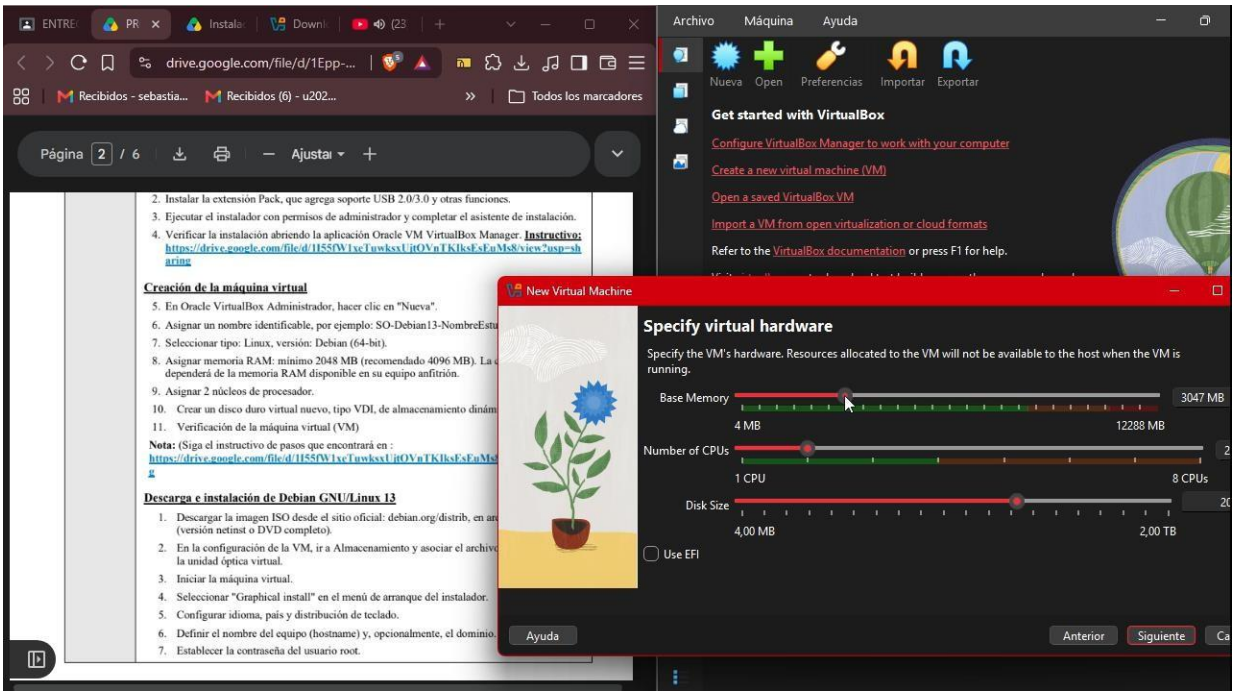


Figura 18

Captura de pantalla – Ventana programa Oracle Virtual box

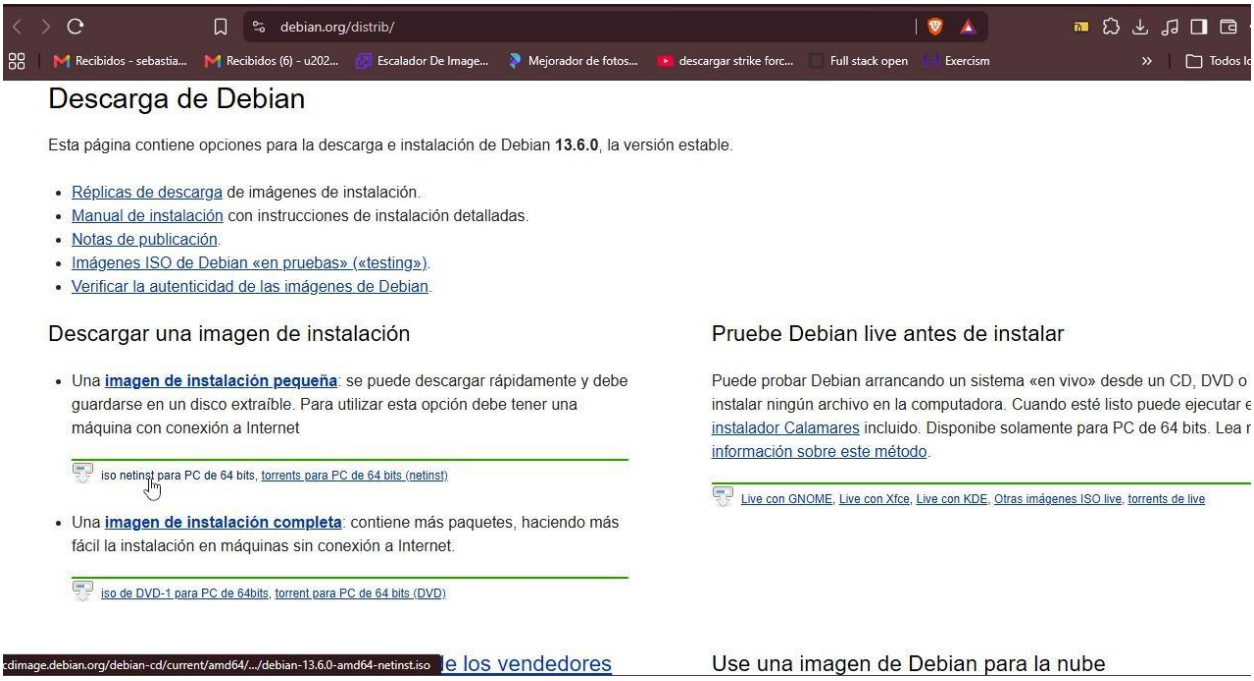


Descarga e Instalación de Debian GNU/Linux 13

Se descargo la imagen ISO oficial de Debian GNU/Linux 13 (arquitectura amd64) desde el sitio oficial de la distribución de Debian https://www.debian.org/distrib/.

Figura 19

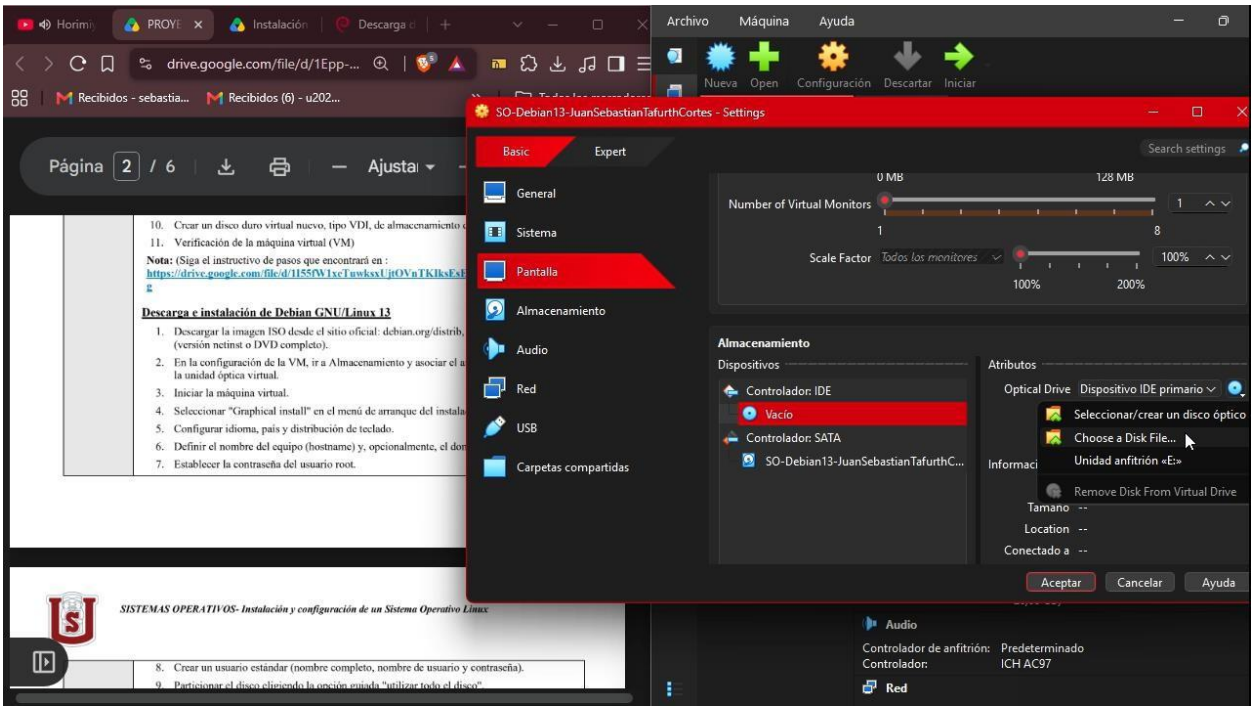
Captura de pantalla – Ventana programa Oracle Virtual box



Nota. Se descargo la imagen ISO oficial de Debian GNU/Linux 13 (arquitectura amd64) desde el sitio oficial de la distribución de Debian https://www.debian.org/distrib/.

Figura 20

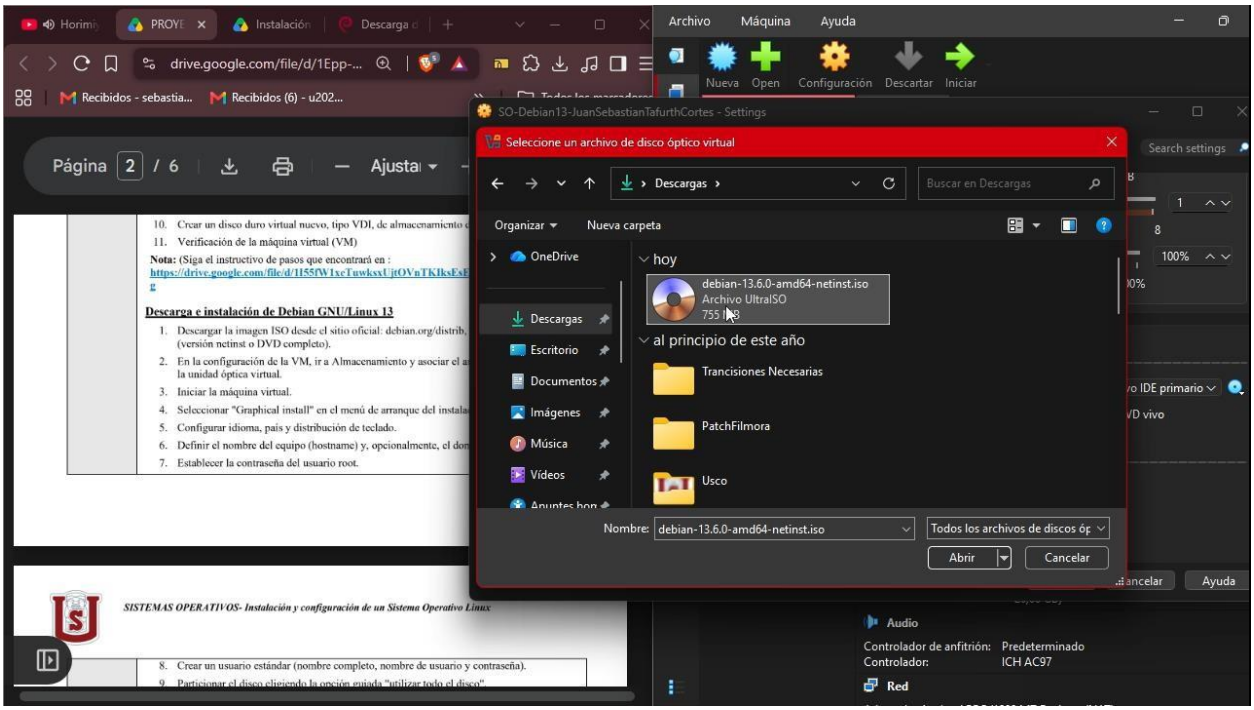
Captura de pantalla – Ventana programa Oracle Virtual box



Nota. Posteriormente de la descarga de la imagen hizo se agregó a la máquina virtual, para esto se accedió a la configuración de la VM creada yendo a la sección de almacenamiento y en la opción de Atributos agregando la Imagen ISO de debian.

Figura 21

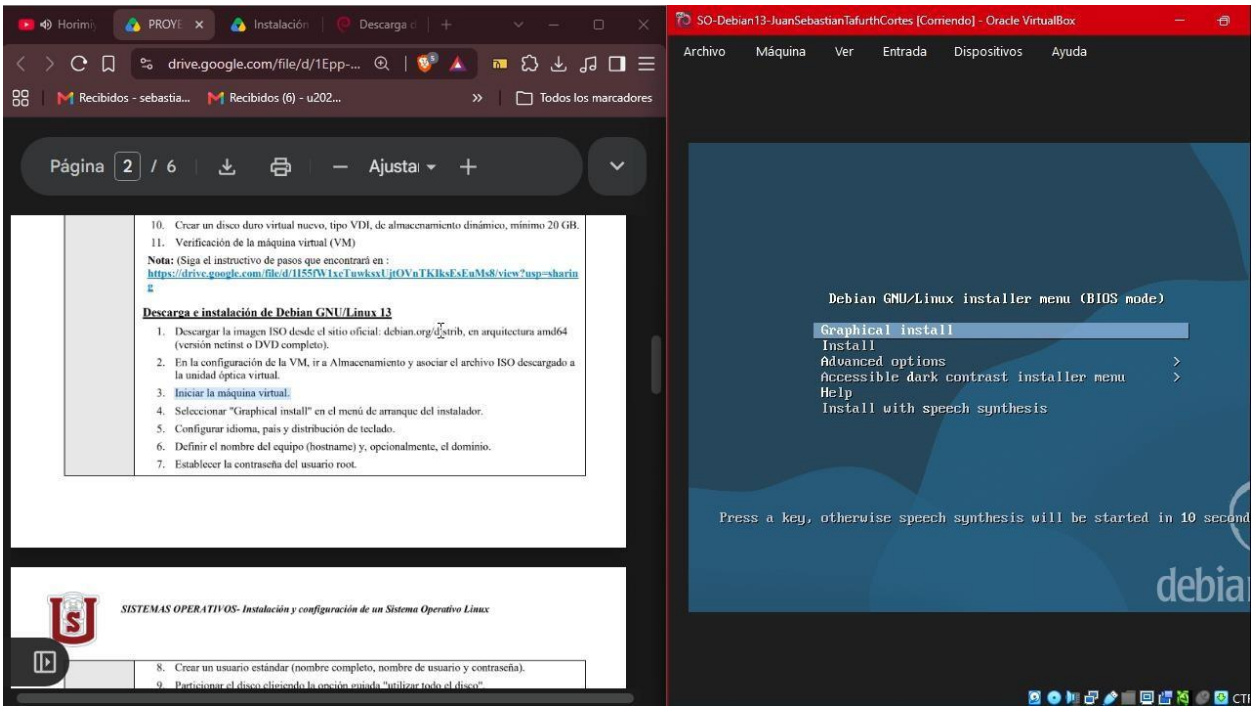
Captura de pantalla – Ventana programa Oracle Virtual box



Instalación del sistema operativo Debian

Figura 22

Captura de pantalla – Ventana Instalación Debian, Oracle Virtual box

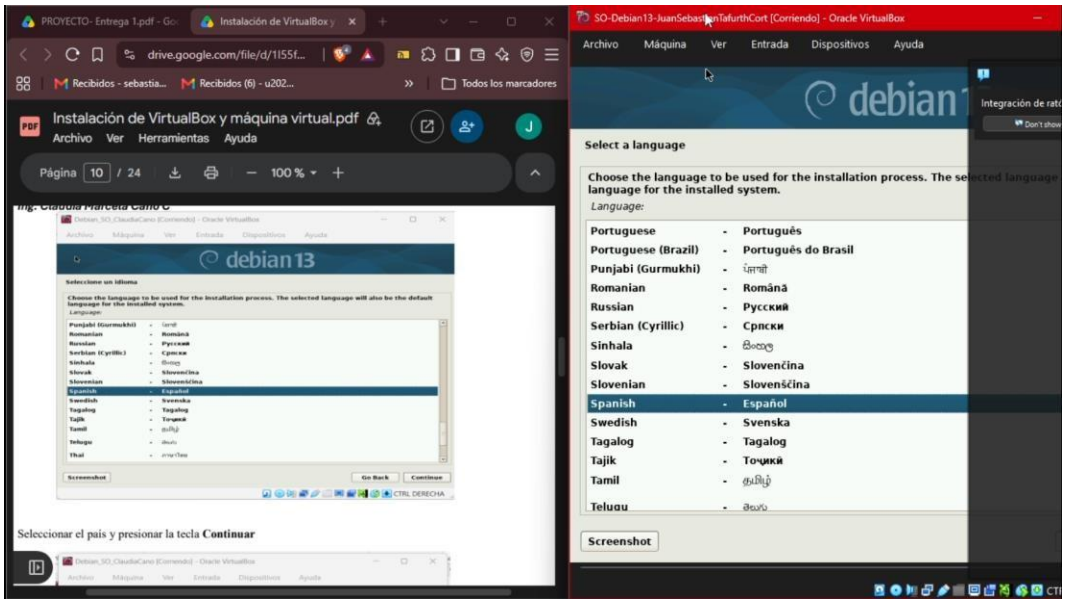


Nota. Una vez configurados todos los parámetros de la máquina virtual se llevó a cabo la instalación del Sistema operativo de la máquina virtual. En primera instancia iniciando la máquina virtual, una vez iniciado el entorno de instalación se seleccionó la opción de “Graphical install”

- **Selección de lenguaje**

Figura 23

Captura de pantalla – Ventana Instalación Debian, Oracle Virtual box



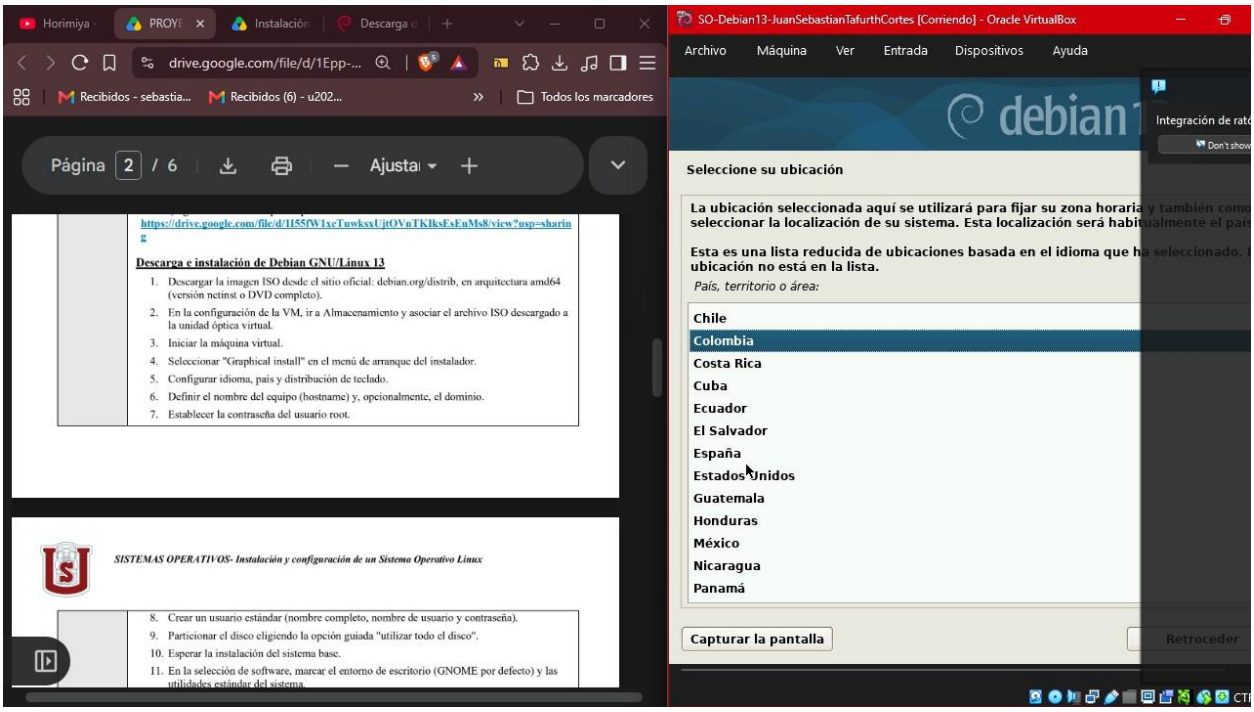
Nota. Una vez seleccionado la instalación gráfica, el sistema nos redirigió al selector de idioma, seleccionando el español.

En la selección del pais de ubicación se escogió la opción Colombia.

- **Selección de ubicación**

Figura 24

Captura de pantalla – Ventana Instalación Debian, Oracle Virtual box

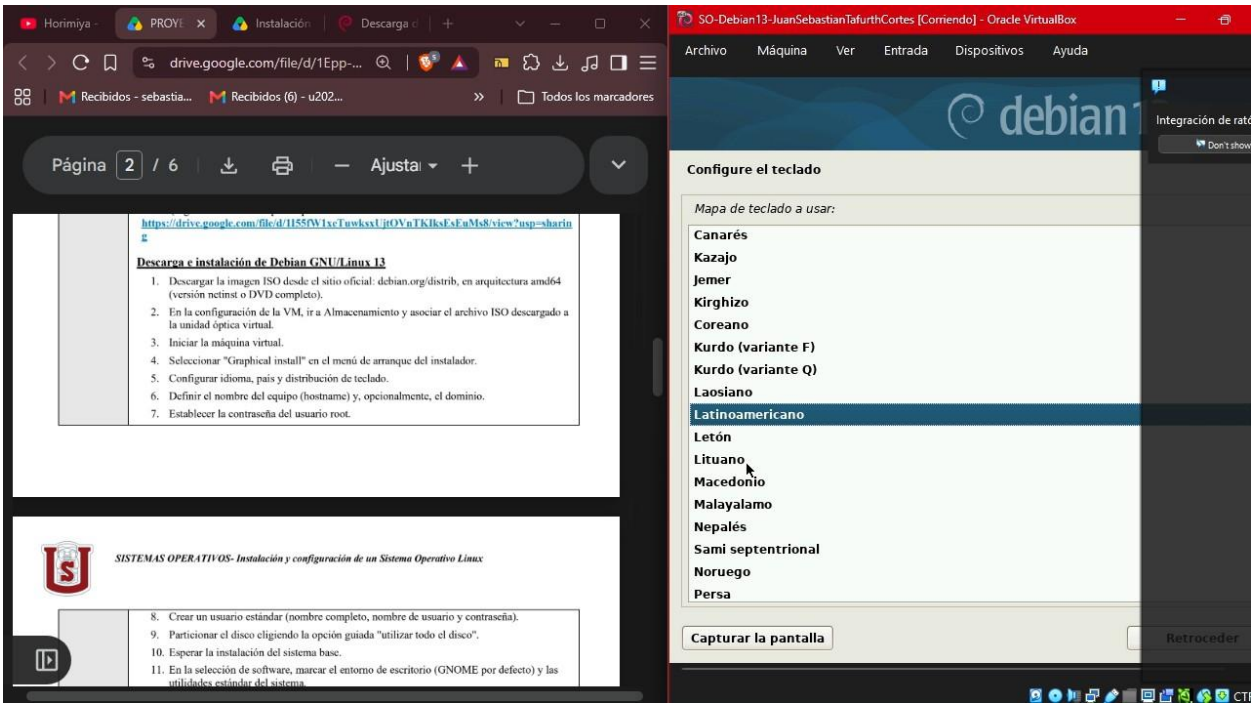


Nota. En la selección del pais de ubicación se escogió la opción Colombia.

- **Configuración de teclado**

Figura 25

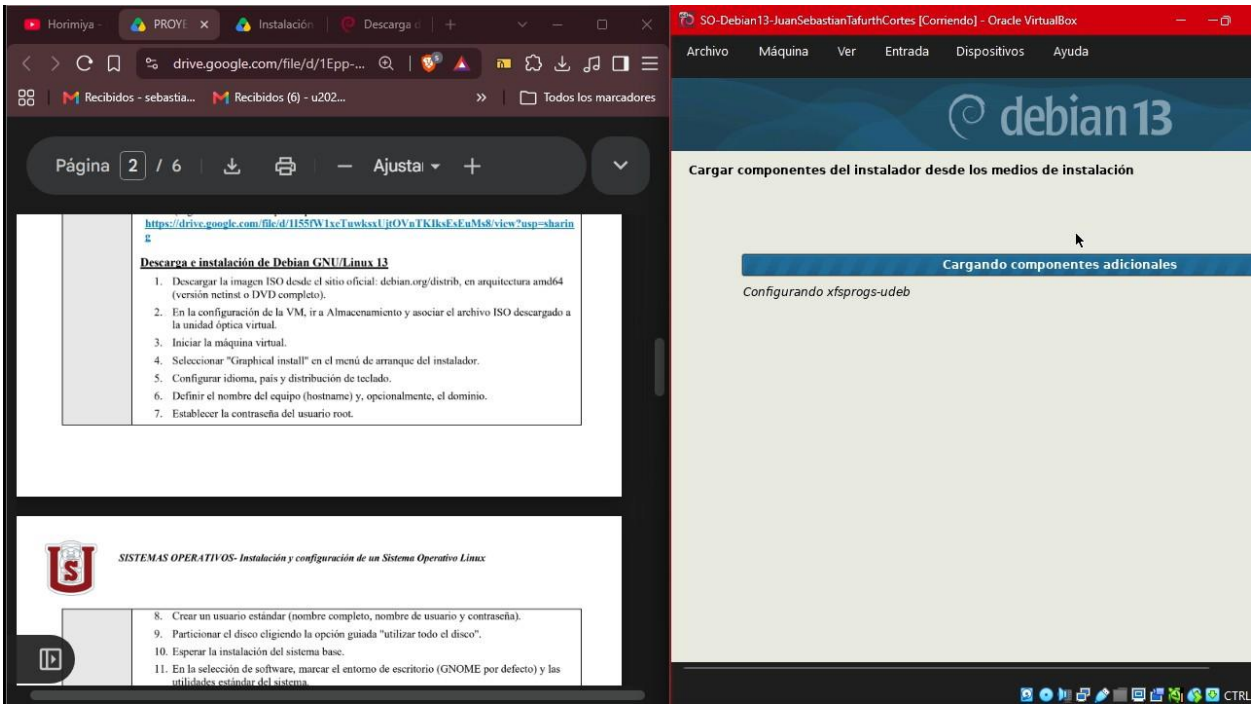
Captura de pantalla – Ventana Instalación Debian, Oracle Virtual box



Nota. Para la configuración del teclado se seleccionó la opción Latinoamericano.

Figura 26

Captura de pantalla – Ventana Instalación Debian, Oracle Virtual box

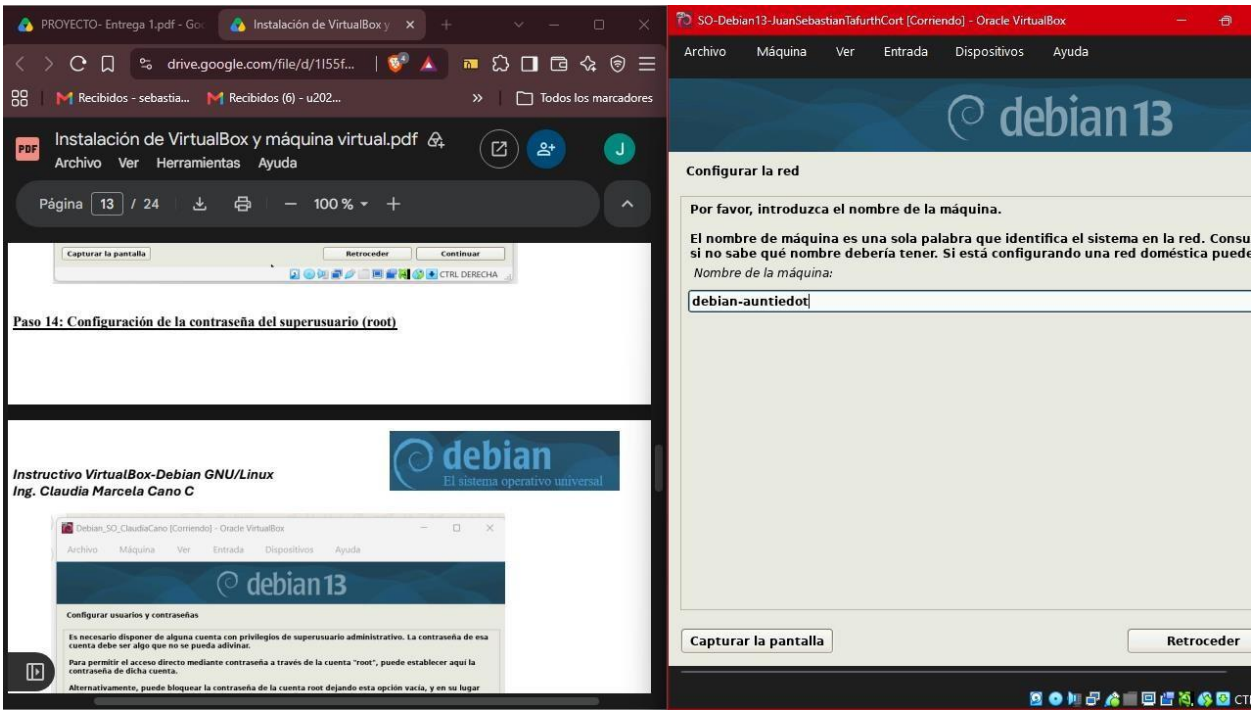


Nota. Al completar la configuración realizada anteriormente se empezaron a cargar los componentes del instalador desde los medios de instalación.

- **Configuración de red:**

Figura 27

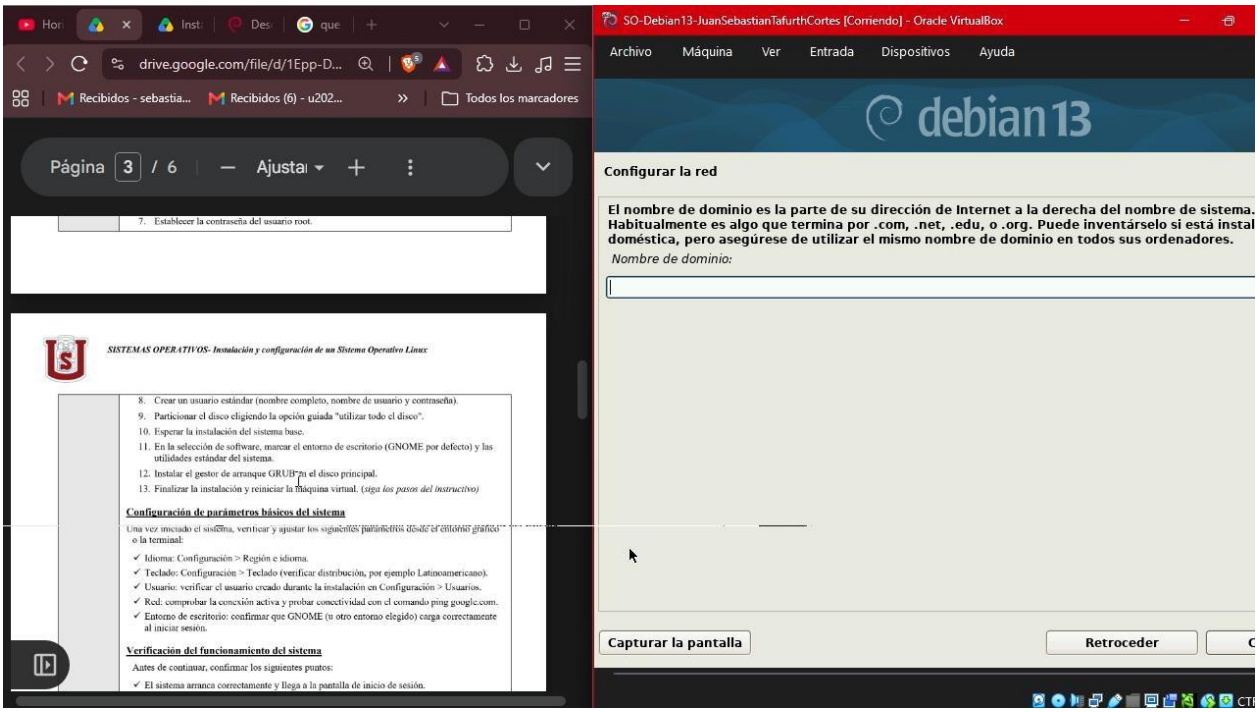
Captura de pantalla – Ventana Instalación Debian, Oracle Virtual box



Nota. Una vez finalizada la carga de los componentes del instalador, se cambió a la configuración de la red, pidiendo primero introducir el nombre que tendría la máquina virtual. El asignado fue “Debian-auntiedot”.

Figura 28

Captura de pantalla – Ventana Instalación Debian, Oracle Virtual box

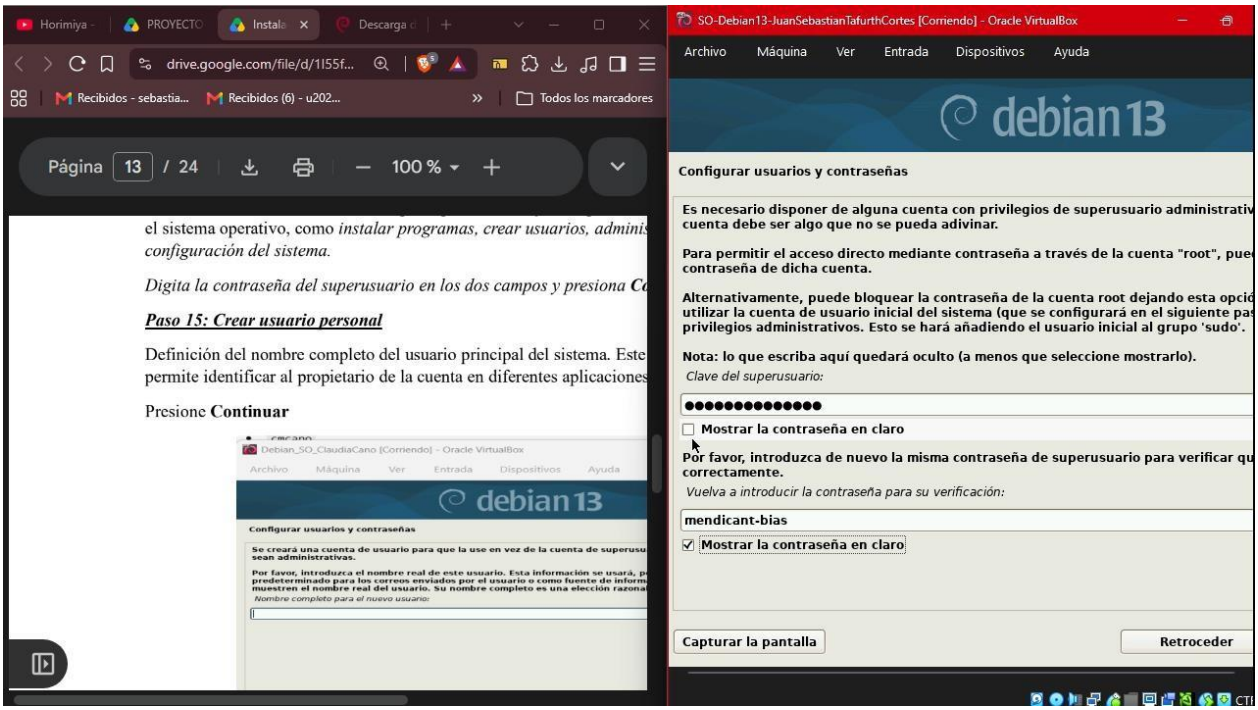


Nota. para el nombre del dominio no se completó el input, dejándolo en blanco y continuando a la otra sección.

- **Configuración usuarios y contraseñas:**

Figura 29

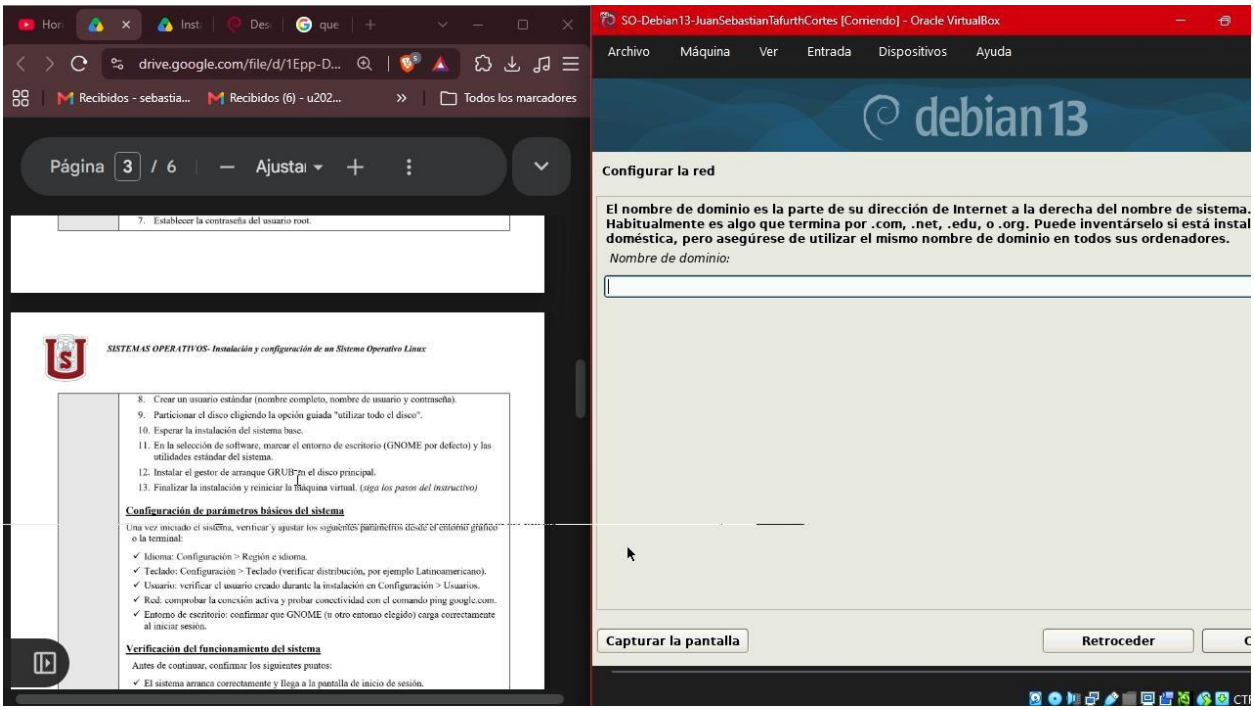
Captura de pantalla – Ventana Instalación Debian, Oracle Virtual box



Nota. En primera instancia se digito la contraseña del Super usuario / root, siendo esta mendicant-bias.

Figura 30

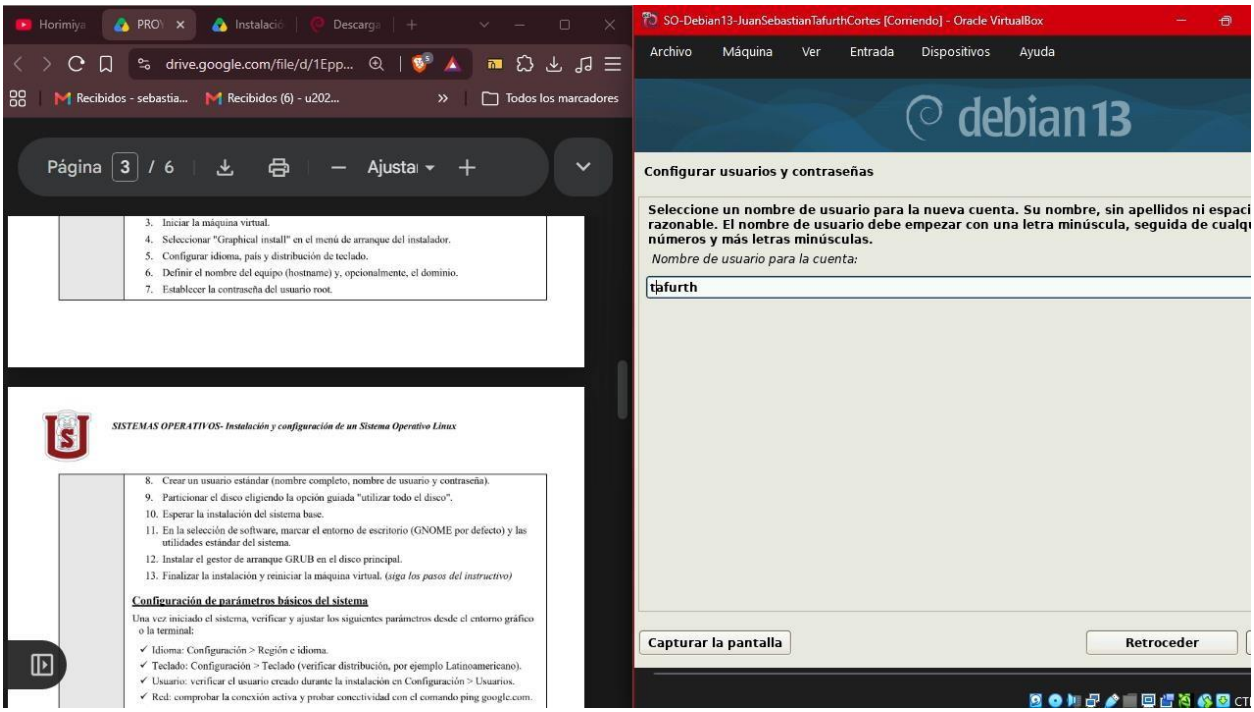
Captura de pantalla – Ventana Instalación Debian, Oracle Virtual box



Nota. En segunda se procedió con la creación del usuario del sistema operativo, pidiendo nombre completo. El introducido fue “Juan Sebastian Tafurth Cortes”.

Figura 31

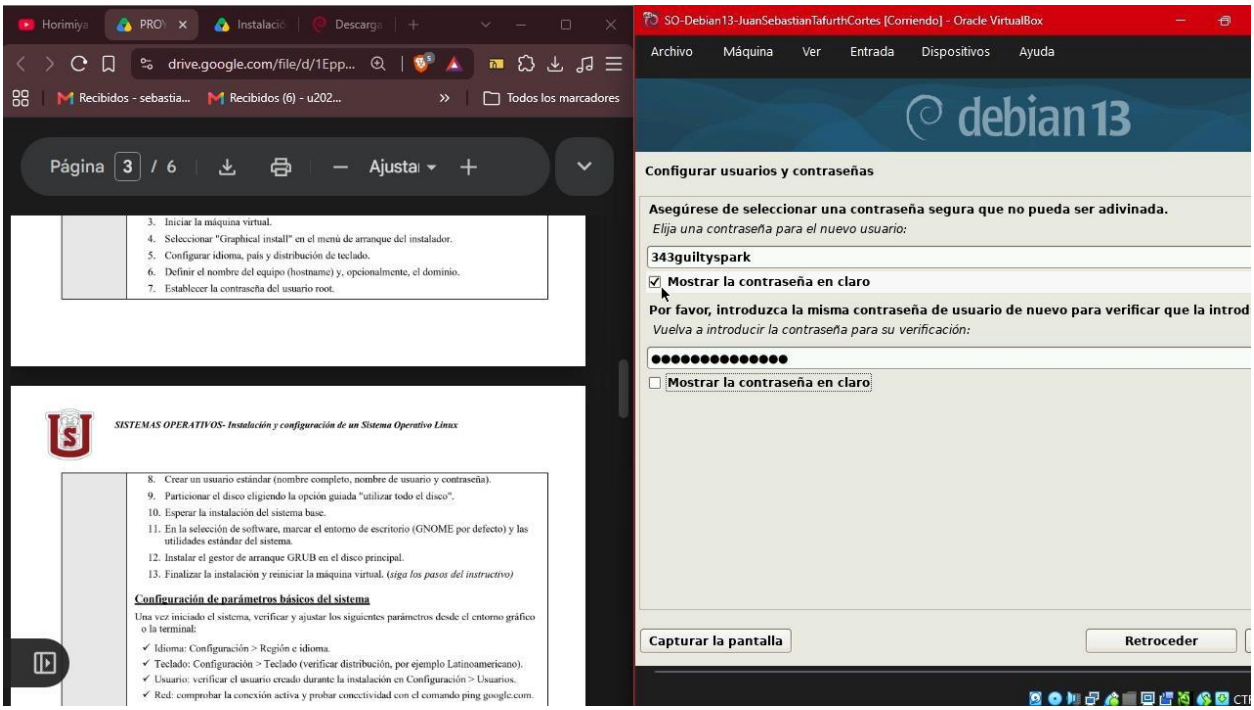
Captura de pantalla – Ventana Instalación Debian, Oracle Virtual box



Nota. A continuación, se ingresó la palabra clave para el nombre del usuario, la palabra digitada para el usuario fue “tafurth”

Figura 32

Captura de pantalla – Ventana Instalación Debian, Oracle Virtual box

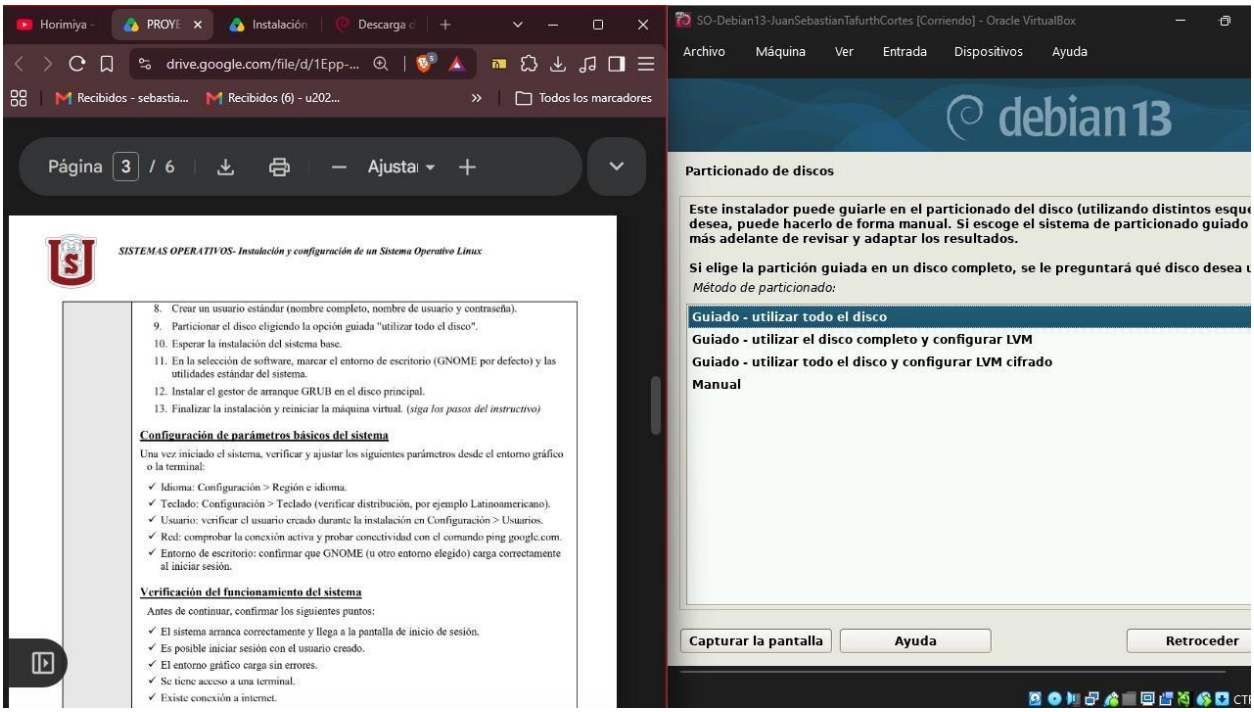


Nota. Para la contraseña del usuario creado se introdujo la clave “343guiltyspark”

- **Partición de discos:**

Figura 33

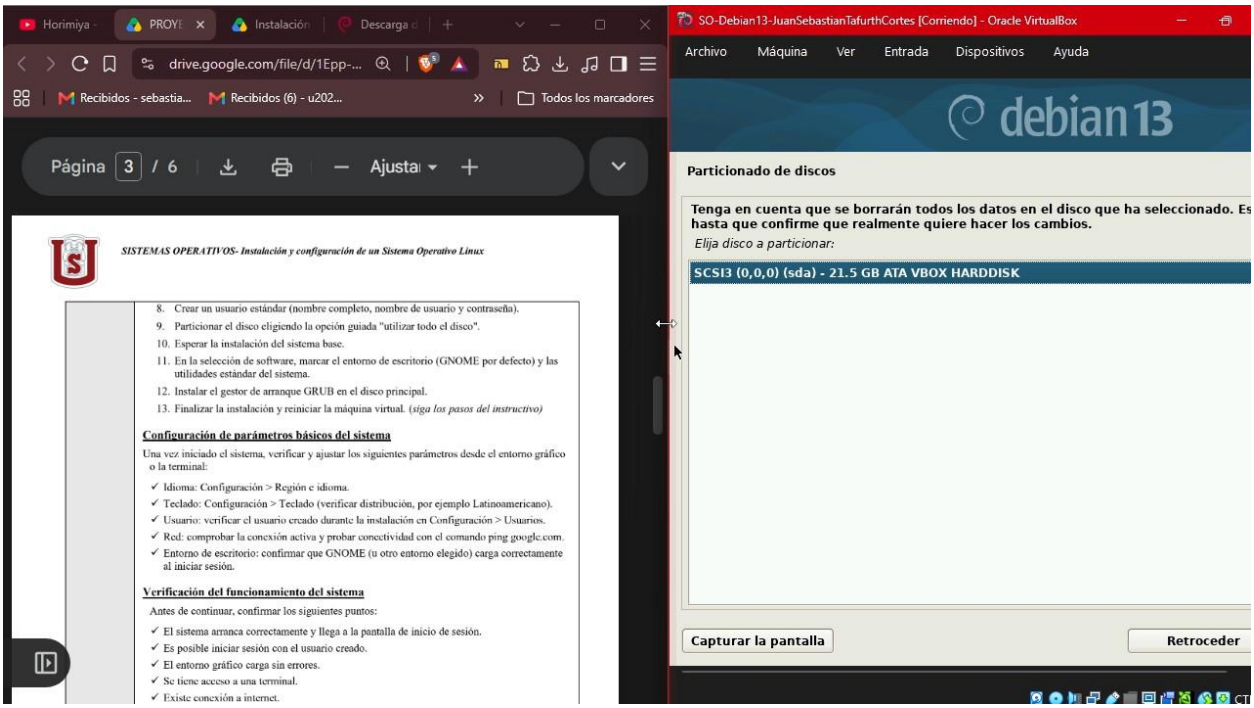
Captura de pantalla – Ventana Instalación Debian, Oracle Virtual box



Nota. Con respecto a la configuración de la partición de discos se seleccionó la opción “Guiado – utilizar todo el disco”.

Figura 34

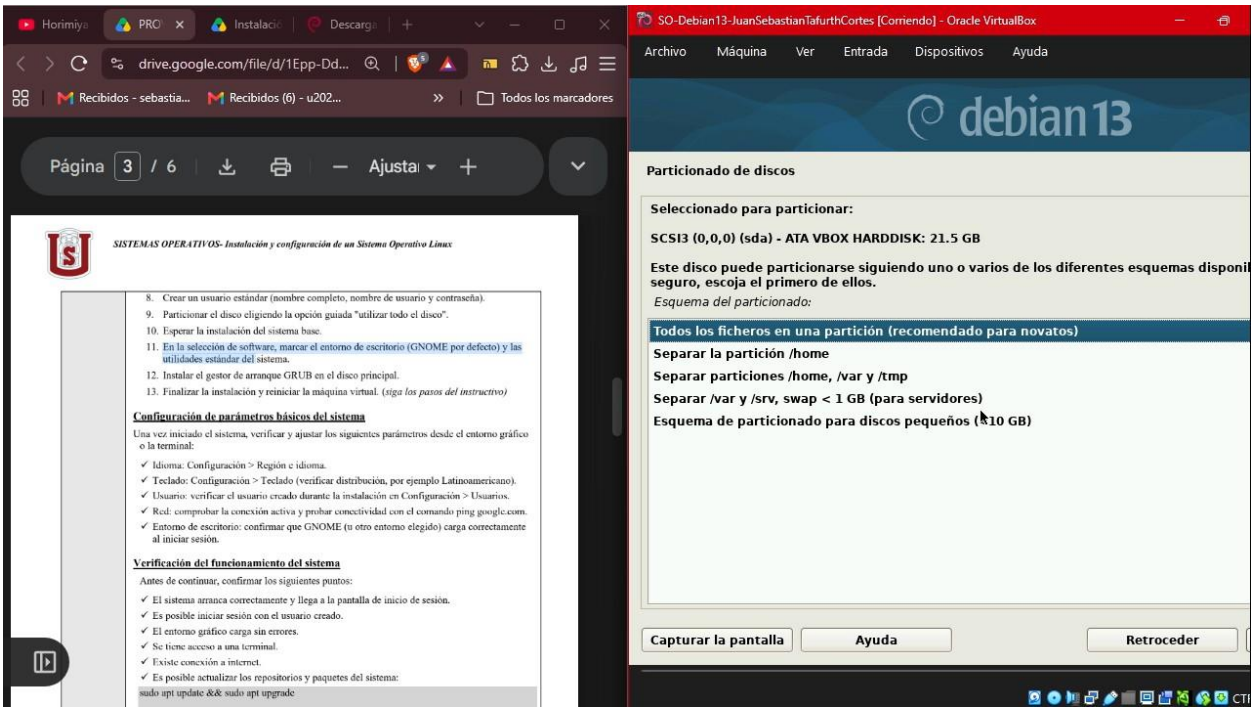
Captura de pantalla – Ventana Instalación Debian, Oracle Virtual box



Nota. Asimismo, el disco elegido para particionar fue el “SCI3 (0,0,0) (sda) – 21.5 GB VBOX HARDISK” el disco virtual creado por el programa de Oracle Virtual Box.

Figura 35

Captura de pantalla – Ventana Instalación Debian, Oracle Virtual box

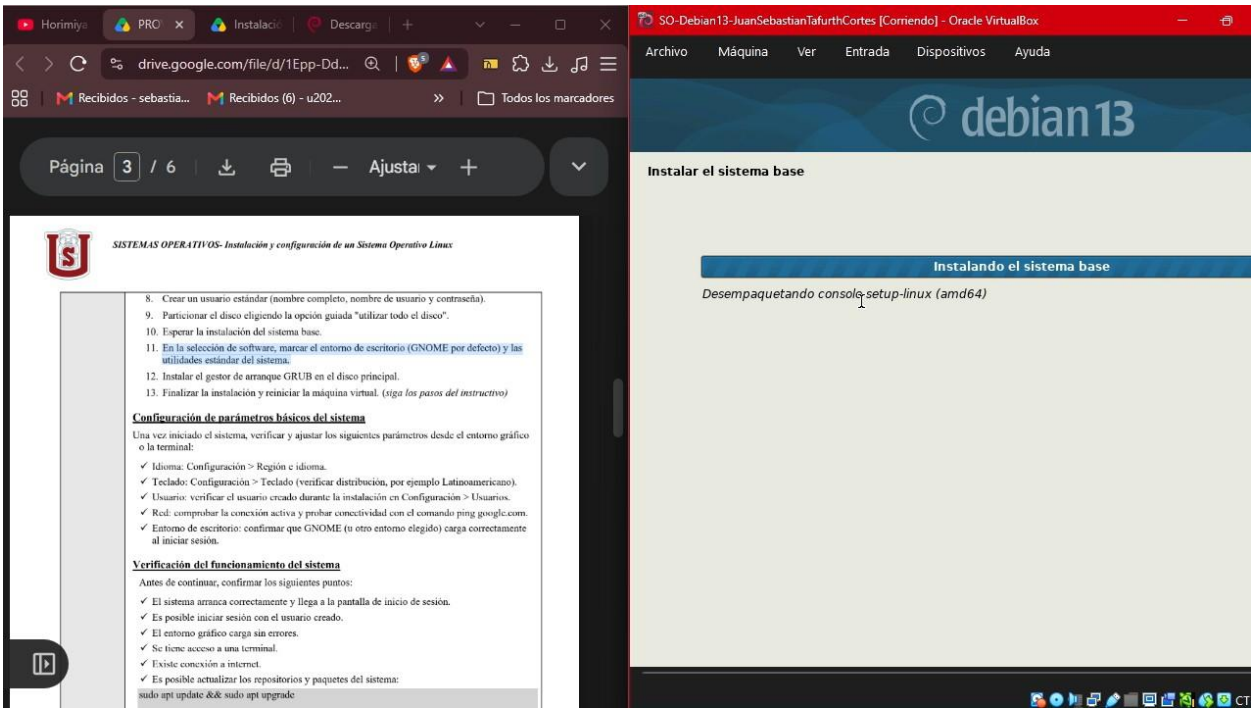


Nota. Conscientemente el esquema de partición seleccionado para el disco duro virtual fue el de "todos los ficheros en una partición" siendo, como lo dice el propio programa, el recomendado para principiantes y finalmente procediendo con la partición de discos.

- **Instalación sistema base**

Figura 36

Captura de pantalla – Ventana Instalación Debian, Oracle Virtual box

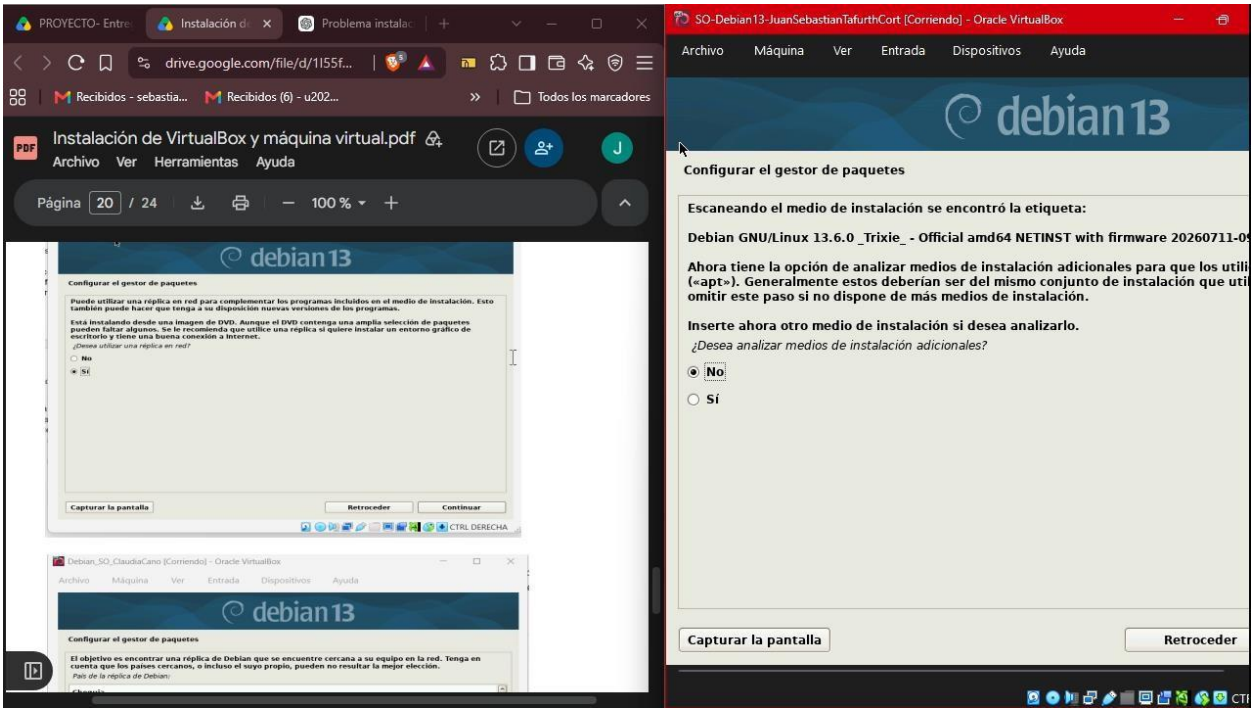


Nota. Con la finalización de la partición del disco el sistema procedió a instalar el sistema base.

- **Configuración del gestor de paquetes:**

Figura 37

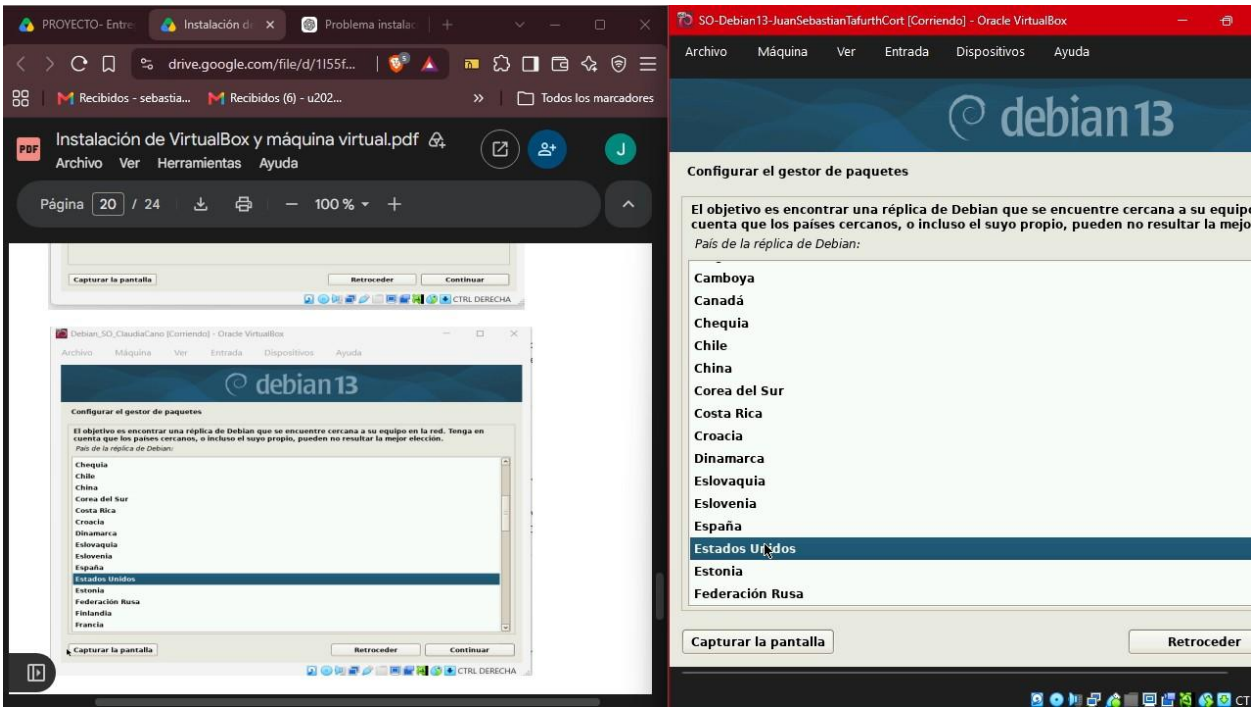
Captura de pantalla – Ventana Instalación Debian, Oracle Virtual box



Nota. Empezando con la configuración del gestor de paquetes el sistema el sistema dio la opción de analizar medios de instalación adicionales para paquetes que no están dentro de la instalación principal. Al no requerir paquetes externos se seleccionó la opción “No”.

Figura 38

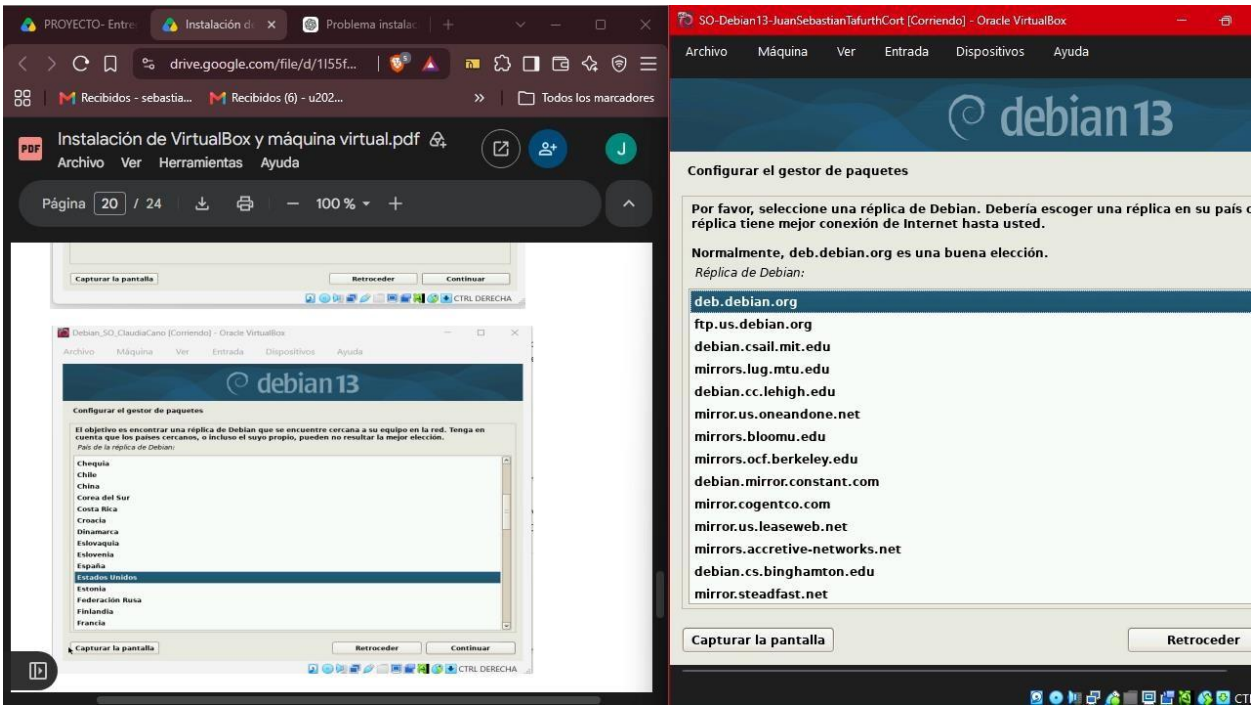
Captura de pantalla – Ventana Instalación Debian, Oracle Virtual box



Nota. A continuación, se seleccionó el pais más cercano a la ubicación del anfitrión, al no estar la opción de Colombia se optó por la opción de Estados unidos.

Figura 39

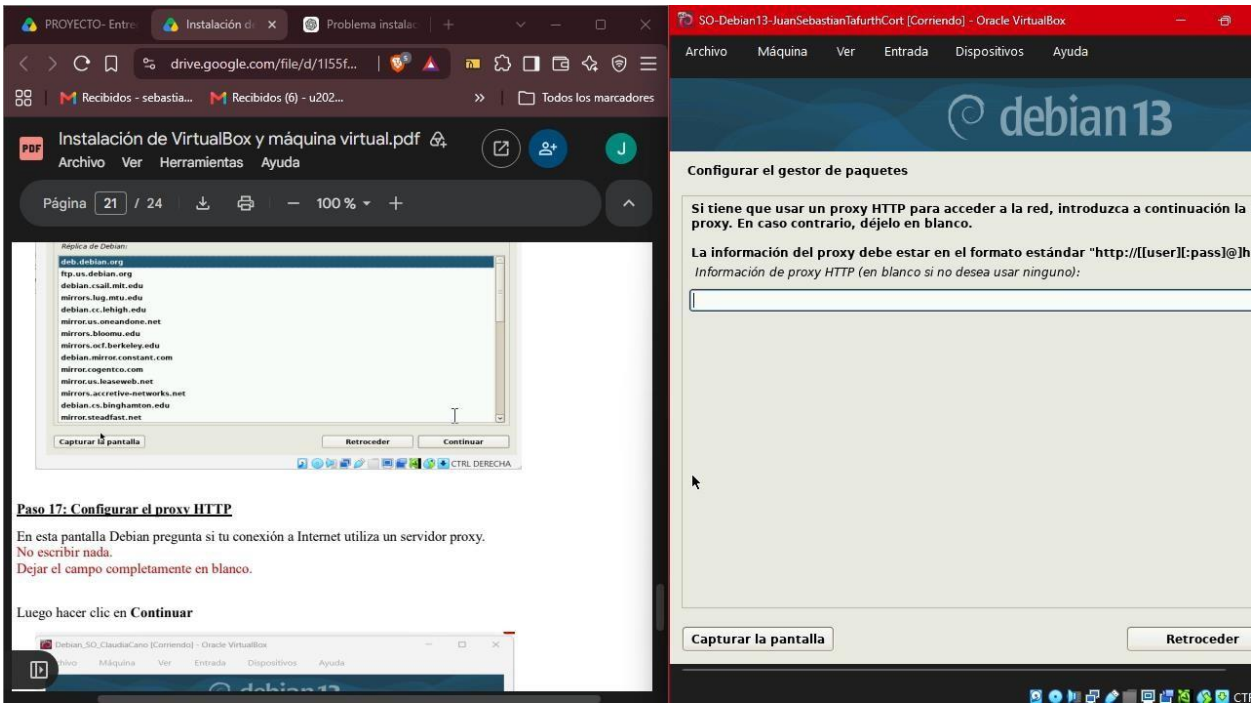
Captura de pantalla – Ventana Instalación Debian, Oracle Virtual box



Nota. Para la réplica de debian se seleccionó la recomendada de deb.debian.org.

Figura 40

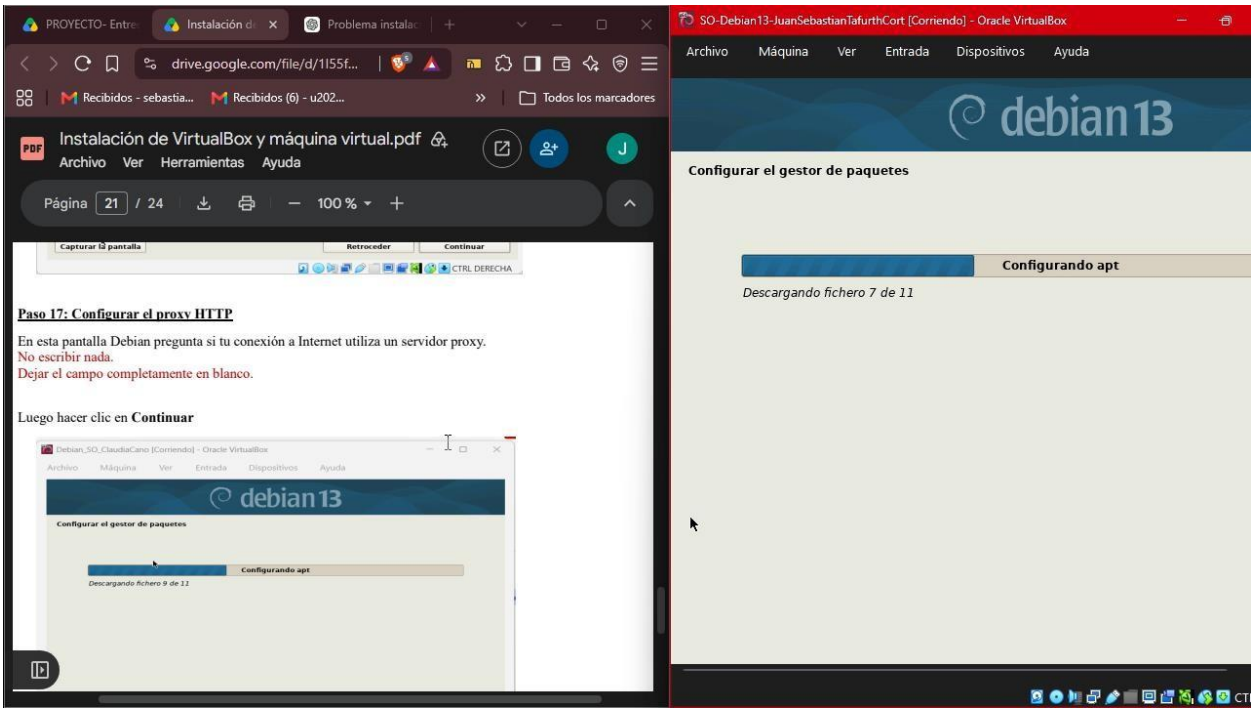
Captura de pantalla – Ventana Instalación Debian, Oracle Virtual box



Nota. No se digito ninguna opción para el Proxy HTTP, se dejó el espacio en blanco y se continuo.

Figura 41

Captura de pantalla – Ventana Instalación Debian, Oracle Virtual box

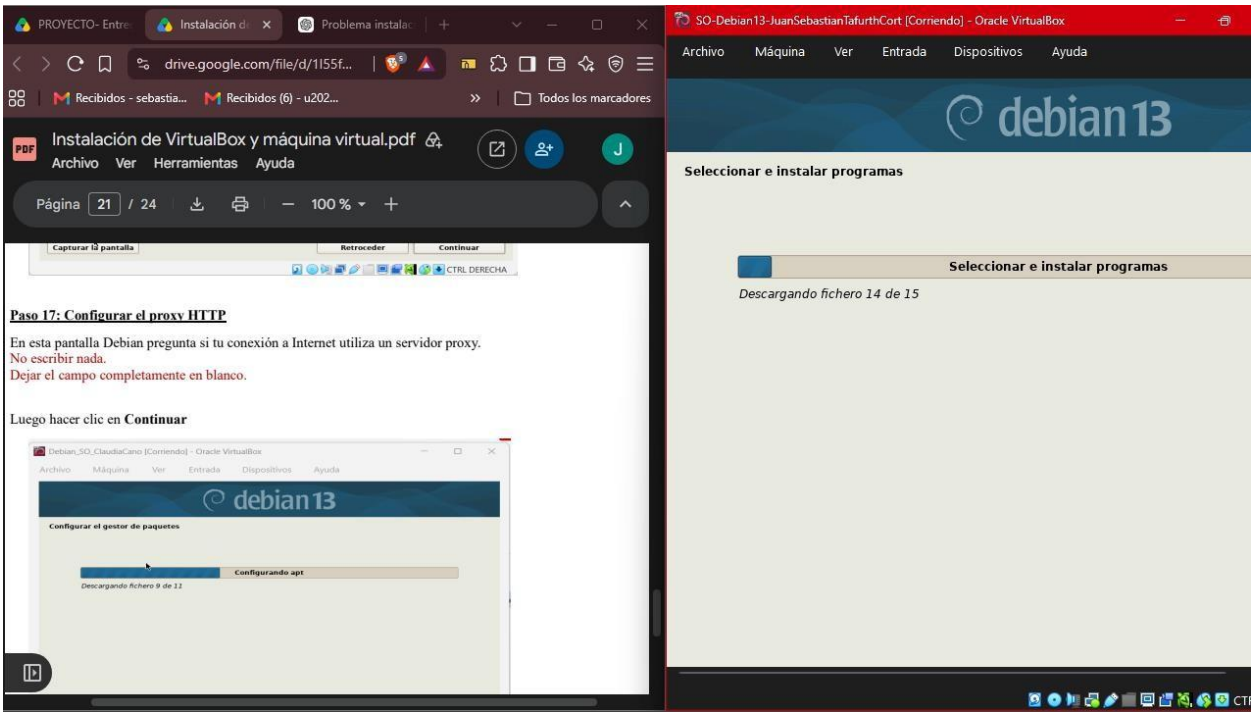


Nota. De esta manera el sistema aplico todos los cambios a la configuracion del gesto de paquetes.

- **Selección e instalación de programas**

Figura 42

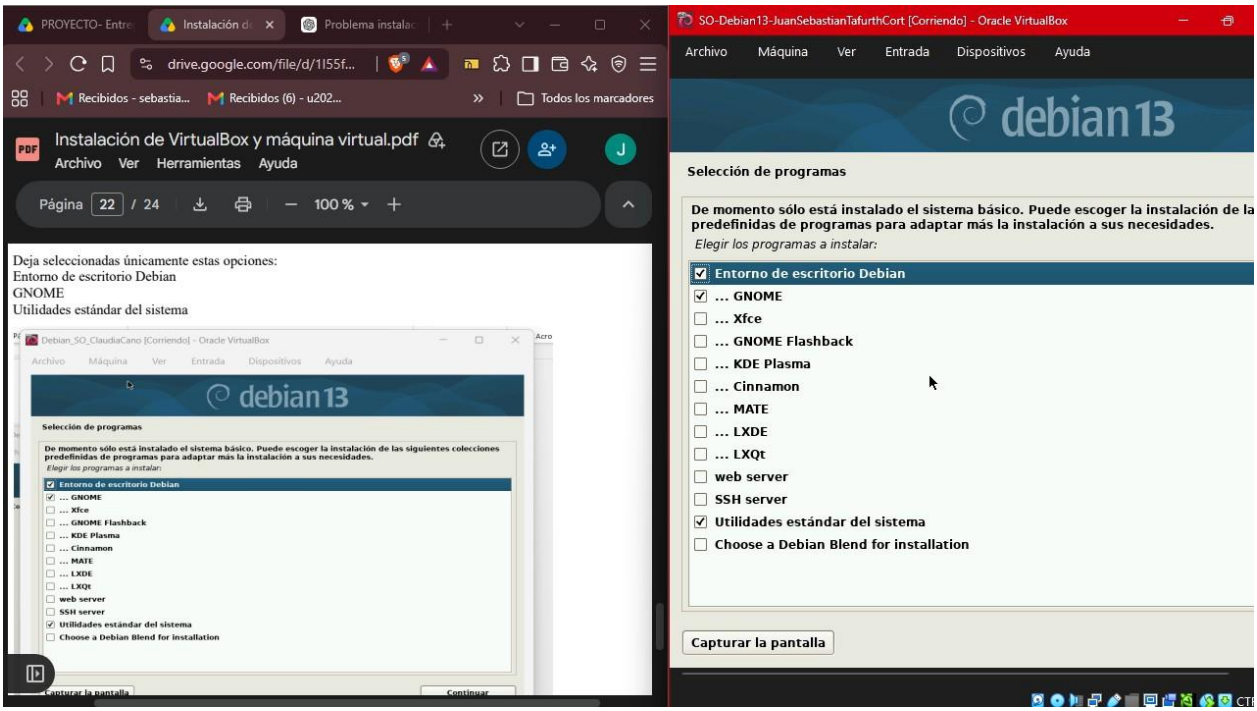
Captura de pantalla – Ventana Instalación Debian, Oracle Virtual box



Nota. A continuación, el sistema procedió con la selección e instalación de programas.

Figura 43

Captura de pantalla – Ventana Instalación Debian, Oracle Virtual box



Nota. Durante el proceso, se nos dio la opción de elegir programas básicos predefinidos dependiendo de las necesidades para la instalación. En este caso se seleccionó seleccionado:

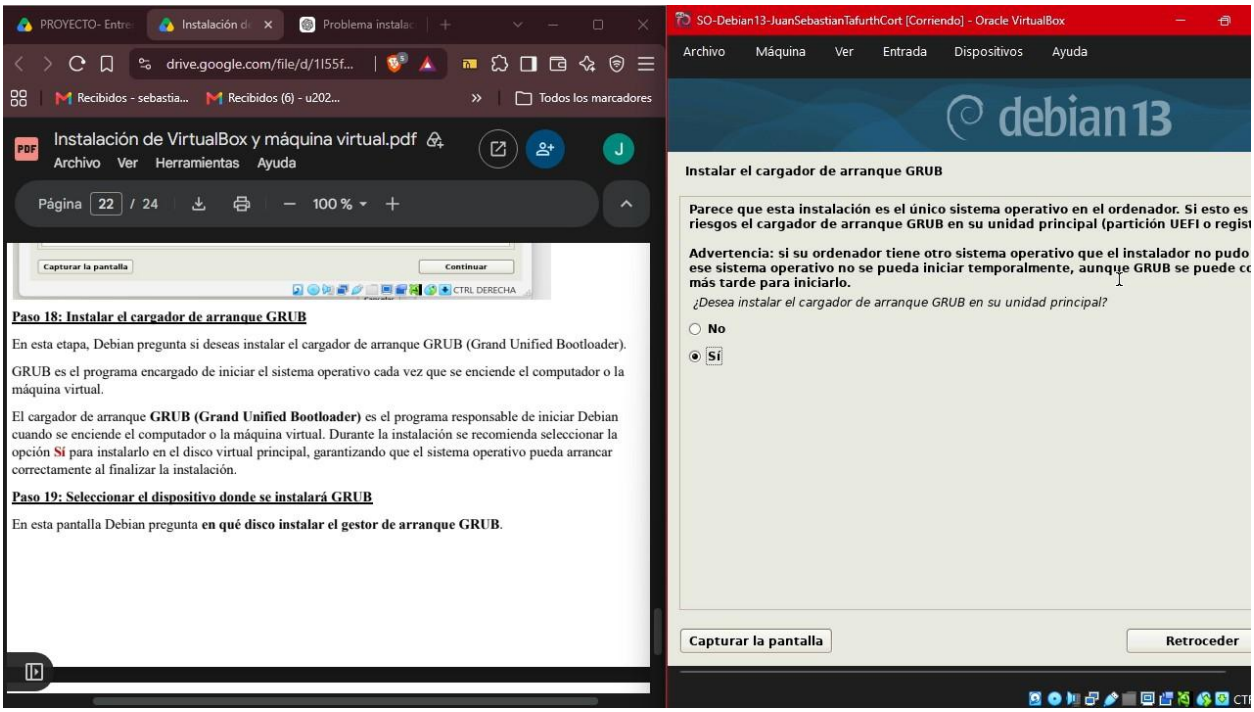
- Entorno de Escritorio Debian
- GNOME (Interfaz gráfica del sistema operativo)
- Utilidades estándar del sistema

Y se continuo con la instalación de los programas.

- **Instalación del cargador de arranque GRUB:**

Figura 44

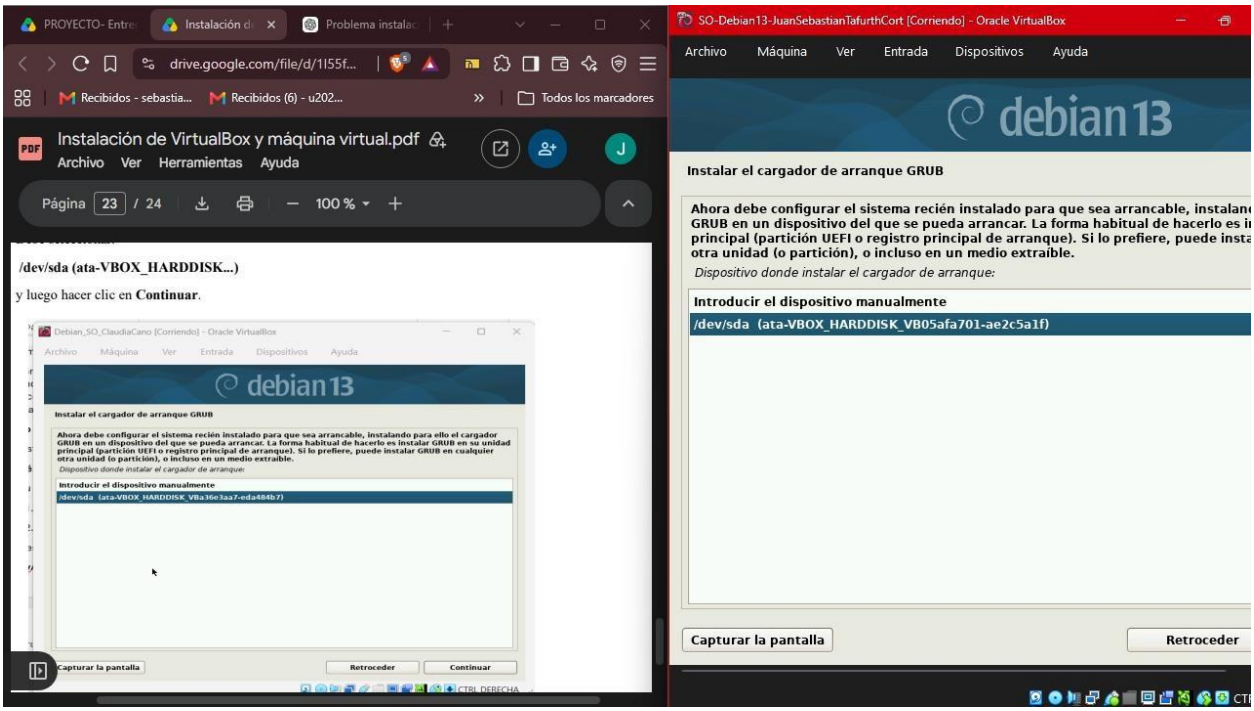
Captura de pantalla – Ventana Instalación Debian, Oracle Virtual box



Nota. Como último apartado para la instalación de Debian, se realizó la configuración de la instalación del cargador de arranque GRUB. Indicándole que el sistema operativo a instalar es el único en el ordenador, en este caso al ser una máquina virtual.

Figura 45

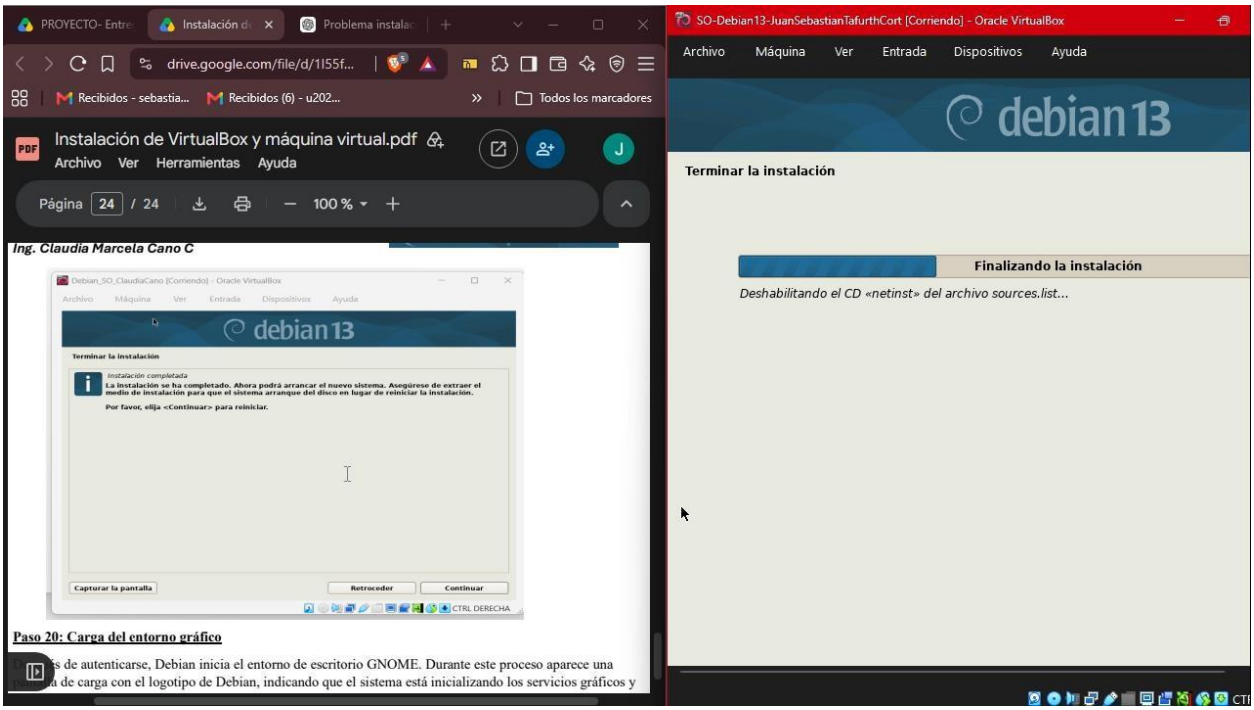
Captura de pantalla – Ventana Instalación Debian, Oracle Virtual box



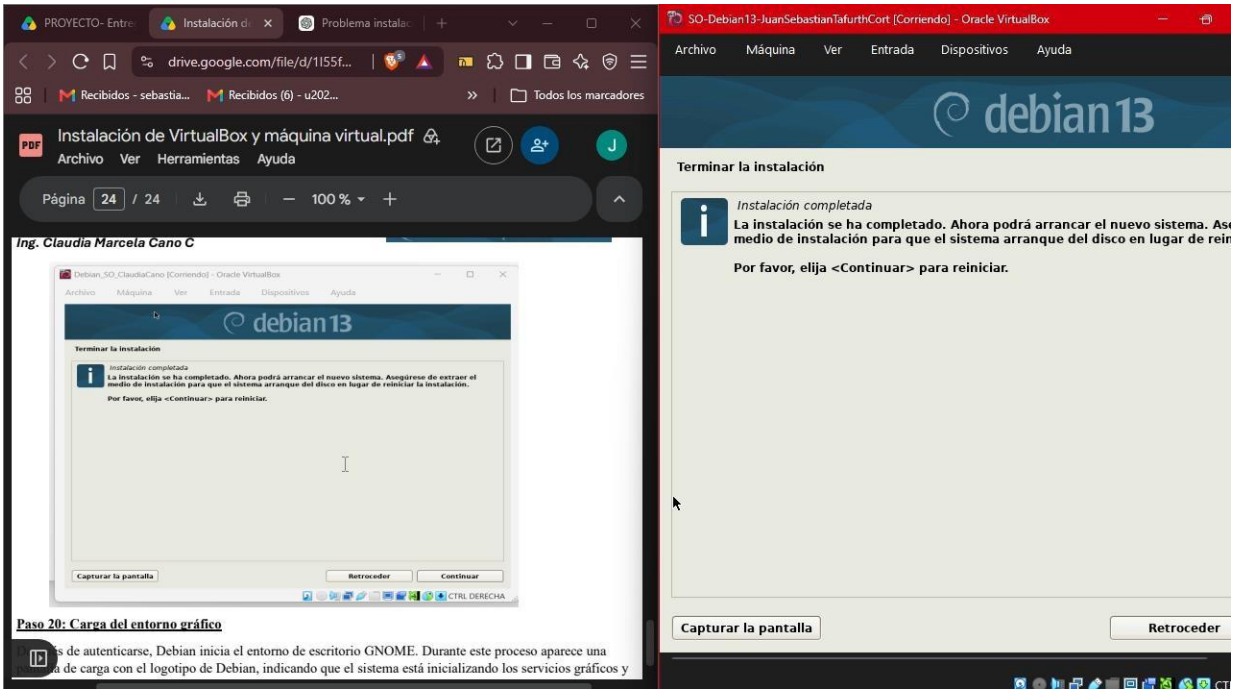
Nota. Por último, seleccionando el sistema ya instalado anteriormente con todas las configuraciones realizadas por nosotros los operadores anteriormente. Este sistema es el "/dev/sda (ata-VBOX_HARDDISK_VB05afa701-ae2c5alf)".

Figura 46

Captura de pantalla – Ventana Instalación Debian, Oracle Virtual box



Nota. Finalmente, con todas la configuraciones y apartados del sistema operativo terminando de instalarse, el sistema operativo se descargó completando todos los procesos necesarios.

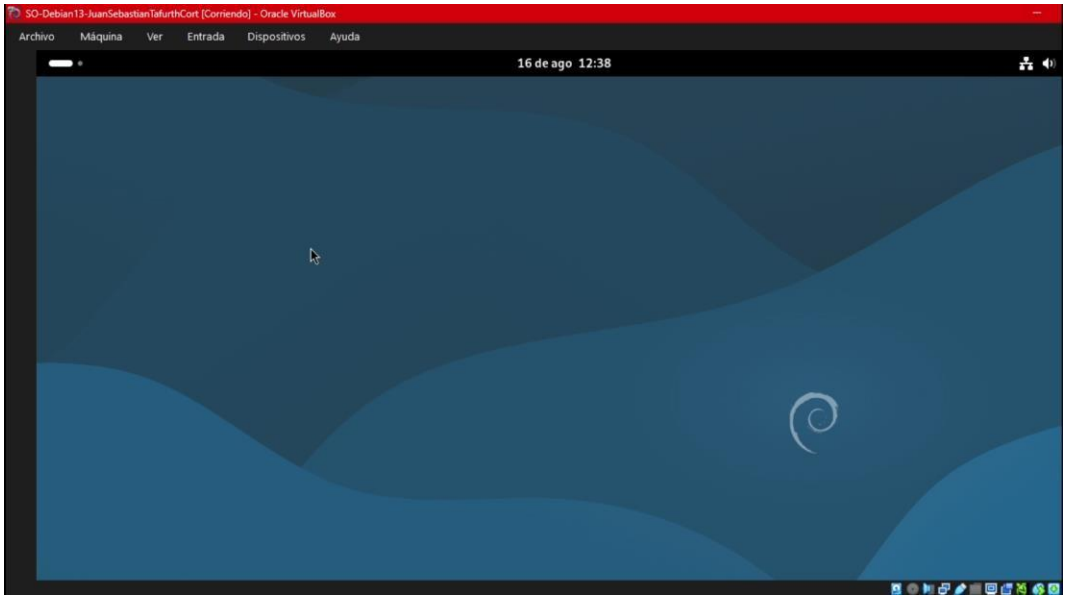


Verificación del Funcionamiento del Sistema:

Una vez finalizada la instalación y reiniciada la máquina virtual, se verificó el correcto arranque del entorno gráfico, el inicio de sesión y la conectividad a internet, actualizando los repositorios mediante comandos de terminal.

Figura 47

Captura de pantalla – Máquina Virtual, Entorno grafico debian



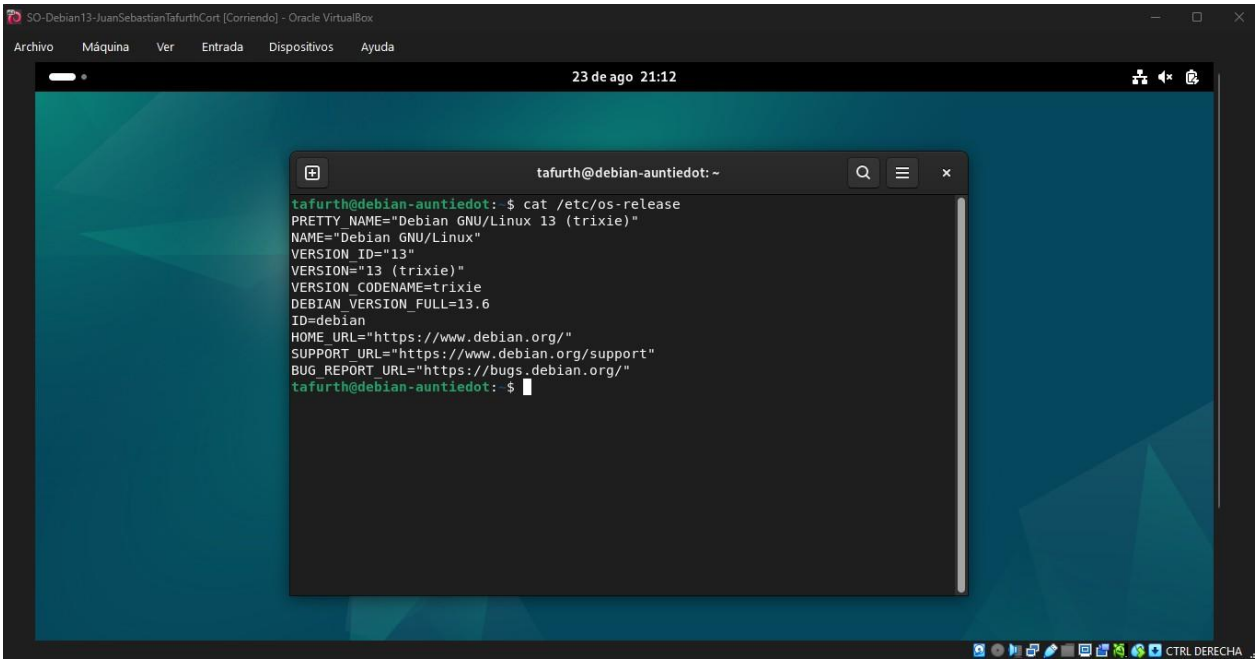
Parte II: Identificación de Características del Sistema

A continuación, se presentan las salidas de los comandos de verificación y diagnóstico ejecutados en la terminal para explorar el hardware y el Kernel del sistema.

Comando: cat /etc/os-release

Propósito: Nombre y versión de la distribución
Figura 48

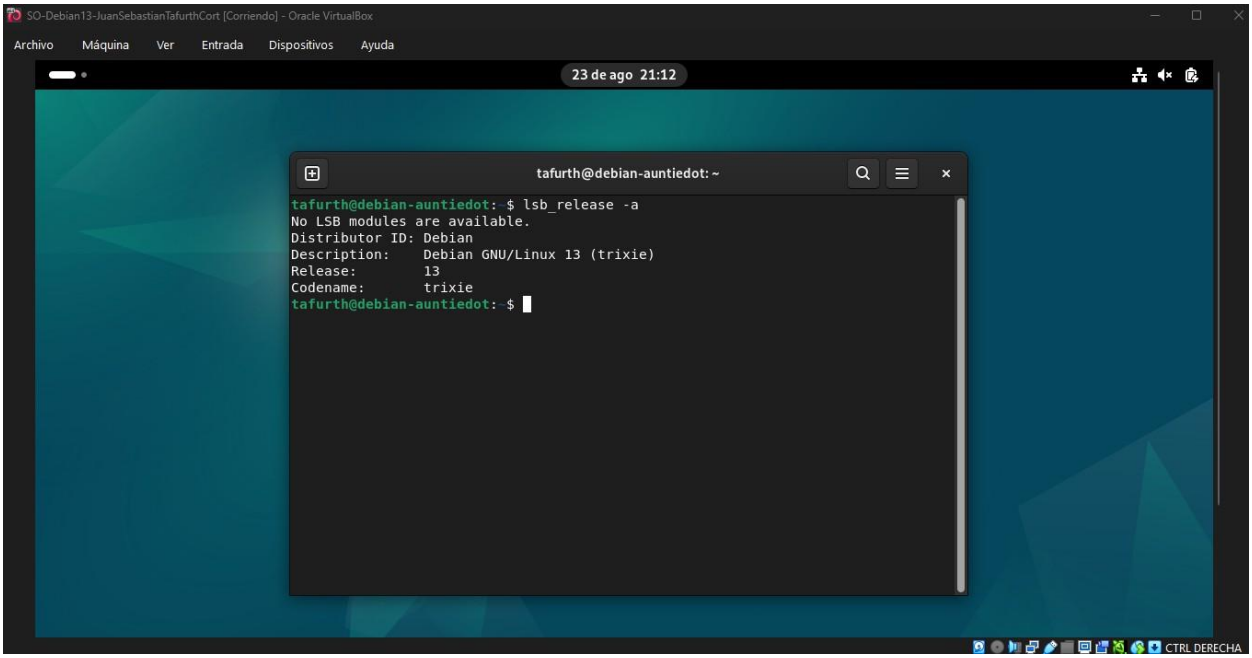
Captura de pantalla – Máquina Virtual, Entorno grafico debian



Comando: lsb_release -a

Propósito: Distribuidor, versión y codename
Figura 49

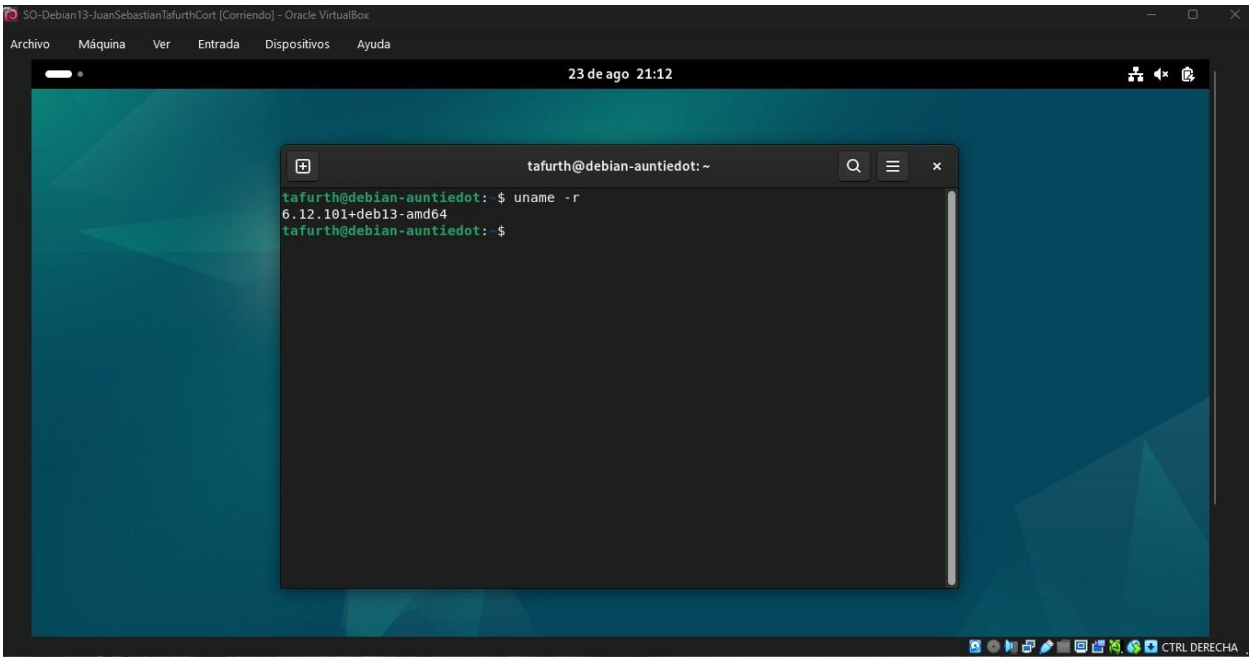
Captura de pantalla – Máquina Virtual, Entorno grafico debian



Comando: uname -r

Propósito: Versión del Kernel en ejecución
Figura 50

Captura de pantalla – Máquina Virtual, Entorno grafico debian

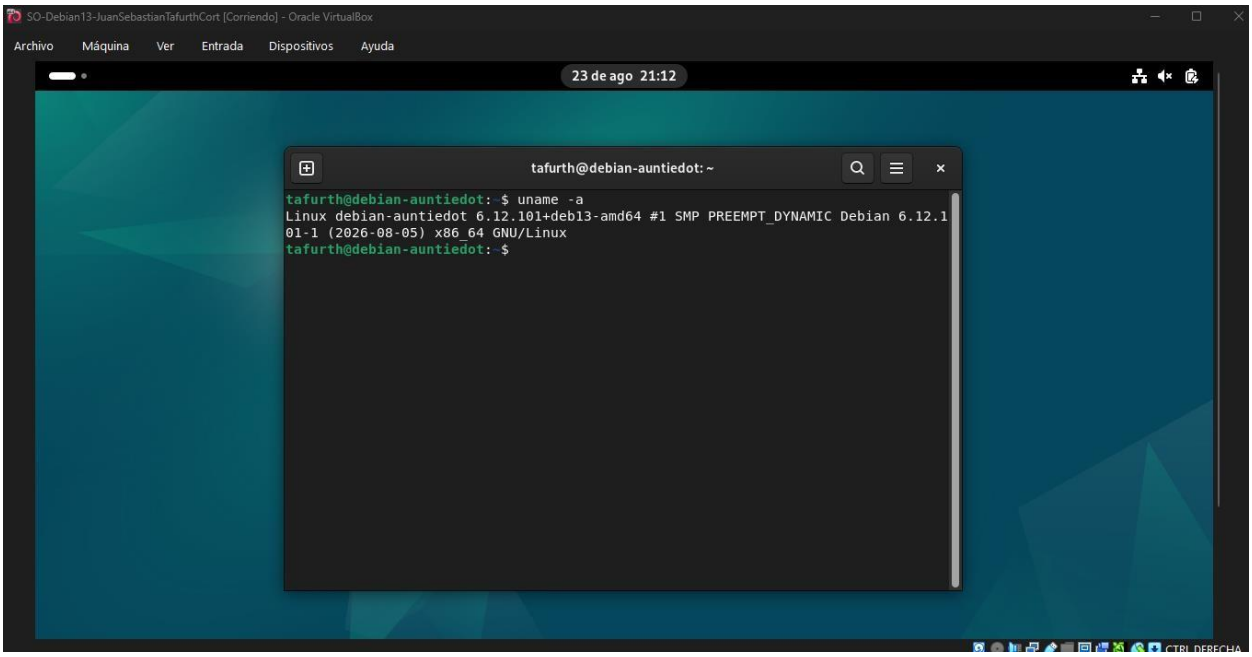


Comando: uname -a

Propósito: Kernel, hostname, arquitectura y fecha

Figura 51

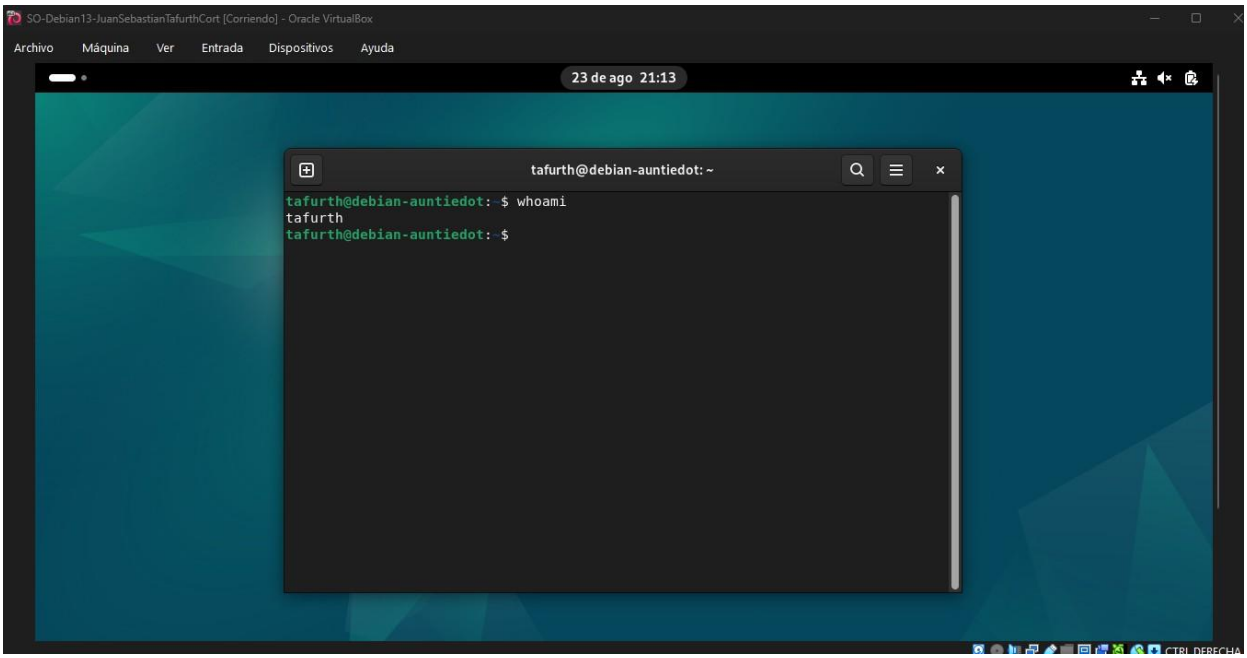
Captura de pantalla – Máquina Virtual, Entorno grafico debian



Comando: whoami

Propósito: Usuario con el que se inició sesión
Figura 52

Captura de pantalla – Máquina Virtual, Entorno grafico debian

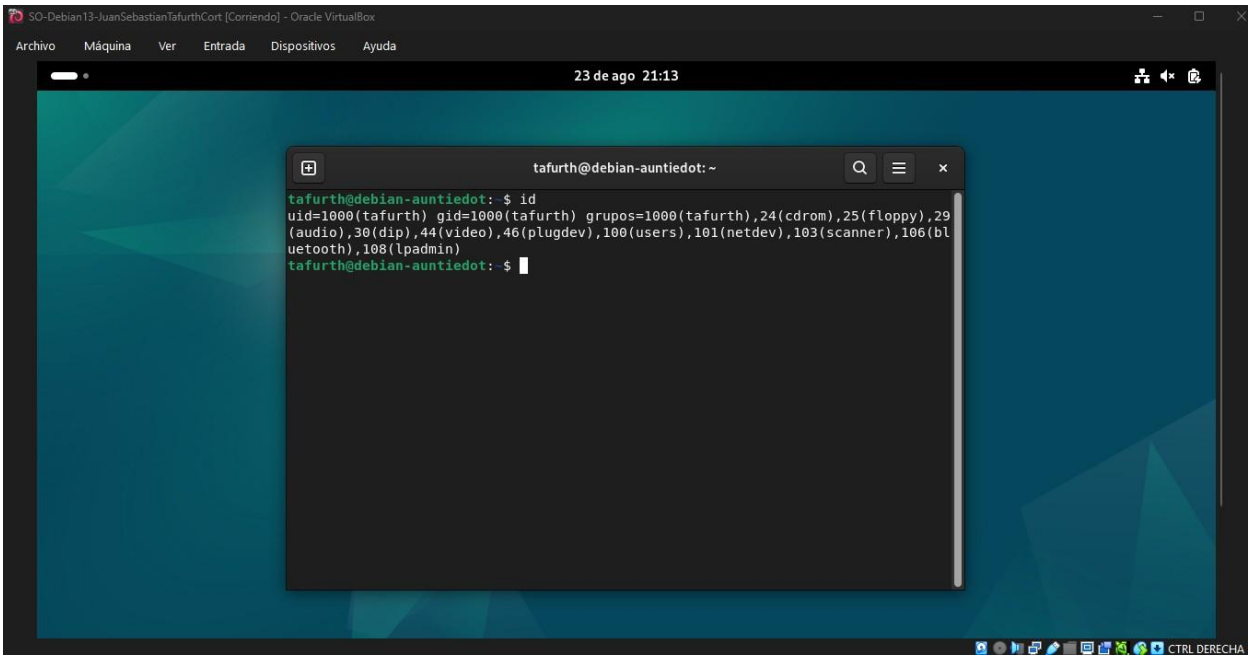


Comando: id

Propósito: UID, GID y grupos del usuario

Figura 53

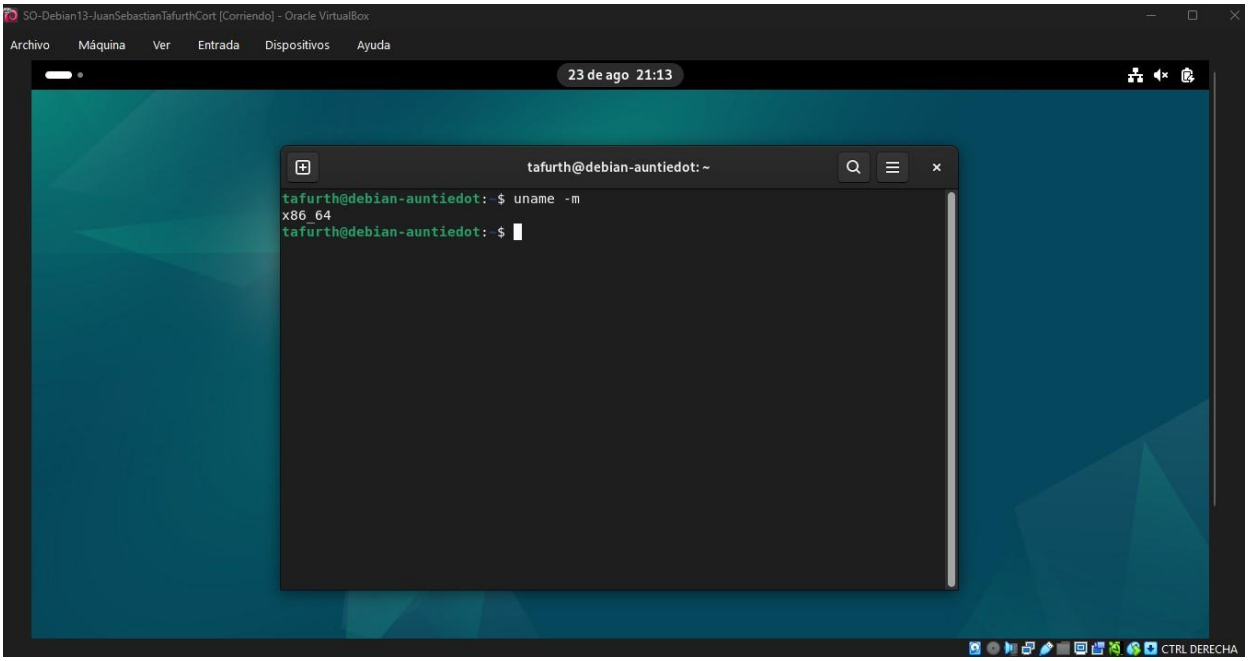
Captura de pantalla – Máquina Virtual, Entorno grafico debian



Comando: uname -m

Propósito: Arquitectura del hardware
Figura 54

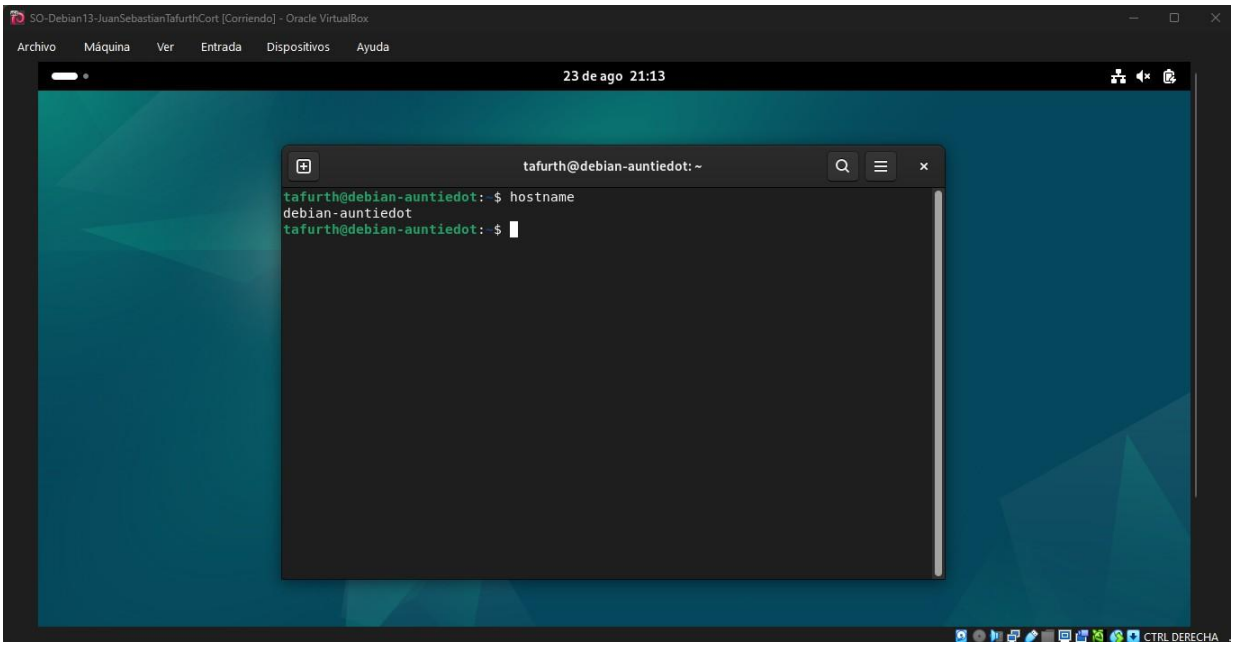
Captura de pantalla – Máquina Virtual, Entorno grafico debian



Comando: hostname

Propósito: Nombre configurado del equipo
Figura 55

Captura de pantalla – Máquina Virtual, Entorno grafico debian

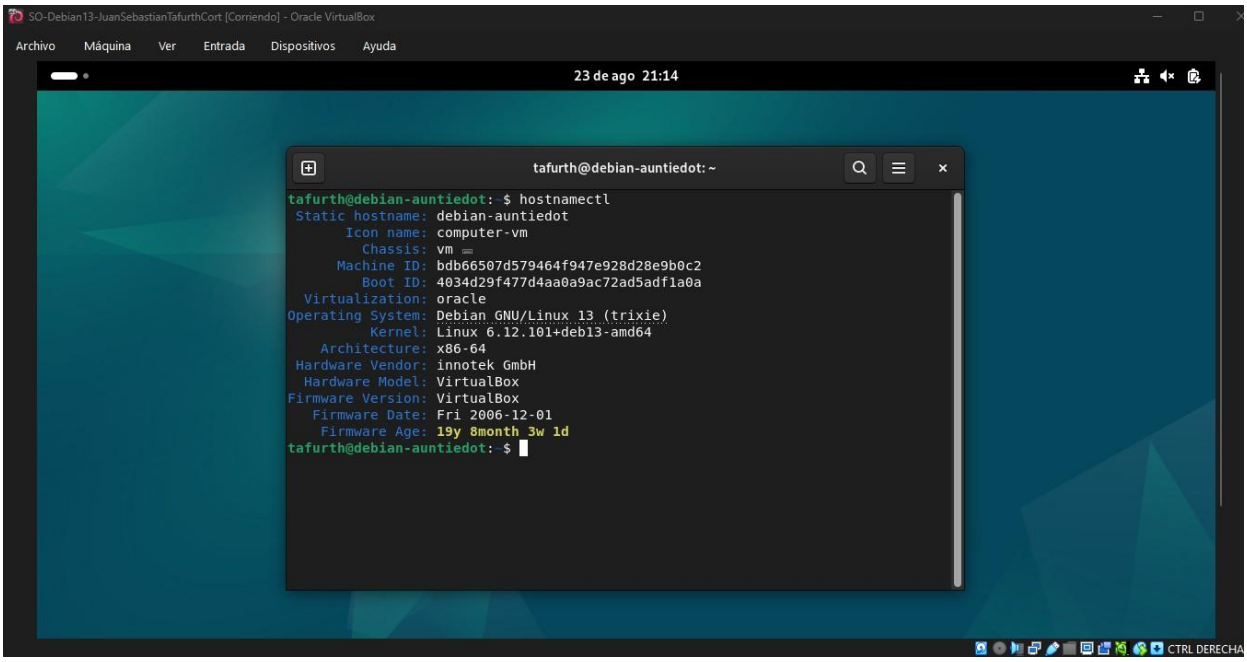


Comando: hostnamectl

Propósito: Reporte combinado: SO, kernel, arquitectura

Figura 56

Captura de pantalla – Máquina Virtual, Entorno grafico debian

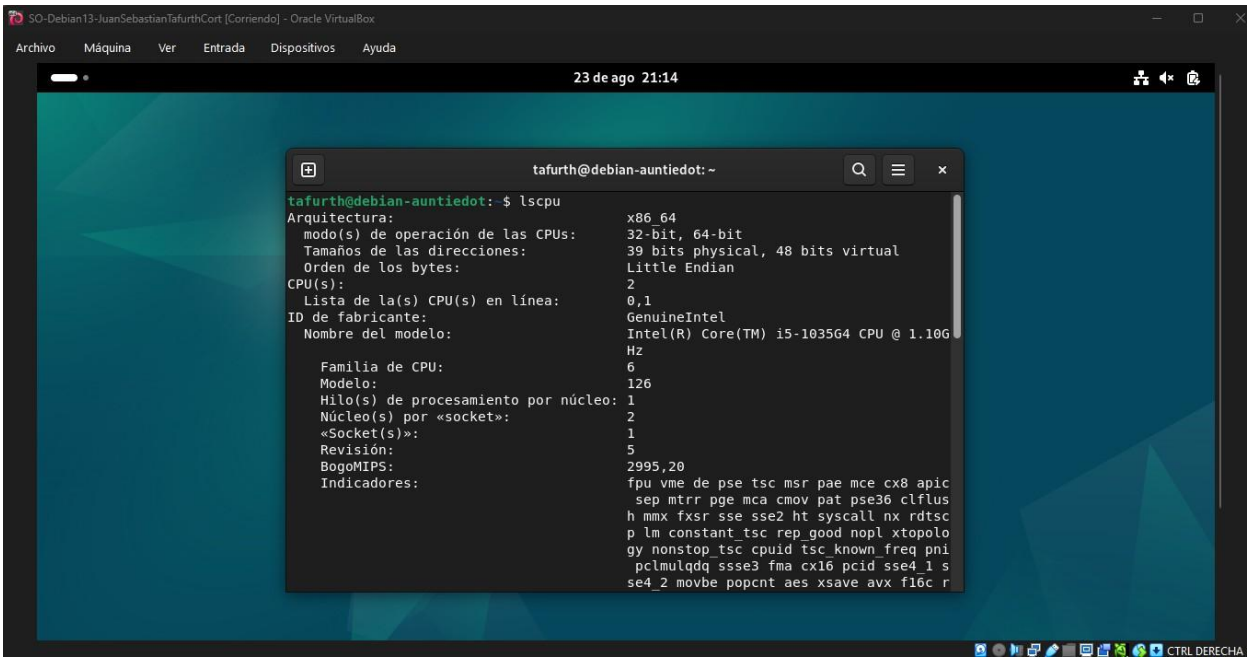


Comando: lscpu

Propósito: Modelo, núcleos y arquitectura del procesador

Figura 57

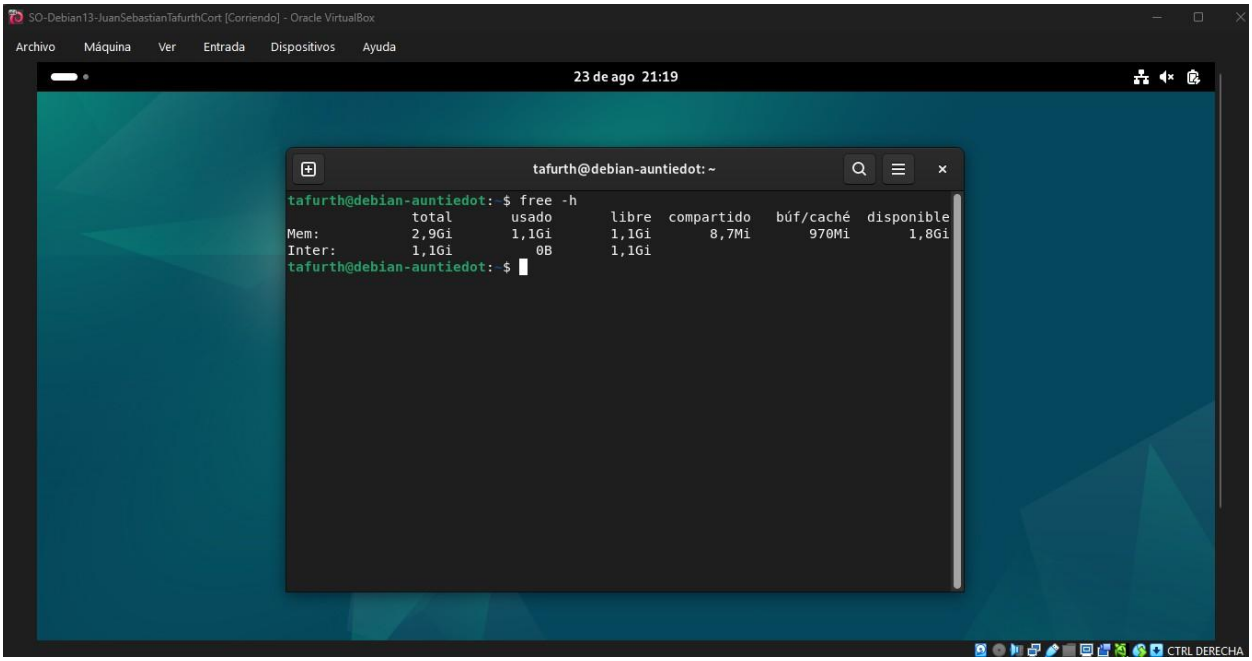
Captura de pantalla – Máquina Virtual, Entorno grafico debian


SO-Debian13-JuanSebastianTafurthCort [Corriendo] - Oracle VirtualBox
Archivo
Máquina
Ver
Entrada
Dispositivos
Ayuda
23 de ago 21:14
tafurth@debian-auntiedot: ~
tafurth@debian-auntiedot: $ lscpu
Arquitectura: x86_64
modo(s) de operación de las CPUs: 32-bit, 64-bit
Tamaños de las direcciones: 39 bits physical, 48 bits virtual
Orden de los bytes: Little Endian
CPU(s): 2
Lista de la(s) CPU(s) en línea: 0,1
ID de fabricante: GenuineIntel
Nombre del modelo: Intel(R) Core(TM) i5-1035G4 CPU @ 1.10GHz
Familia de CPU: 6
Modelo: 126
Hilo(s) de procesamiento por núcleo: 1
Núcleo(s) por «socket»: 2
«Socket(s)»: 1
Revisión: 5
BogoMIPS: 2995,20
Indicadores: fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush mmx fxsr sse sse2 ht syscall nx rdtscp lm constant_tsc rep_good nopl xtopology nonstop_tsc cpuid tsc_known_freq pni pclmulqdq ssse3 fma cx16 pcid sse4_1 sse4_2 movbe popcnt aes xsave avx f16c r
CTRL DERECHA

Comando: free -h

Propósito: Memoria RAM y swap
Figura 58

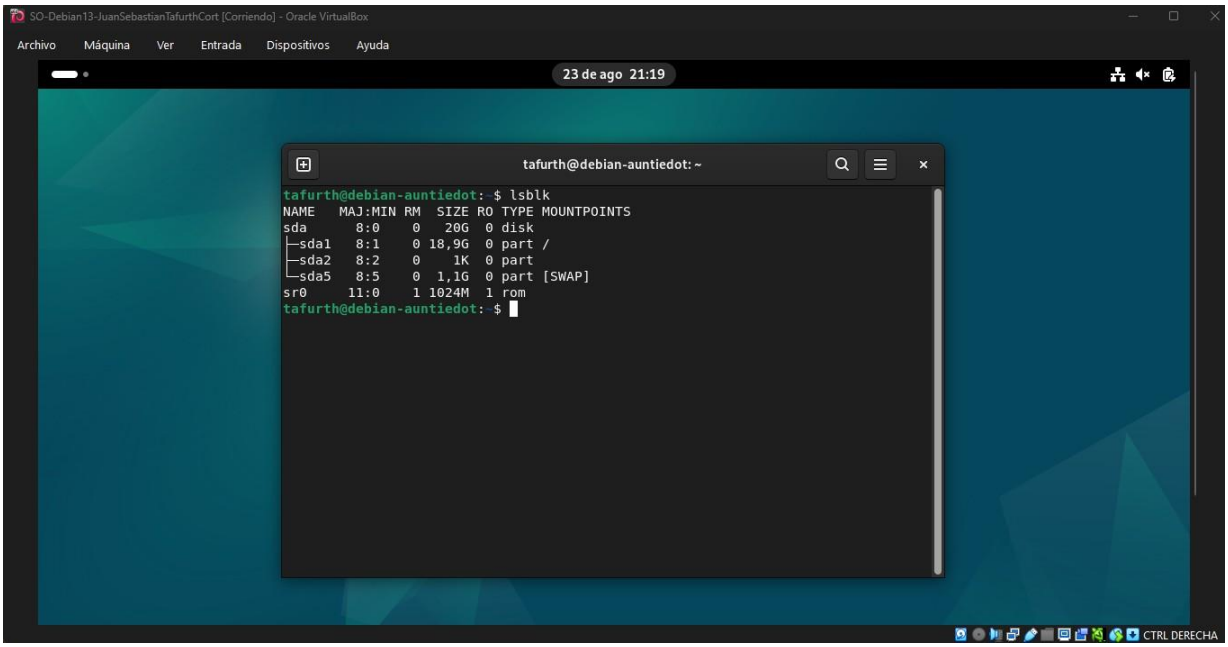
Captura de pantalla – Máquina Virtual, Entorno grafico debian



Comando: lsblk

Propósito: Discos y particiones del sistema
Figura 59

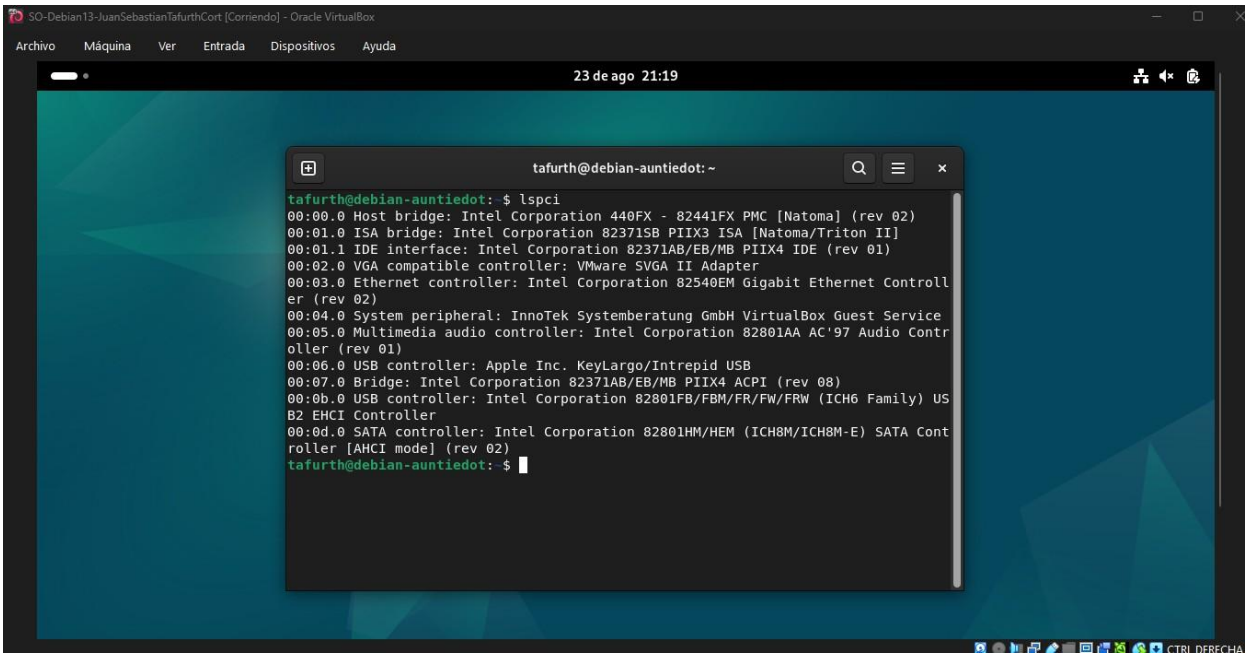
Captura de pantalla – Máquina Virtual, Entorno grafico debian



Comando: lspci

Propósito: Buses y dispositivos internos
Figura 60

Captura de pantalla – Máquina Virtual, Entorno grafico debian



Comando: lsusb

Propósito: Dispositivos conectados por USB
Figura 61

Captura de pantalla – Máquina Virtual, Entorno grafico debian

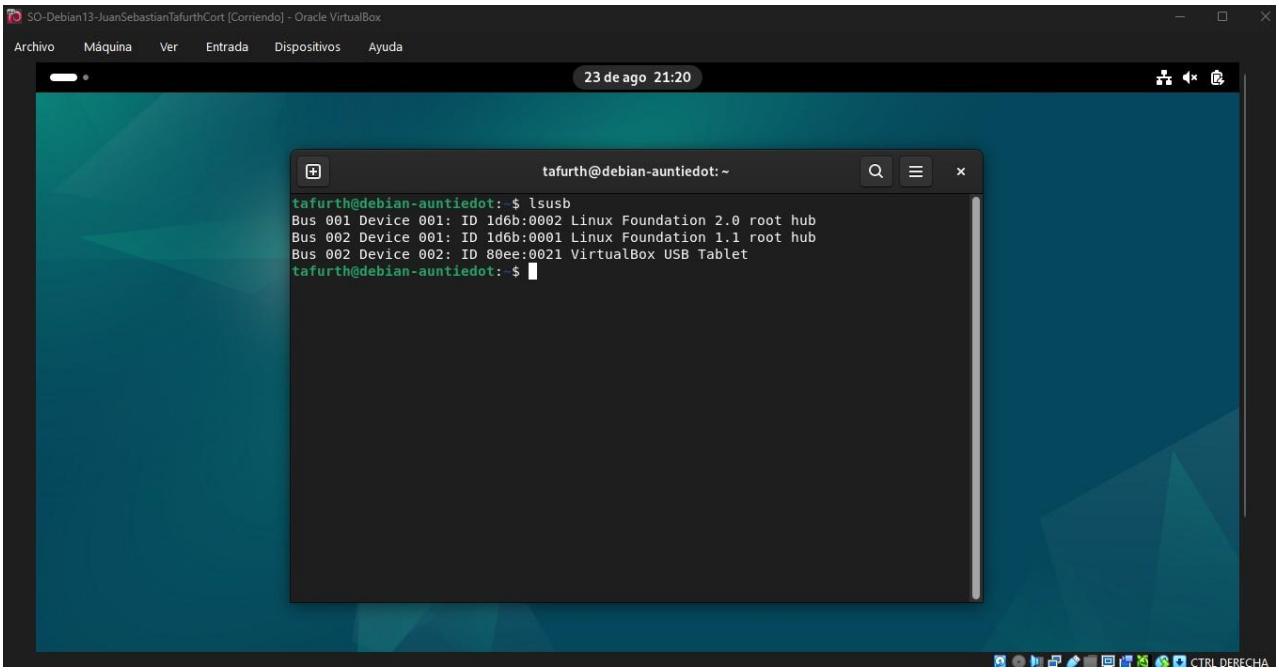

SO-Debian13-JuanSebastianTafurthCort [Corriendo] - Oracle VirtualBox
Archivo
Máquina
Ver
Entrada
Dispositivos
Ayuda
23 de ago 21:20
tafurth@debian-auntiedot: ~
tafurth@debian-auntiedot:~$ lsusb
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
Bus 002 Device 001: ID 1d6b:0001 Linux Foundation 1.1 root hub
Bus 002 Device 002: ID 80ee:0021 VirtualBox USB Tablet
tafurth@debian-auntiedot:~$
CTRL DERECHA

Tabla de Interpretación de Resultados

En la siguiente tabla se explica, con palabras propias, el significado de cada salida obtenida en los comandos de diagnóstico ejecutados anteriormente.

Comando	Salida Obtenida	Interpretación Propia
Cat /etc/os-release	PRETTY NAME="Debian GNU/Linux 13 (trixie)" NAME="Debian GNU/Linux" VERSION ID="13" VERSION="13 (trixie)" VERSION CODENAME=trixie DEBIAN VERSION FULL=13.6 ID=debian HOME URL="https: //www.debian.org/" SUPPORT URL="https://www.debian.org/support" BUG REPORT_URL="https://bugs.debian.org/"	Este comando el nombre y versión de la distribución instalada: • El nombre del SO es Debian GNU/Linux. • La versión del ID es "13" • La versión del OS "13 (trixie)" • La versión completa del sistema operativo es "13.6"
Lsb_release -a	No LSB modules are available. Distributor ID: Debian Description: Debian GNU/Linux 13 (trixie) Release:13 Codename: trixie	El comando muestra el distribuidor, la versión y el codename del SO: • El ID del distribuidor es "Debian" • La descripción del SO es "Debian GNU/Linux 12 (trixie) • Siendo la reléase (lanzamiento) número 13 del OS. • El nombre clave (Codename) de la versión es "trixie"

Uname -r	6.12.101+deb13-amd64	Sirve para mostrar la versión del Kernel en ejecución. • El "6" es la versión principal (major) del Kernel. Siendo la familia 6 del Kernel. • El "12" indica la versión dentro de la familia (minor). • El "101" indica la versión actual de parche.

		• "Deb13" indica que es un Kernel para Debian. • "amd64" indica la arquitectura sobre la que está construido el Kernel.
Uname -a	Linux debian-auntiedot 6.12.101+deb13-amd64 #1 SMP PREEMPT DYNAMIC Debian 6.12.1 01-1 (2026-08-05) x86_64 GNU/Linux	Proporciona información sobre el sistema y el Kernel, mostrando: • nombre del Kernel siendo "Linux". • nombre del equipo siendo "debian-auntiedot" • Versión del Kernel "6.12.101+deb13-amd64" • Información de compilación "PREEMPT DYNAMIC Debian 6.12.1" • Arquitectura "x86_64" • Sistema operativo "GNU/Linux"
Whoami	tafurth	• Este comando muestra el nombre del usuario activo actualmente
id	uid=1000(tafurth) gid=1000(tafurth) grupos=1000(tafurth), 24(cdrom), 25(floppy),29 (audio), 30(dip), 44(video) ,46(plugdev), 100(users), 101(netdev), 103 (scanner), 106(bluetooth), 108 (lpadmin)	Proporciona información acerca de la identificación del usuario actual, mostrando: • User Identifier / identificador del usuario, siendo 1000. • Group Identifier / Identificador del grupo, siendo 1000. • También muestra todos los grupos a los que pertenece el usuario y el identificador de cada grupo. En este caso el usuario pertenece a los grupos de audio, dip, video, plugdev, users, netdev, scanner, bluetooth y lpadmin.

Uname -m	x86_64	El comando muestra la arquitectura de hardware de la maquina en la que se ejecuta el Kernel.
Hostname	Debian-auntiedot	El comando muestra el nombre del equipo.
Hostnamectl	Static hostname: debian-auntiedot Icon name: computer-vm Chassis: vm m Machine ID: bdb66507d579464f947e928d28e9b0c2 Boot ID: 4034d29f477d4aa0a9ac72ad5adf1a0a Virtualization: oracle Operating System: Debian GNU/Linux 13 (trixie) Kernel: Linux 6.12.101+deb13-amd64 Architecture: x86-64 Hardware Vendor: innotek GmbH Hardware Model: VirtualBox Firmware Version: VirtualBox Firmware Date: Fri 2006-12-01 Firmware Age: 19y 8month 3w 1d	El comando muestra diversa información acerca del nombre del equipo, datos del sistema operativo y del Kernel, como: • Nombre del equipo "debian-auntiedot" • ID de la maquina: bdb66507d579464f947e928d28e9b0c2 • El identificar único del sistema (Boot ID): 4034d29f477d4aa0a9ac72ad5adf1a0a • Sistema operativo: Debian GNU/Linux 13 (trixie) • Kernel: 6.12.101+deb13-amd64 • Arquitectura: x86-64 • Fecha del firmware
Lscpu	Arquitectura: x86 64, modo(s) de operación de las CPUs: 32-bit, 64-bit. Tamaños de las direcciones: 39 bits physical, 48 bits virtual. Orden de los bytes: Little Endian. CPU (s): 2, Lista de la(s) CPU(s) en línea: 0,1. ID de fabricante: GenuineIntel. Nombre del modelo: Intel(R) Core (TM) 15-1035G4 CPU @ 1.10GHz. Familia de CPU:6. Modelo:126. Hilo(s) de procesamiento por nucleo:1, Núcleo(s) por «socket»: «Socket (s)»: Revisión:5.	El comando muestra información específica sobre el procesador y la arquitectura del sistema, por ejemplo: • Arquitectura: x86 64 • Modos de operación de CPUs: 32-bit, 64-bit • Cantidad CPUs: 2 • ID fabricante: GenuineIntel • Modelo del CPU: Intel® Core™ i5-1035G4 CPU 1.10Ghz • Familia del CPU: 6 • Modelo: 126 • Hilos de procesamiento: 1 • Núcleos por socket: 2

		• Socket(s) = 1
Free -h	Mem.total = 2.91Gi Mem.usada = 1.16Gi Mem.libre = 1,1Gi Mem.compartida: 8,7i Mem.búf/cache: 970Mi Mem.disponible: 1,8Gi Inter.total: 1.16Gi Inter.usado: 0B Inter.libre: 1,16Gi	El comando muestra de manera detalla información sobre la memoria RAM y la memoria de intercambio (Swap) del sistema. Mostrando aspectos como: • Memoria total de ambos • Memoria usada de ambos • Memoria libre de ambos • Memoria compartida, cache, y disponible de la RAM.
Lsblk	NAME: Sda -sda1 -sda2 -sda5 sr@ MAJ: MIN 8:0 8:1 8:2 8:5 11:6 RM: 0 0 0 0 1 SIZE: 20G 18,9G 1K 1,1G 1024M RO: 0 0 0 0 1 TYPE: disk part part part rom MOUNTPOINTS / [SWAP]	El comando muestra todos los dispositivos de almacenamiento y los quede de almacenito del sistema. • En la columna Name muestra los nombres de los dispositivos • En la columna RM se evidencia si el dispositivo es extraíble, el 1 representa el si y el 0 el no. • En la columna Size tamaño de cada partición • En la columna RO se evidencia si el dispositivo es de solo lectura, el 1 representa el si y el 0 el no. • Mountpoints dice el lugar donde se ubica la partición.
Lspci	00:00.0 Host bridge: Intel Corporation 440FX - 82441FX PMC [Natoma] (rev 02) 00:01.0 ISA bridge: Intel Corporation 82371SB PIIX3 ISA [Natoma/Triton II]	El comando sirve para mostrar todos los dispositivos de hardware conectados al bus PCI/PCIe del computador, por ejemplo:

	00:01.1 IDE interface: Intel Corporation 82371AB/EB/MB PIIX4 IDE (rev 01) 00:02.0 VGA compatible controller: VMware SVGA II Adapter 00:03.0 Ethernet controller: Intel Corporation 82540EM Gigabit Ethernet Controll er (rev 02) 00:04.0 System peripheral: InnoTek Systemberatung GmbH VirtualBox Guest Service 00:05.0 Multimedia audio controller: Intel Corporation 82801AA AC'97 Audio Controller (rev 01) 00:06.0 USB controller: Apple Inc. KeyLargo/Intrepid USB 00:07.0 Bridge: Intel Corporation 82371AB/EB/MB PIIX4 ACPI (rev 08) 00:0b.0 USB controller: Intel Corporation 82801FB/FBM/FR/FW/FRW (ICH6 Family) US B2 EHCI Controller 00:0d.0 SATA controller: Intel Corporation 82801HM/HEM (ICH8M/ICH8M-E) SATA Cont roller [AHCI mode] (rev 02)	• El host bridge es el puente principal que conecta la CPU con otros componentes. • ISA bridge es el puente para controladores antiguos • IDE interface es el controlador de almacenamiento IDE • VGA compatible controller es la tarjeta gráfica virtual • Ethernet controller es la tarjeta de red virtual • El System peripheral es el dispositivo de integración • El multimedia audio controller es la tarje de sonido virtual • El USB controller como su nombre lo dice es el controlador USB virtual • El Bridge es el puente relacionado con la administración de energía. • El SATA controller como lo dice su nombre es el controlador SATA virtual para el almacenamiento.
Lsusb	Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub Bus 002 Device 001: ID 1d6b:0001 Linux Foundation 1.1 root hub Bus 002 Device 002: ID 80ee:0021 VirtualBox USB Tablet	En última instancia este controlador muestra todos los dispositivos USB que Linux reconoce y están conectados al sistema. En este caso hay 3 conectados. Dentro de la información que proporciona esta: • El número de Bus • El numero del dispositivo conectado al

		Bus

		• El identificador del dispositivo y del fabricante

Conclusiones y Análisis Crítico

Reflexiones del Equipo

- En mi experiencia, Juan Sebastián Tafurth Cortes, fue interesante como entretenida debido a que es la primera vez manejando una distribución de Linux, habiendo escuchado tantas cosas llamativas sobre el uso de estos sistemas operativos. Aprendí diversas cosas nuevas acerca de cómo función internamente un sistema operativo y como trabaja en conjunto con el Kernel, que es una máquina virtual y cómo funciona para la ejecución de diversos sistemas operativos. Además de la importancia de los comandos y la terminal en los OS, aspecto que por lo menos en el uso común de Windows no se llega a apreciar. Durante este proceso solo surgió 1 dificultad con la instalación del entorno gráfico, sin embargo, no fue algo complicado de soluciona y se sobrellevo bien.

- En mi experiencia, Juan Esteban Bustos Chilatra, la verdad el proceso estuvo bastante interesante y me sirvió para consolidar lo que ya sabía. Ya que había trabajado antes con distros como Kali, Arch, Ubuntu y Debian, y me había puesto a configurar entornos gráficos como Hyprland y XFCE, pude enfocarme en otras cosas del sistema. Esta vez me di cuenta que usar estos comandos nuevos es clave para entender cómo manejar y hacer automatizaciones directo en servidores Linux. Por otro lado, aunque usar una máquina virtual es mucho más cómodo ya que uno no tiene que ponerse a configurar el GRUB ni particionar el disco a mano, considero que siempre va a ser mucho mejor trabajar en un dual boot para sacarle todo el rendimiento bruto al equipo sin ningún tipo de limitación.

mano, considero que siempre va a ser mucho mejor trabajar en un dual boot para sacarle todo el rendimiento bruto al equipo sin ningún tipo de limitación.

Preguntas de Reflexión

1. ¿Qué diferencia existe entre un hipervisor de Tipo 1 y uno de Tipo 2? ¿En qué categoría se ubica VirtualBox y por qué?

En primera instancia, hay que tener en cuenta qué es un hipervisor. Un hipervisor es el software que permite crear y ejecutar máquinas virtuales dentro de computadores físicos. Agrupa los recursos del computador, como el procesamiento, la memoria y el almacenamiento, y los destina para el uso de la máquina virtual.

Teniendo esto en cuenta los hipervisores de clasifican en Hipervisor tipo I y tipo II.

- ✓ El hipervisor tipo 1 conocido como bare metal se ejecuta directamente sobre el hardware, siendo como una especie de OS host, encargado de administrar los recursos físicos, como CPU, memoria, almacenamiento y repartirlo entre las demás máquinas virtuales. Este tipo de hipervisor es principalmente usado en ámbito de servidores
- ✓ El hipervisor tipo 2 conocido como hipervisor alojado se ejecuta sobre un sistema operativo ya instalado previamente, actuando como una capa de software/aplicación dentro del sistema. El OS anfitrión es el que administra los recursos del hardware de la maquina y le designa al hipervisor tipo II y el mismo le designa una parte de los recursos. Después cada sistema operativo de la máquina virtual actuaria como un OS guest. Este tipo de hipervisor de utiliza principalmente para uso personal.

Analizando las características de cada tipo de hipervisor, se puede afirmar que Oracle VirtualBox se encuentra en la categoría de hipervisor 2, ya que este se instala sobre un sistema operativo ya vigente en la máquina. En el caso de la práctica, se instaló la máquina en el SO de Windows; por lo mismo, este no se instala sobre el hardware de la máquina. Por el contrario, desde el SO host se le asignan los recursos a la máquina virtual, práctica realizada durante la creación de la instancia de VM para Debian. (Red Hat, 2026)

2. ¿Qué ventajas ofrece trabajar con una máquina virtual en lugar de instalar el sistema operativo directamente sobre el hardware físico?

La principal ventaja es que se pueden instalar sin afectar directamente el SO host que se maneja. Todo lo que se ejecute dentro de la máquina virtual que queda ahí no afecta de ninguna manera los archivos o procesos del SO host. Además, se pueden ejecutar múltiples sistemas operativos en un solo computador. En última instancia, otra ventaja es el aprendizaje, al ofrecer las VM un entorno aislado. Cualquier error cometido en la instalación no resulta catastrófico para todo el sistema, por lo que es una excelente herramienta de aprendizaje para desarrolladores. (Scale Computing, 2025).

3. A partir de la información obtenida con uname -a y hostnamectl, ¿qué relación existe entre el Kernel y el sistema operativo completo (distribución)?

En primera instancia, la información obtenida con el comando “uname -a” es exclusivamente del Kernel, mientras que el comando “hostnamectl” muestra información variada sobre el sistema operativo. La relación entre estos dos componentes consiste en que el Kernel actúa como parte central del sistema operativo, siendo el que controla y distribuye los recursos

físicos para su correcto funcionamiento, mientras que el sistema operativo trabaja en conjunto con el Kernel para permitir que el usuario interactúe con el sistema mediante una interfaz gráfica, sin necesidad de que el usuario tenga conocimiento técnico de cómo funcionan los procesos del SO y el Kernel. (IBM, s. f.; Red Hat, s. f.)

4. ¿Por qué es útil para un administrador de sistemas conocer estos comandos de diagnóstico?

Es útil e importante que un administrador de sistemas conozca estos comandos de diagnóstico, debido a que permiten obtener información sobre el estado, la configuración y diversas características adicionales del sistema. Esto permite verificar que todo esté funcionando correctamente y, en caso de presentar problemas, saber identificarlos mediante los comandos de diagnóstico y poder solucionarlos. (IBM, s. f.)

5. ¿Qué dificultades se presentaron durante la instalación o configuración y qué solución encontraron?

Durante la instalación de la máquina virtual solo surgió una dificultad durante la instalación, que surgió después de la configuración del gestor de paquetes, debido a que, durante la selección de programas, solo aparecía una opción, la cual era “Entorno de escritorio Debian”; sin embargo, no aparecían las opciones del entorno gráfico GNOME y las utilidades estándar del sistema. Lo que se hizo para resolver esta problemática fue darle al botón de retroceder, llevándonos a un apartado donde nos permitían volver a realizar la configuración de cualquiera de las secciones realizadas anteriormente. Para asegurar que no volviera a tener el mismo error, se decidió reconfigurar la sección anterior, la cual es “Configuración del gestor de paquetes”, y la “Selección e instalación de programas”. Finalmente, una vez realizada nuevamente la

configuración, ya aparecieron las opciones de entorno gráfico y las utilidades estándar del sistema.

Parte III: Administración de Procesos y Laboratorio de Concurrencia

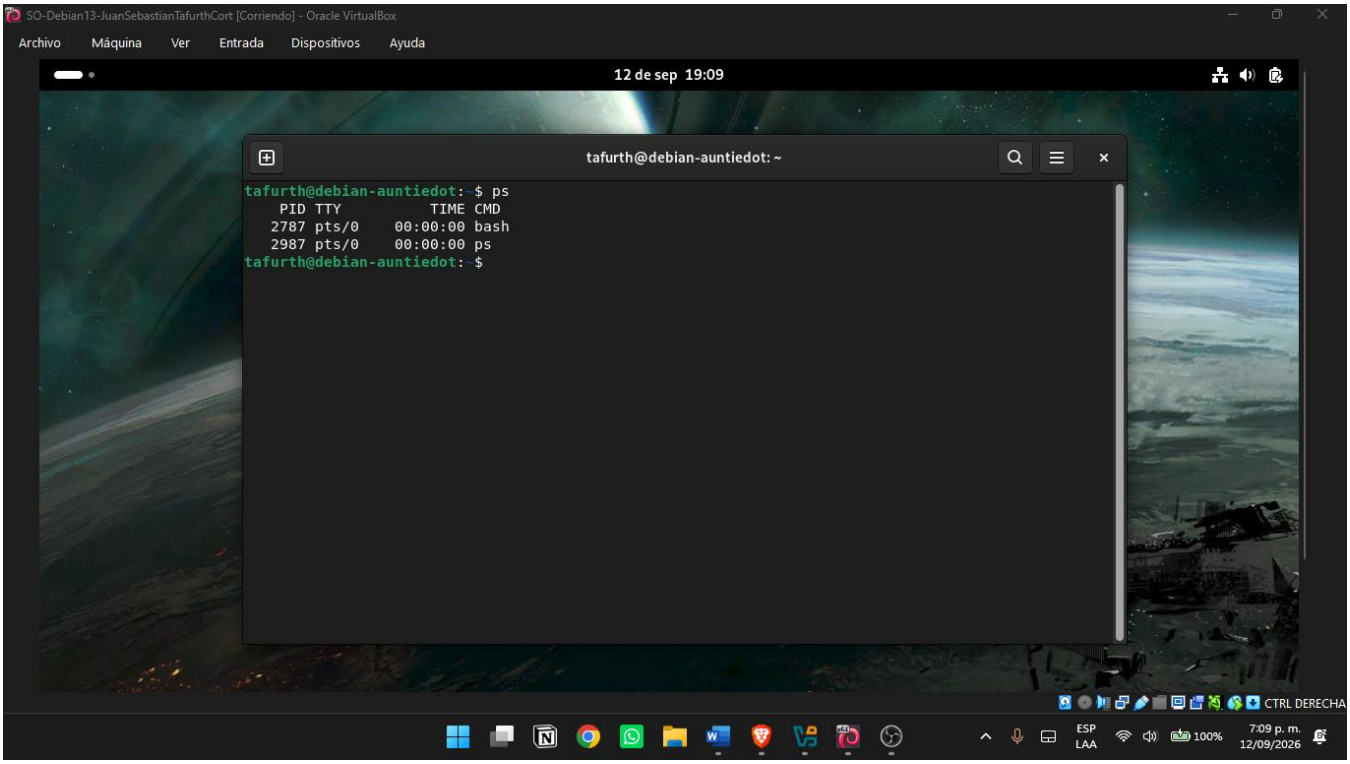
Es útil e importante que un administrador de sistemas conozca estos comandos de diagnóstico, debido a que permiten

PARTE A — Administración real de procesos (Unidad 3)

A.1 Identificación y consulta de procesos

Figura 62

Captura de pantalla – Máquina Virtual, Terminal



Nota. Se ejecutó el comando **«Ps»** en la terminal de Debian. Muestra de manera básica los procesos activos únicamente asociados al usuario que esté manejando el sistema, en este caso «tafurth», además de la terminal en uso. Como salida muestra una tabla con los datos: PID (*Process ID*), el cual es el identificador del proceso; TTY (*Teletypewriter*), la consola en uso; TIME, el tiempo total de uso de la CPU; y en última instancia CMD (*Command*), la orden o

comando que inició el proceso. CMD (Command) siendo el nombre del comando en ejecución.

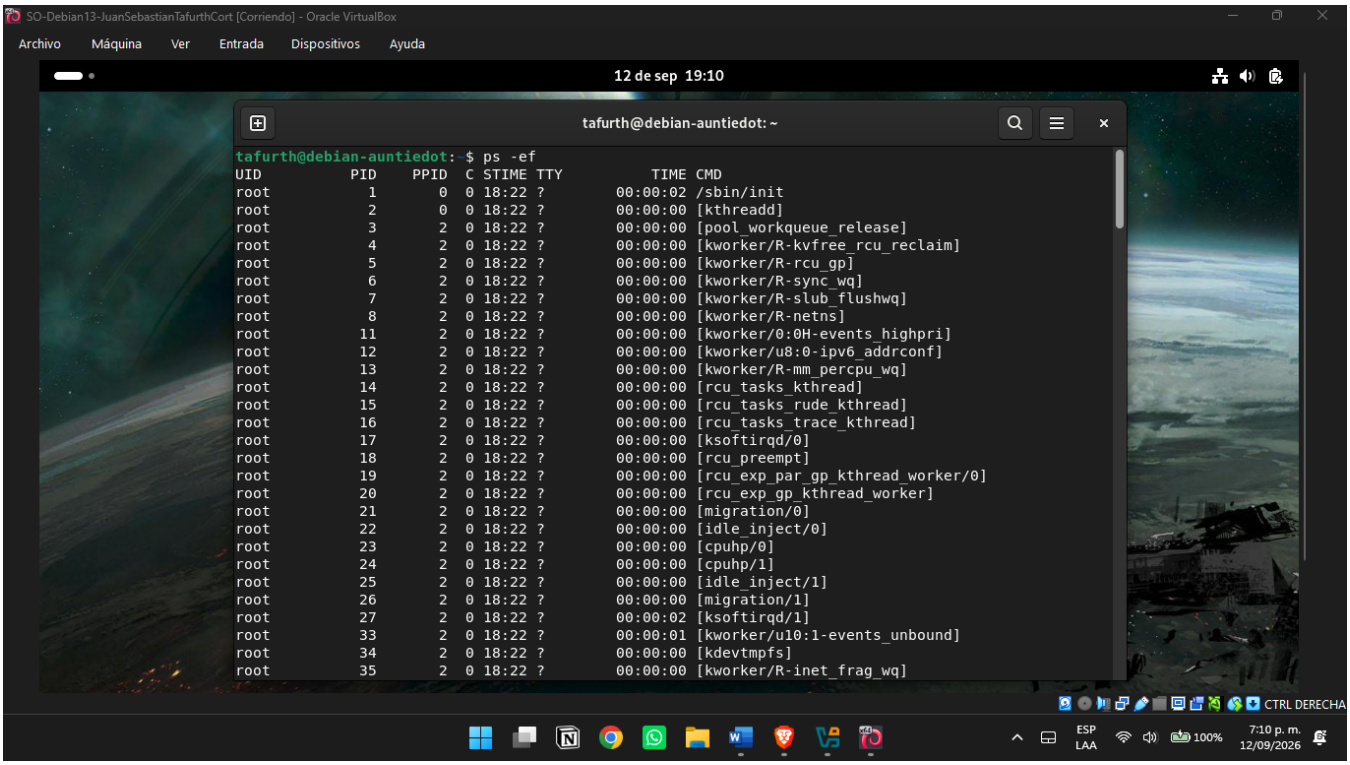
Ejemplo proceso:

PID	TTY	TIME	CMD
2787	Pts/0	00:00:00	bash

En este ejemplo se ejecutó el proceso con el PID 2787 en la terminal 0, usando 00:00:00 de tiempo de procesamiento, siendo este proceso el intérprete de comandos *bash*.

Figura 63

Captura de pantalla – Máquina Virtual, Terminal



Nota. Muestra todos los procesos ejecutados al instante por el sistema de manera ordenada en listas y con diversa información: el UID (User ID), el nombre del usuario que ejecutó el proceso; el PID (Process ID), el identificador único del proceso; el PPID (Parent Process ID), el identificador del proceso padre; C, el uso estimado de CPU; STIME (Start Time), la hora en la que inició el proceso; TTY (Teletypewriter), la terminal desde donde se ejecutó; TIME, el

tiempo total de ejecución del proceso; y CMD (Command), el nombre del comando ejecutado.

Como se puede apreciar en la captura, se ven diferentes procesos con valores en cada apartado mencionado anteriormente. En este caso, todos fueron ejecutados por el usuario root.

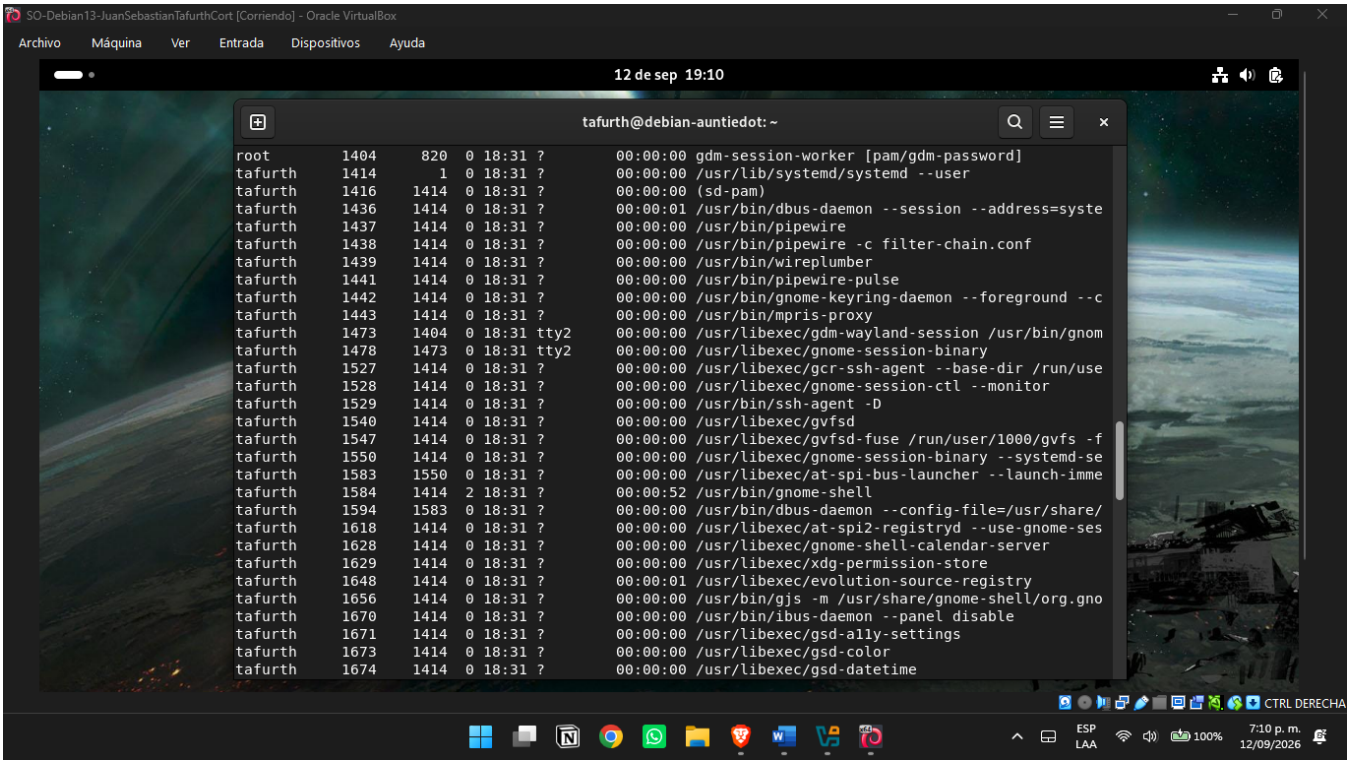
Ejemplo de proceso:

UID	PID	PPID	C	STIME	TTY	TYME	CMD
ROOT	1	0	0	18:22	?	00:00:02	/sbin/init

En este ejemplo, el usuario **«root»** ejecutó un proceso con PID 1, con un uso de CPU de 0, a las 18:22, tardando 00:00:02 en su procesamiento, siendo este comando **<</sbin/init>>.** Este comando es el primero que se ejecuta tras el núcleo del kernel, debido a que es el encargado de inicializar el sistema.

Figura 64

Captura de pantalla – Máquina Virtual, Terminal



Nota. Esta captura sigue perteneciendo a la ejecución del comando **«ps -ef»,** mostrando todos

los procesos ejecutados, esta vez por el usuario **«tafurth».**

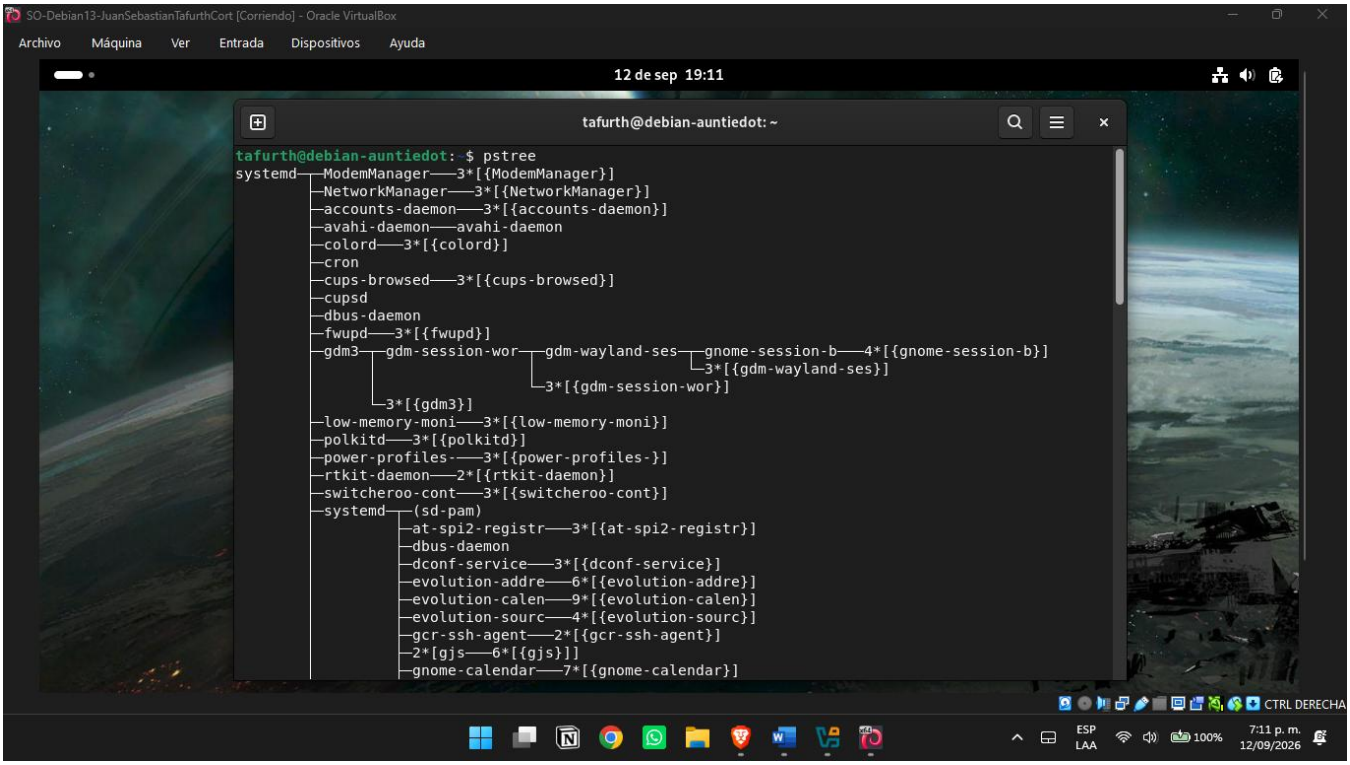
Ejemplo de proceso:

UID	PID	PPID	C	STIME	TTY	TYME	CMD
TAFURTH	1584	1414	1	18:31	?	00:00:57	/urs/bin/gnome-shell

En este ejemplo, el usuario «tafurth» ejecutó un proceso con PID 1548, un uso de CPU de 1, a las 18:31, tardando 00:00:57 en su procesamiento, siendo este comando **<</usr/bin/gnome-shell>>.** Este comando se encarga de ejecutar la interfaz gráfica del sistema operativo.

Figura 65

Captura de pantalla – Máquina Virtual, Terminal

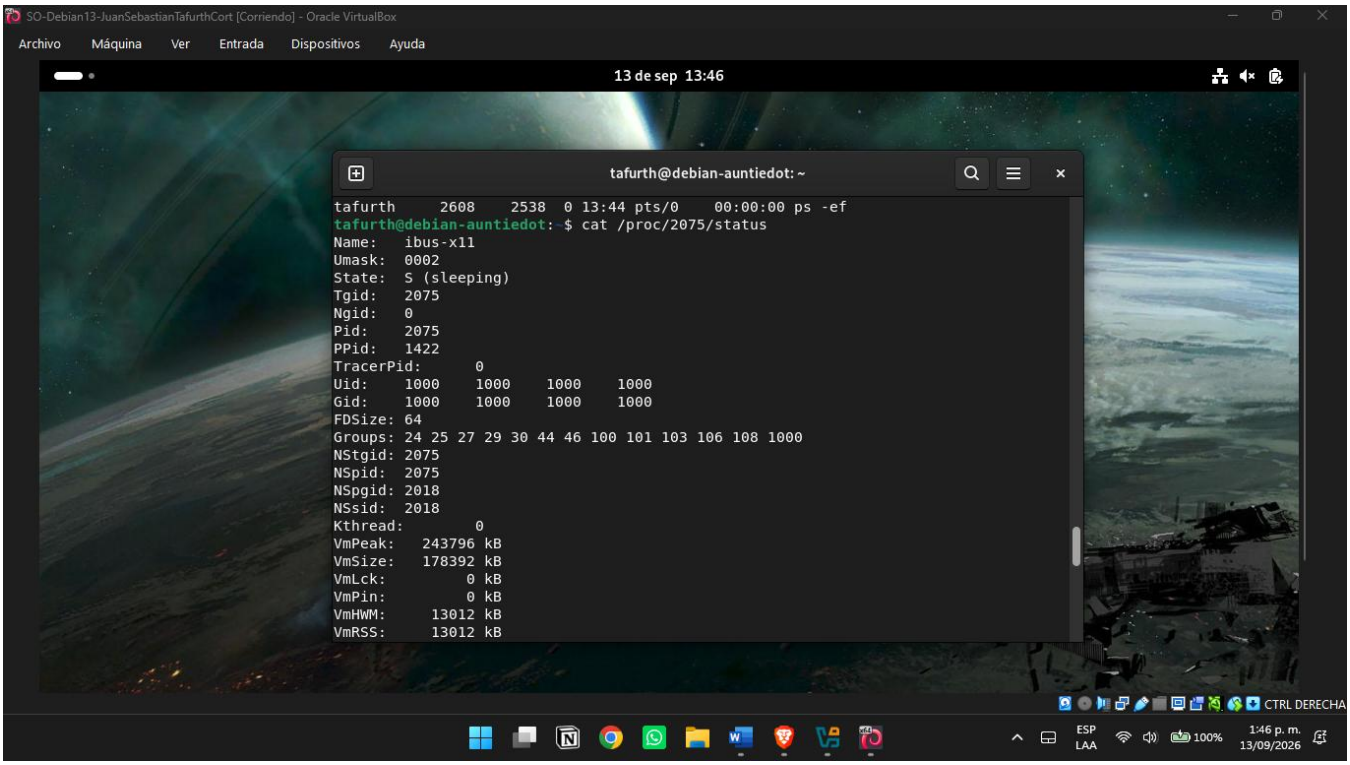


Nota. El comando **<<Pstree>>** muestra de manera estructurada los procesos en un árbol jerárquico, permitiendo ver de forma intuitiva la interrelación entre procesos padres e hijos. Analizando la captura, se puede evidenciar que el primer proceso en ejecutarse y del cual derivan todos los demás es **<<System>>,** el cual actúa como administrador central del sistema. De este se derivan más procesos, como **«ModemManager»**, encargado de integrar y gestionar

dispositivos de banda ancha móvil; **«gdm3»,** el administrador de la pantalla de inicio de sesión del entorno gráfico; o **«power-profiles-daemon»**, que se ocupa de administrar los perfiles de consumo de energía del equipo.

Figura 66

Captura de pantalla – Máquina Virtual, Terminal



Nota. Se ejecutó el comando **<<cat /proc/2075/status>>** con el fin de consultar el estado del proceso con PID 2075, siendo este el proceso **<</usr/libexec/ibus-x11>>**, encargado de gestionar la entrada de texto en el entorno gráfico. Algunas de las características que muestra son: *State*, si el proceso está activo o inactivo; *VmRSS*, la memoria RAM que está ocupando el proceso; y *Threads*, el número total de hilos que el proceso tiene activos para su ejecución.

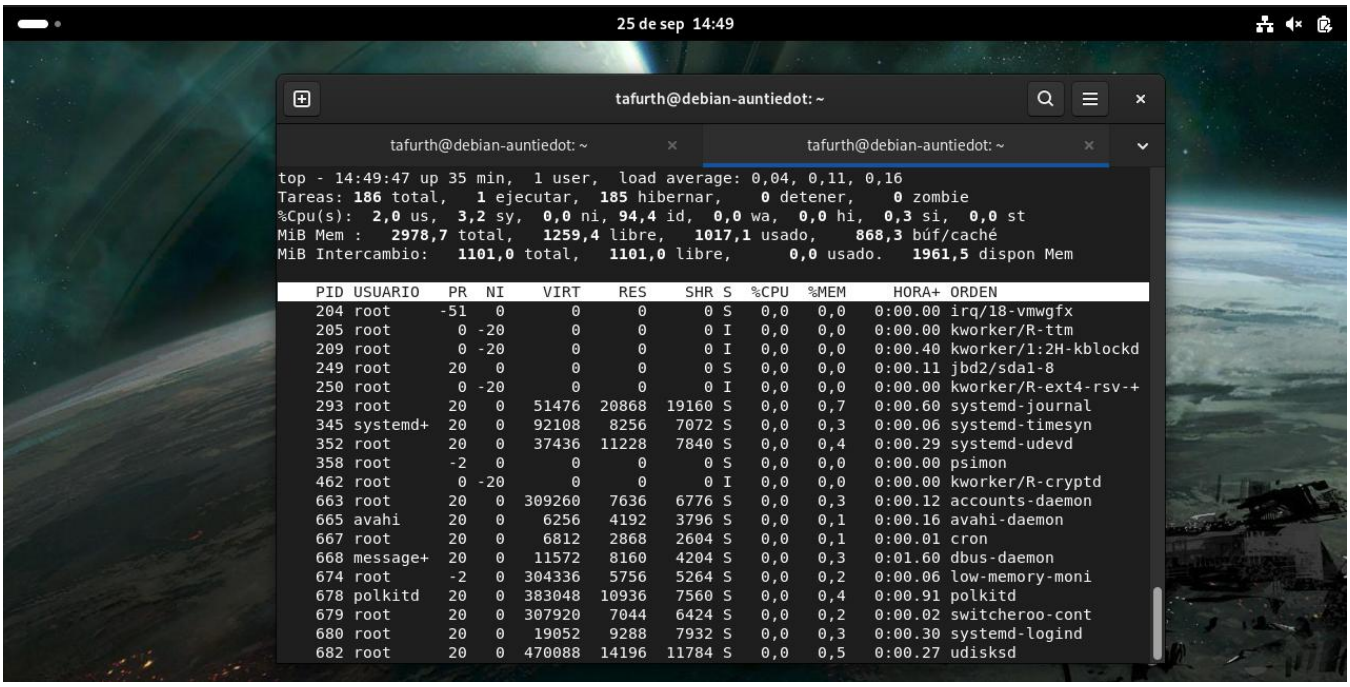
PROCESO	STATE	VMRSS	THREAD
/USR/LIBECEX/UBIS-X11	S (sleeping)	130212 Kb	4

En este caso el proceso está suspendido, consume 130212 Kb me memoria RAM aproximadamente 13.012 Mb y usa 4 hilos activos para su procesamiento.

A.2 Monitoreo del procesador y recursos

Figura 67

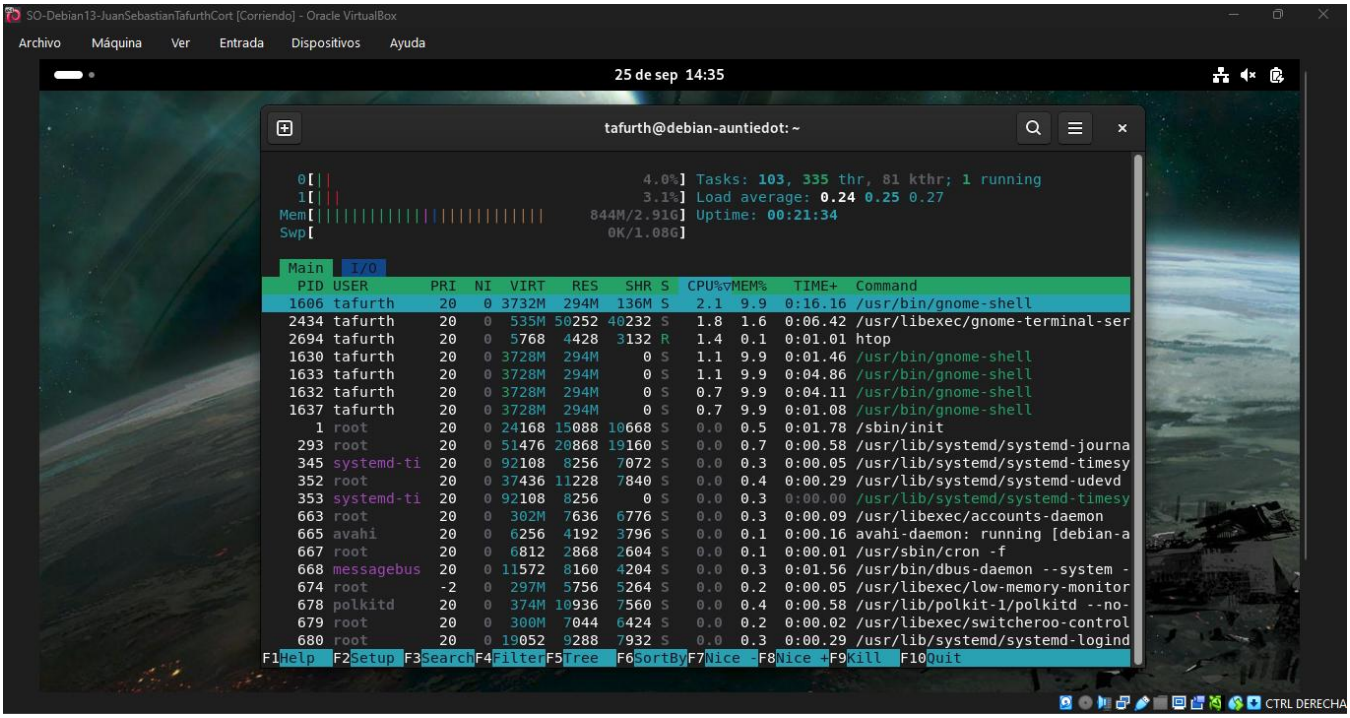
Captura de pantalla – Máquina Virtual, Terminal



Nota. Se realizo la ejecución del comando **“top”** con el propósito de visualizar los procesos activos en el sistema operativo.

Figura 68

Captura de pantalla – Máquina Virtual, Terminal



Nota. Se realizó la ejecución del comando **<<Htop>>** con el propósito de visualizar los

procesos activos en el sistema operativo.

Analizando ambos comandos, que permiten visualizar de manera detallada la administración de tareas que realiza el sistema, se evidenció que en términos generales son semejantes. Ambos muestran exactamente el mismo tipo de información: PID, usuario, consumo de memoria RAM, CPU, memoria virtual, tiempo y el proceso. En los valores presentados en cada categoría no se evidencia un desfase relevante entre los dos comandos, a excepción de la variable del tiempo, la cual muestra en algunos procesos una leve diferencia que no se aleja demasiado entre un valor y otro en la mayoría de los casos.

Por ejemplo, podemos analizar los procesos que permiten evidenciar este desfase en la categoría del tiempo; se elegirán los procesos con el PID 293 y 665.

- Proceso PID 293:
  - Información comando "Top"

PID	USUARIO	VIRT	SHR	%CPU	%MEM	HORA+	ORDEN
293	Root	51476	19160	0.0	0.7	00:00:60	Systemd-journal

  - Información comando "Htop"

PID	USUARIO	VIRT	SHR	%CPU	%MEM	HORA+	ORDEN
293	Root	51476	19160	0.0	0.7	00:00:58	Systemd-journal

Como se puede evidenciar en todas las categorías los datos presentados coinciden en las interfaces de ambos comandos, a excepción de la Hora teniendo un desfase de 0.2 segundos.

- Proceso PID 665:
  - Información comando "Top"

PID	USUARIO	VIRT	SHR	%CPU	%MEM	HORA+	ORDEN
665	Avahi	6256	4192	0.0	0.1	00:00:16	Avahi-Daemon…

- Información comando "Htop"

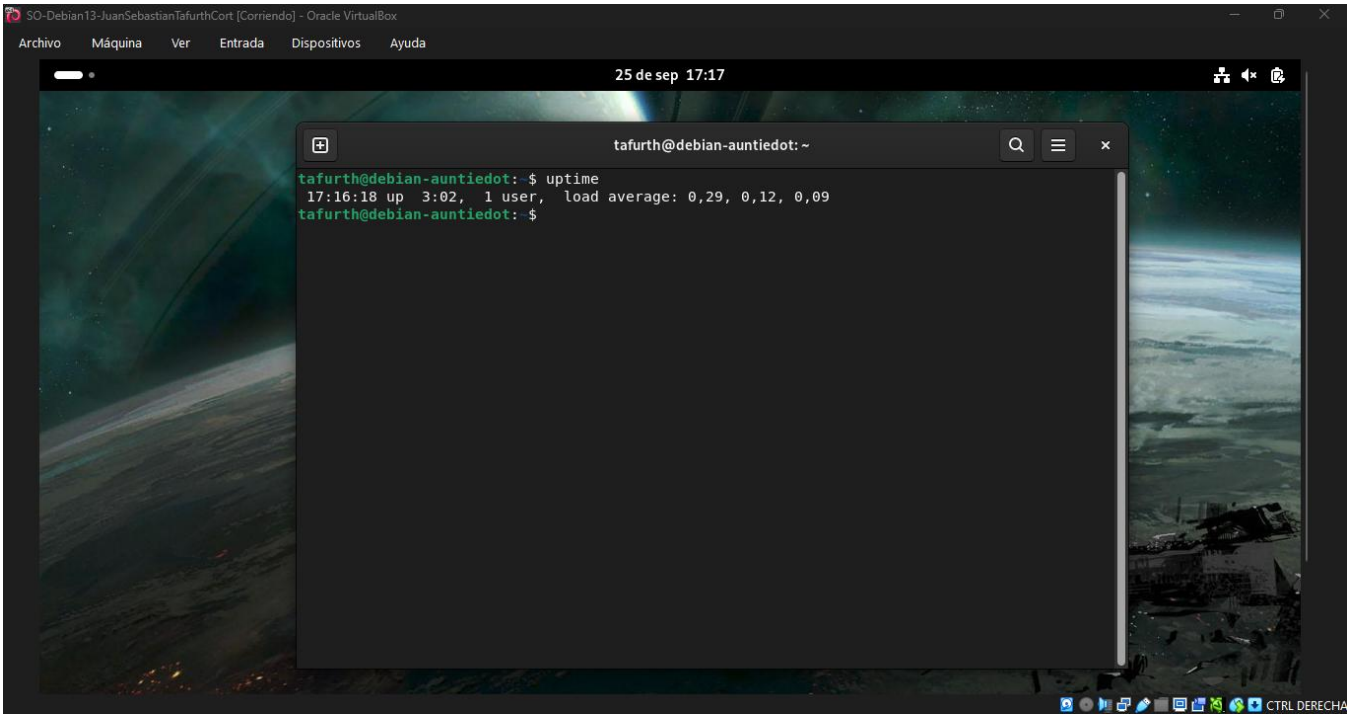
PID	USUARIO	VIRT	SHR	%CPU	%MEM	HORA+	ORDEN
665	Avahi	6256	4192	0.0	0.1	00:00:16	Avahi-Daemon…

En este caso, toda la información de todas las categorías coincide exactamente en las interfaces de ambos comandos, e incluso la variable del tiempo presenta los mismos valores. Esto quiere decir que el desfase no se presenta de manera general en todas las lecturas: en algunas se puede presentar una variación ligera, en otras puede ser mayor y, en última instancia, puede no llegar a presentarse.

Otra distinción notable es que la interfaz que nos presenta el comando **<<Htop>>** es más intuitiva y vistosa, pues expone una interfaz colorida que permite distinguir entre los distintos procesos existentes y los valores de sus categorías, facilitando la lectura de estos datos al usuario. Adicionalmente, en la parte inferior presenta un menú con más herramientas que el usuario puede ejecutar, el cual funciona como un menú de ayuda sin necesidad de acceder a nada más. En cambio, la interfaz del comando «top» es más plana, sin una distinción de colores u otro indicio que permita diferenciar las categorías y la información de los procesos.

Figura 69

Captura de pantalla – Máquina Virtual, Terminal



Nota. Ejecución del comando Uptime.

El comando **«Uptime»,** al ejecutarse sin parámetros, nos devuelve varios bloques de información: la hora actual en la que se ejecuta el comando, el tiempo que lleva encendida la máquina, la cantidad de usuarios conectados al momento de la ejecución y la carga media del sistema.

La carga del sistema nos muestra los procesos en espera y en ejecución que buscan utilizar los recursos del computador durante los últimos 1, 5 y 15 minutos, representados por tres valores en números decimales. En este caso, se obtuvo que la sobrecarga del sistema fue:

TIEMPO	1 MINUTO	5 MINUTOS	15 MINUTOS
LOAD AVERAGE:	0.29	0.12	0.09

Esto quiere decir que en el último minuto 0,29 procesos demandaban recursos de la máquina, en los últimos 5 minutos 0,12 procesos demandaban recursos y durante los últimos 15 minutos tan solo 0,09 procesos buscaban recursos. Como se puede evidenciar, en este caso la cantidad de procesos fue aumentando gradualmente a través del tiempo; es decir, la sobrecarga del sistema aumentó con el tiempo, pasando de 0,09 a 0,12 y finalmente a 0,29. El caso contrario se presentaría si la sobrecarga del último minuto fuera menor a la de los periodos anteriores; en tal escenario, la carga estaría disminuyendo.

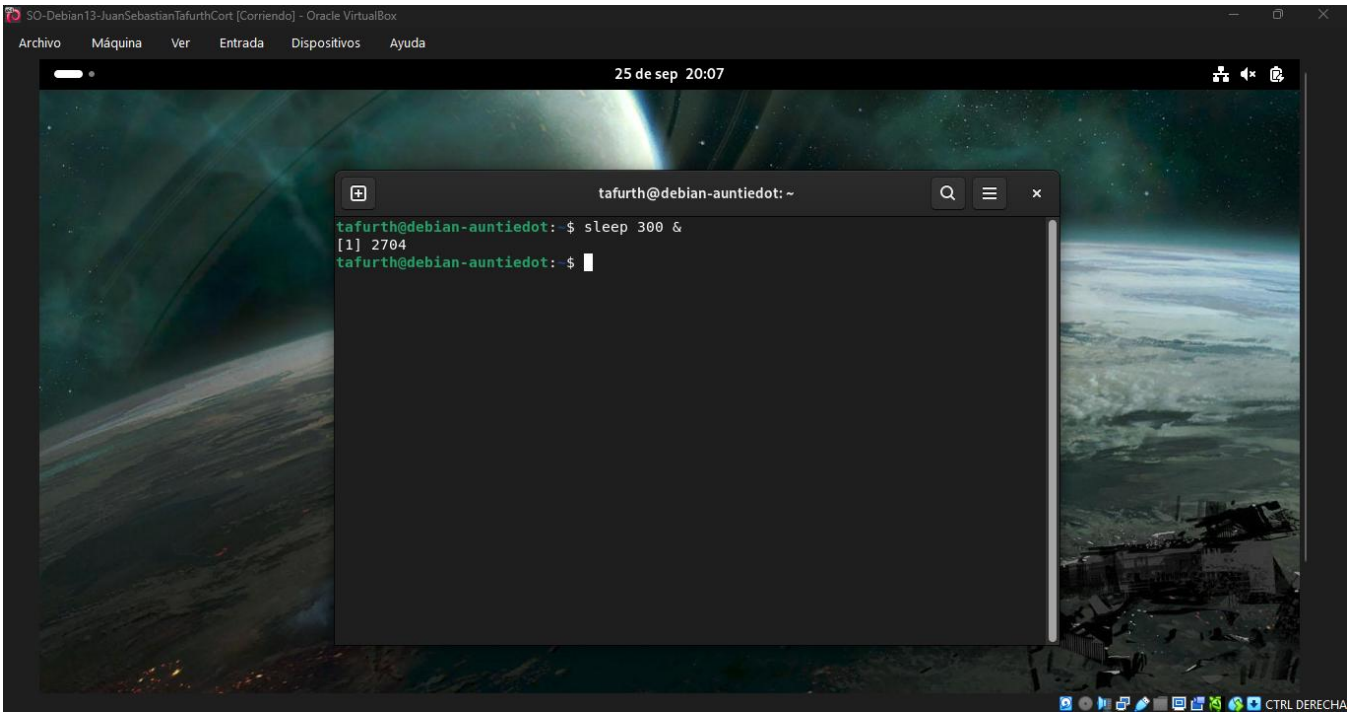
Por otro lado, para determinar si la carga es alta o baja, es necesario conocer la cantidad de núcleos que tiene la CPU de la máquina, debido a que el número de núcleos representa el límite de trabajo en paralelo que soporta el sistema sin generar colas de espera. En este caso, a la máquina virtual se le asignaron 2 núcleos de los 4 disponibles, por lo que una carga de 2.0 sería

la capacidad máxima recomendada de trabajo que el sistema soporta sin sobrecargarse.

A.3 Señales y prioridad

Figura 70

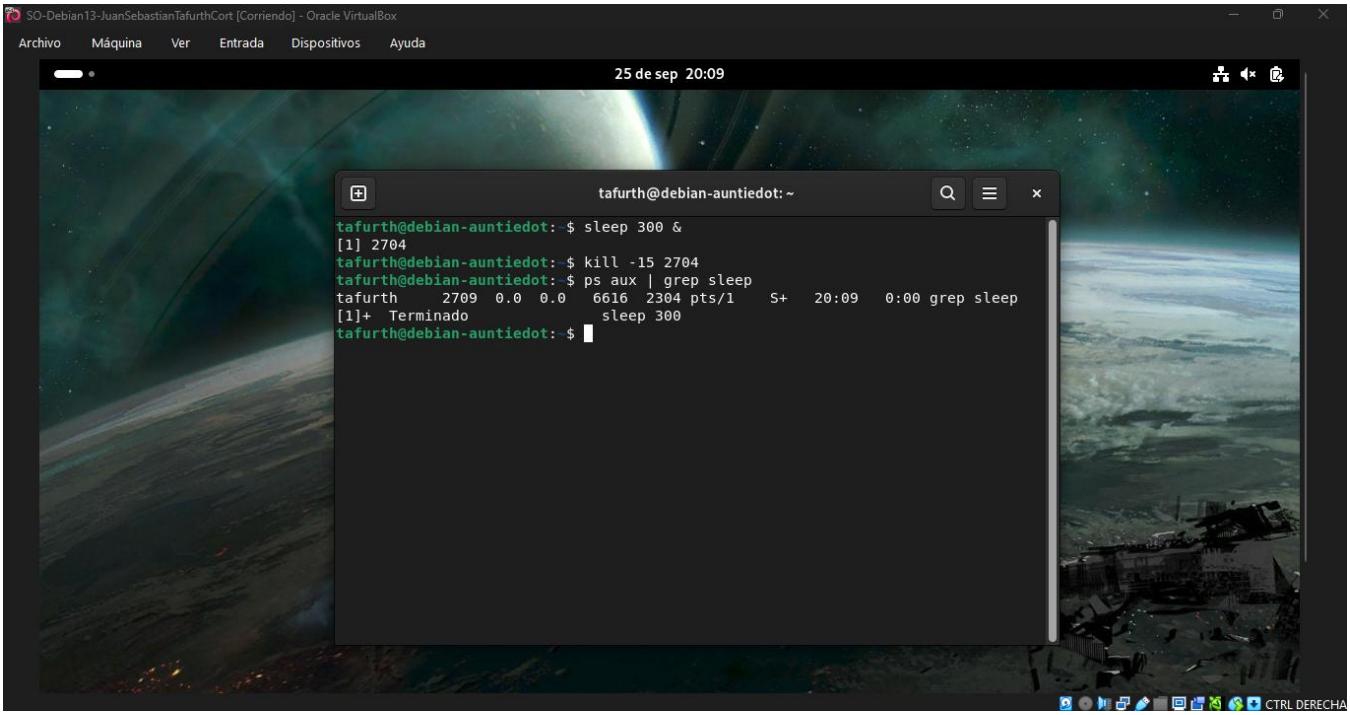
Captura de pantalla – Máquina Virtual, Terminal



Nota. Iniciación del proceso **<<Sleep 300 &>>** bajo el PID 2704

Figura 71

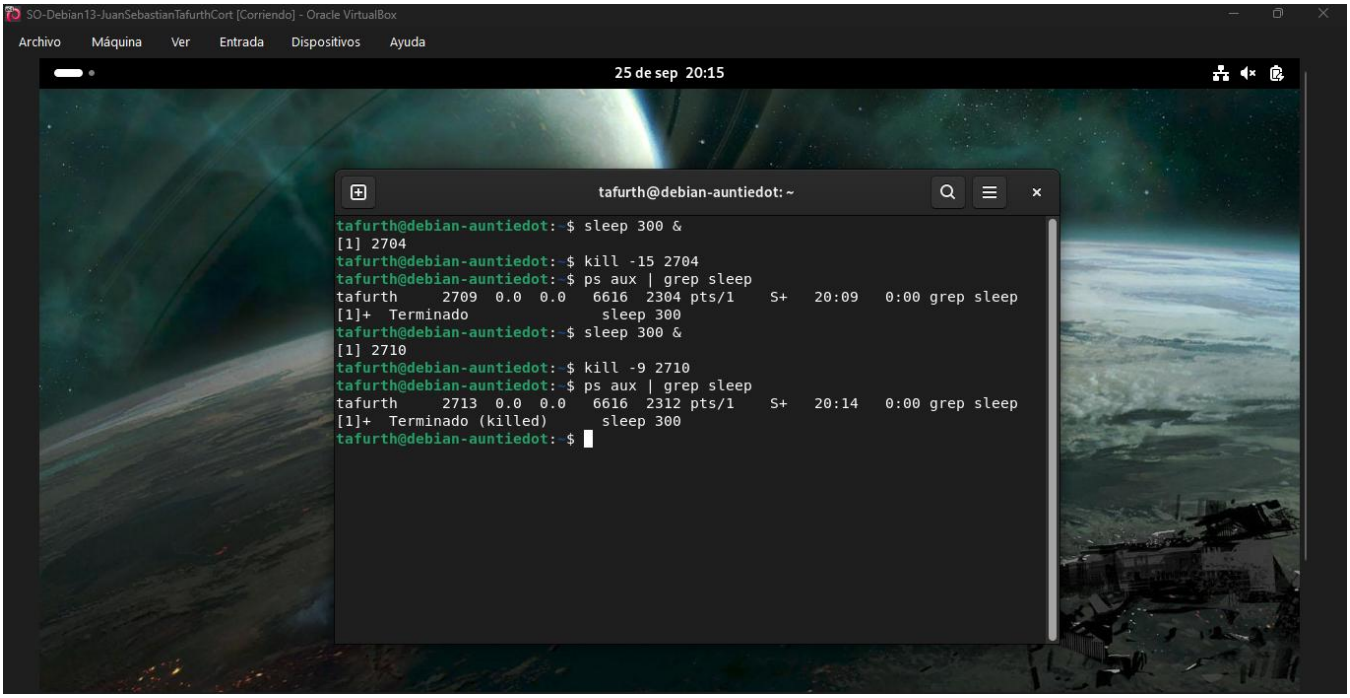
Captura de pantalla – Máquina Virtual, Terminal



Nota. Se finalizo el proceso con el comando **<<kill -15 2704>>,** adicionalmente se utilizó **<<Ps aux | grep sleep>>** con el fin de confirmar la finalización del proceso y efectivamente lo fue, como se evidencia en la figura 70 en la última línea.

Figura 72

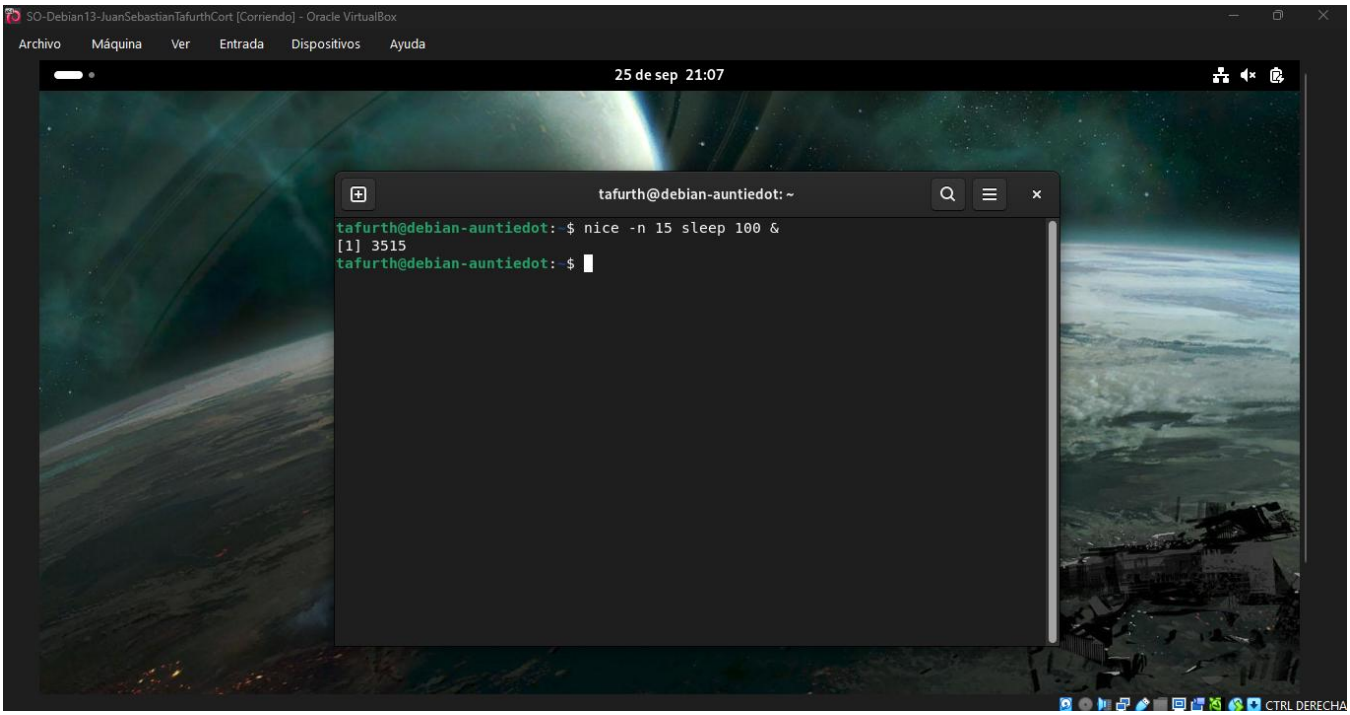
Captura de pantalla – Máquina Virtual, Terminal



Nota. Iniciación de nuevo proceso con sleep 300 bajo el PID 2704 & y finalización del mismo con <<kill -9 2704>> usando adicionalmente el comando <<Ps aux | grep sleep>> para confirmar la finalización del proceso.

Figura 73

Captura de pantalla – Máquina Virtual, Terminal

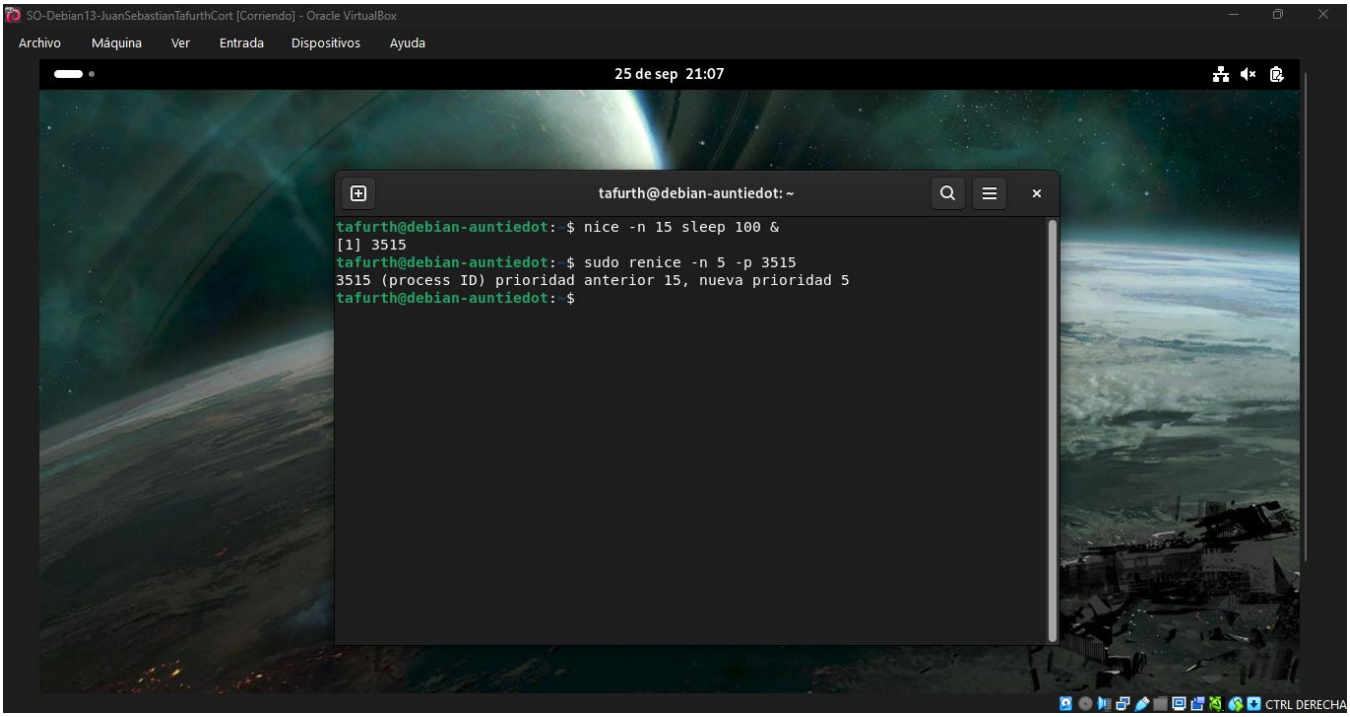


Nota. Inicialización de proceso sleep 100 **con <<nice -n 15 <comando> &>>** bajo el PID 3493

y asignándole una prioridad de 15.

Figura 74

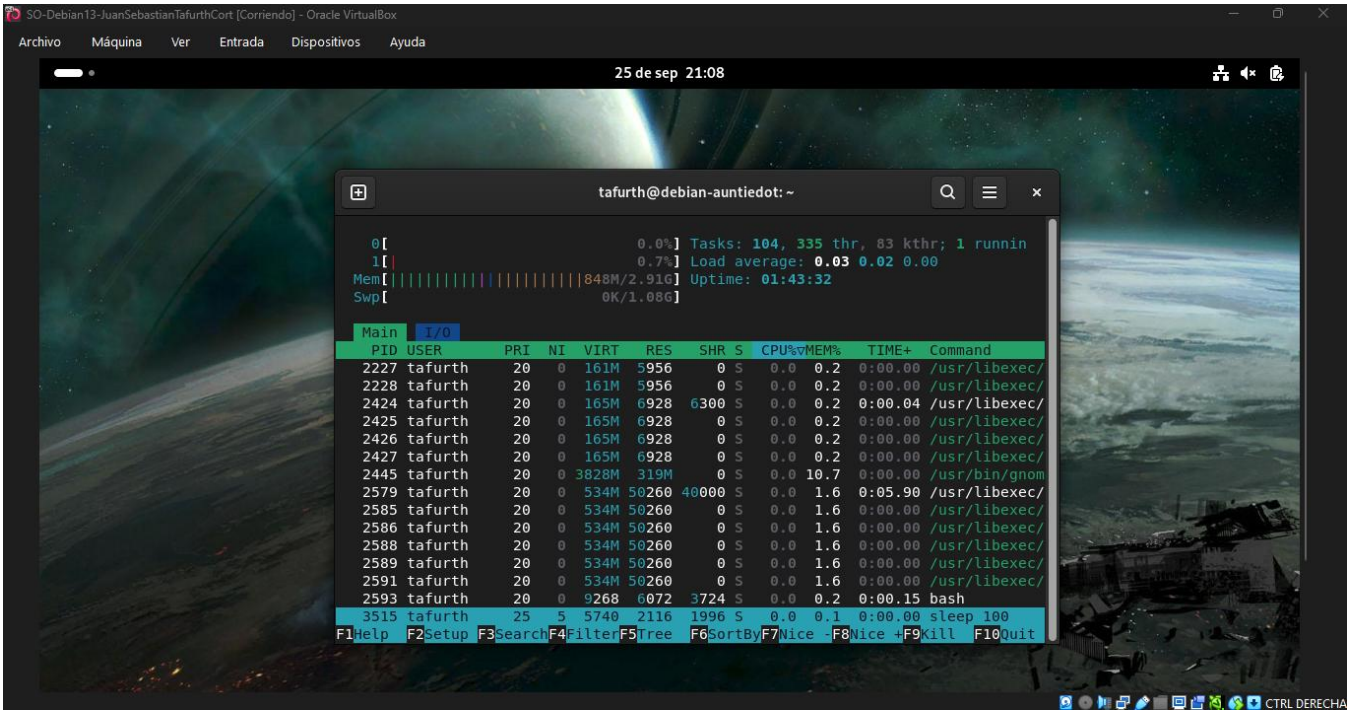
Captura de pantalla – Máquina Virtual, Terminal



Nota. Con el comando **<<renice -n 5 -p 3493>>** usando sudo con el fin no tener limitante a la hora de la ejecución por permisos. Se realizo efectivamente el cambio de prioridad del proceso con el PID 3493 de 15 a 5.

Figura 75

Captura de pantalla – Maquina virtual, Terminal



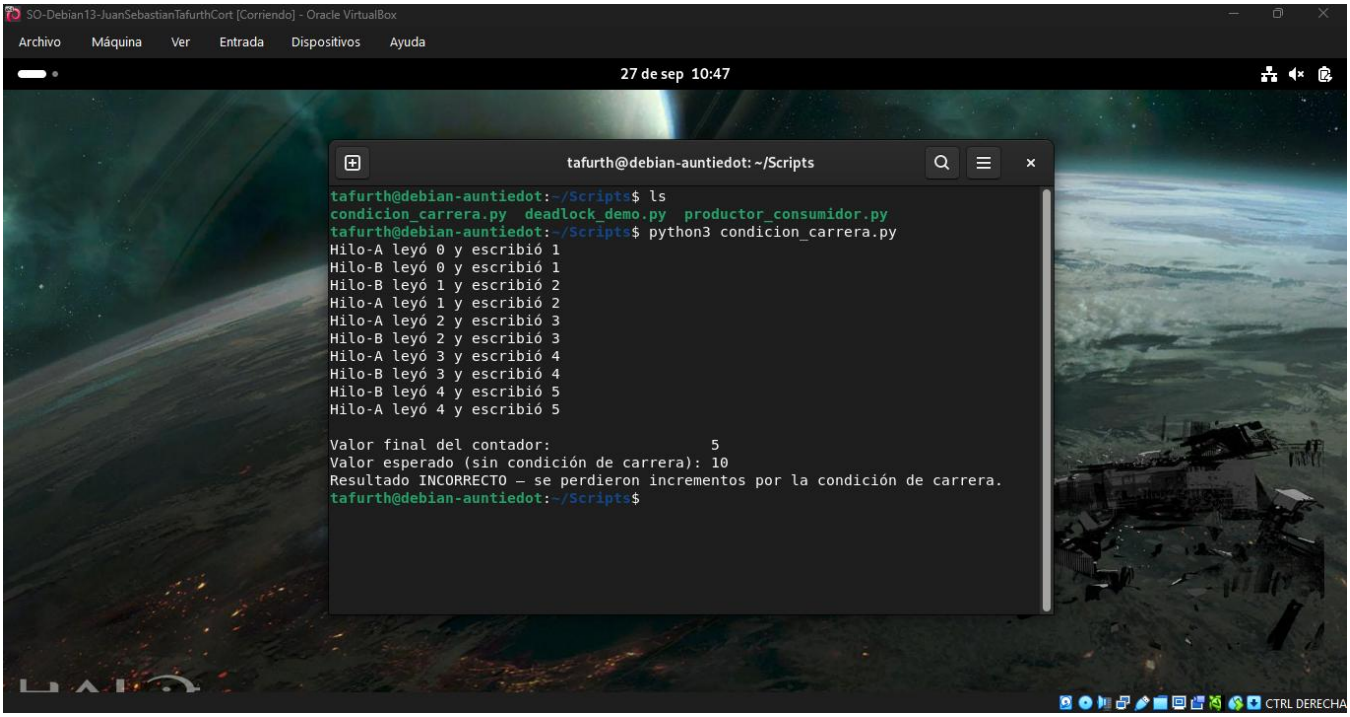
Nota. Se verifico con el administrador de tareas del comando **<<Htop>>** el cambio de prioridad del proceso bajo el PID 3515 y en efecto se realizó el cambio, siendo visible en la imagen siendo el proceso el último en la lista.

PARTE B — Laboratorio experimental de concurrencia (Unidad 4)

B.1 Script 1 — condicion_carrera.py

Figura 76

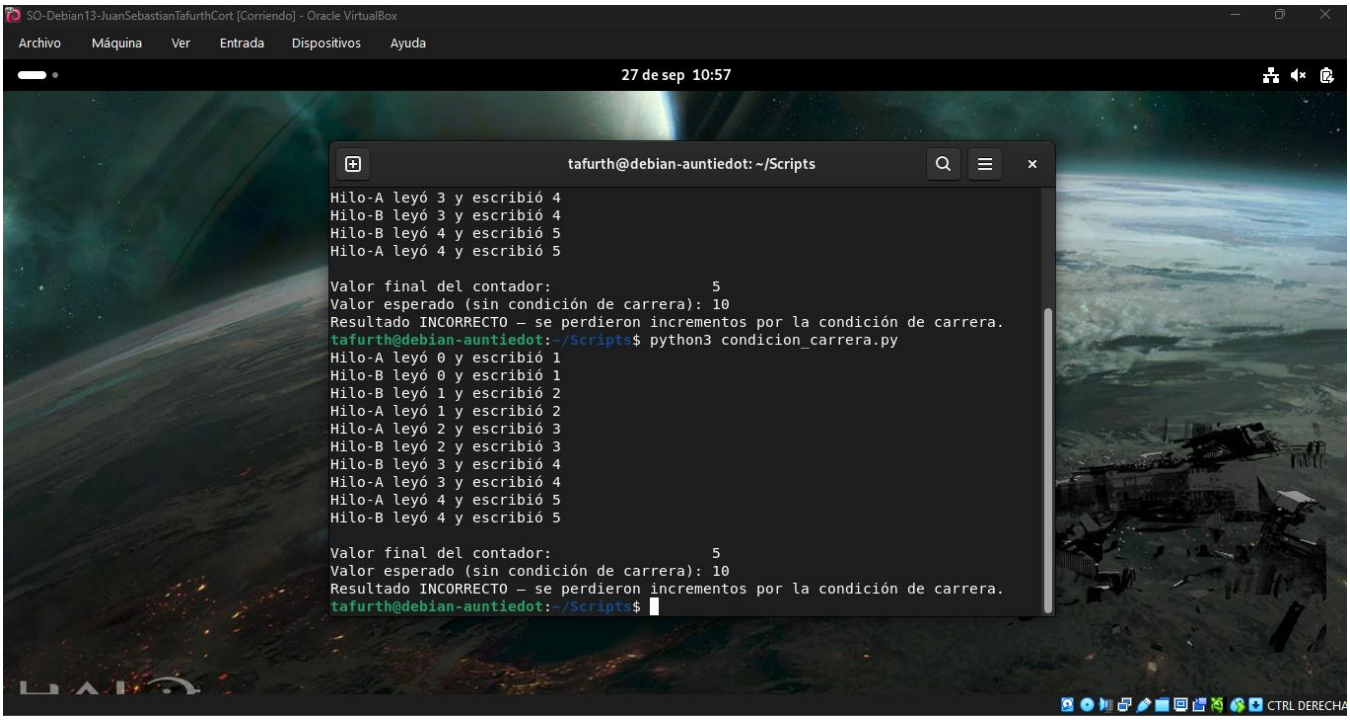
Captura de pantalla – Máquina virtual, Terminal



Nota. Ejecución #1 del script **<<condición_carrera.py>>** sin protección.

Figura 77

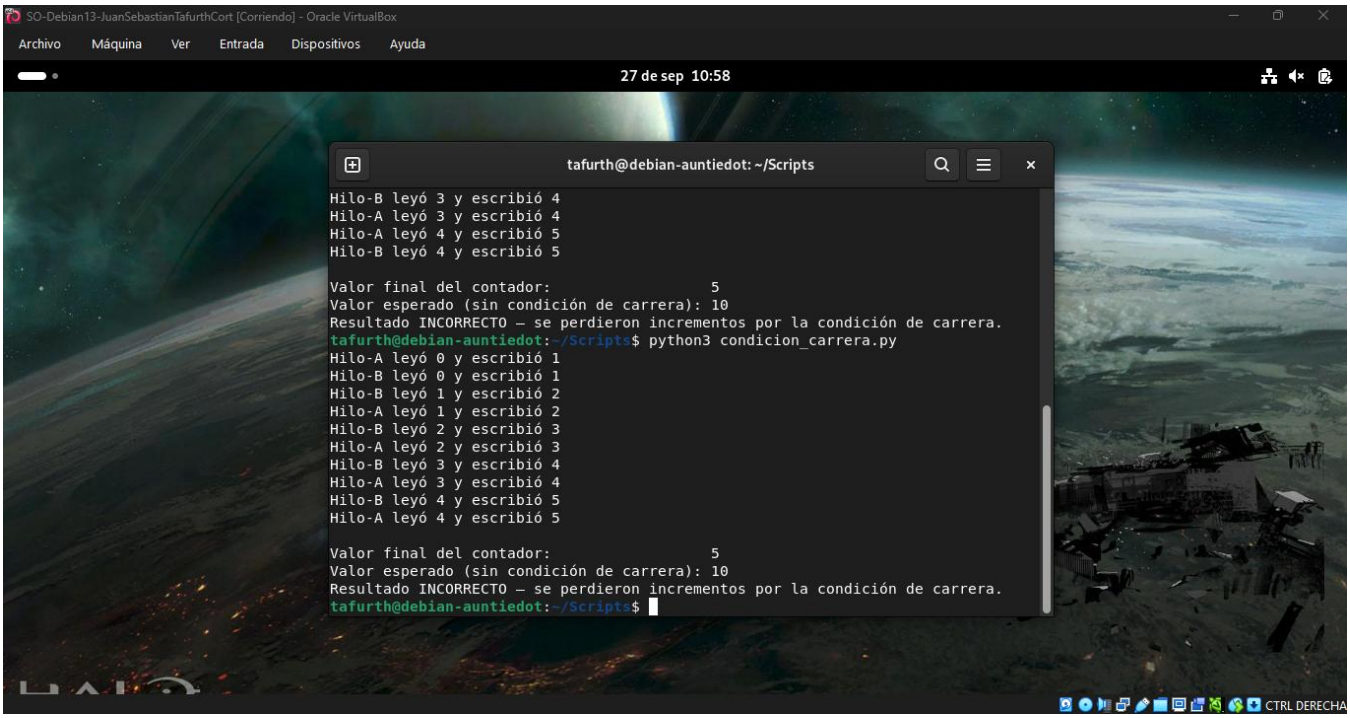
Captura de pantalla – Máquina virtual, Terminal



Nota. Ejecución #2 del script **<<condición_carrera.py>>** sin protección.

Figura 78

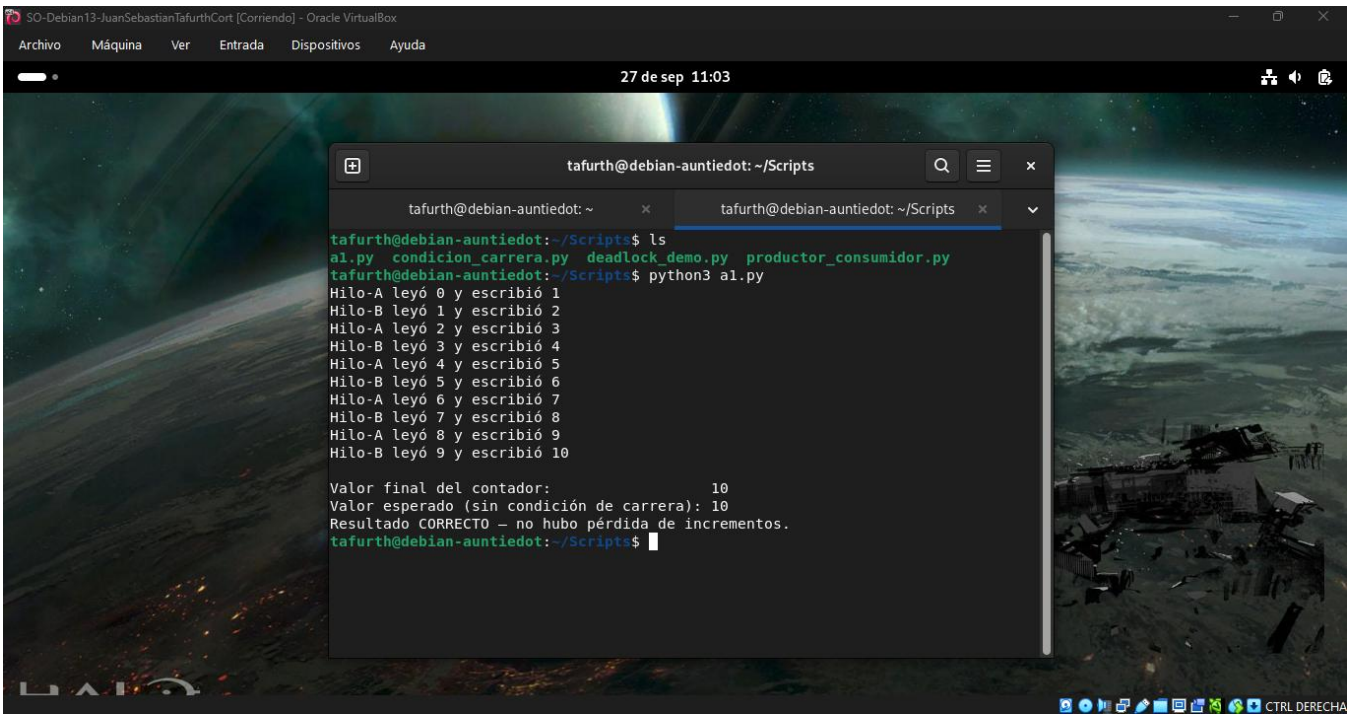
Captura de pantalla – Máquina virtual, Terminal



Nota. Ejecución #3 del script **<<condición_carrera.py>>** sin protección.

Figura 79

Captura de pantalla – Máquina virtual, Terminal



Nota. Ejecución #4 del script **<<condición_carrera.py>>** con protección activa.

Una vez realizadas las tres ejecuciones del script «condición_carrera.py», se analizaron y compararon los resultados de cada prueba. Se observó que los valores obtenidos en las tres ejecuciones fueron los mismos, sin ningún tipo de variación entre ellos. El valor final del contador se mantuvo en 5, siendo un resultado erróneo en contraste con el valor esperado (sin condición de carrera), que era de 10. Por otro lado, una vez que se aplicó la protección en el script, el valor final del contador coincidió con el valor esperado (10), obteniendo así un resultado correcto sin el efecto generado por la condición de carrera.

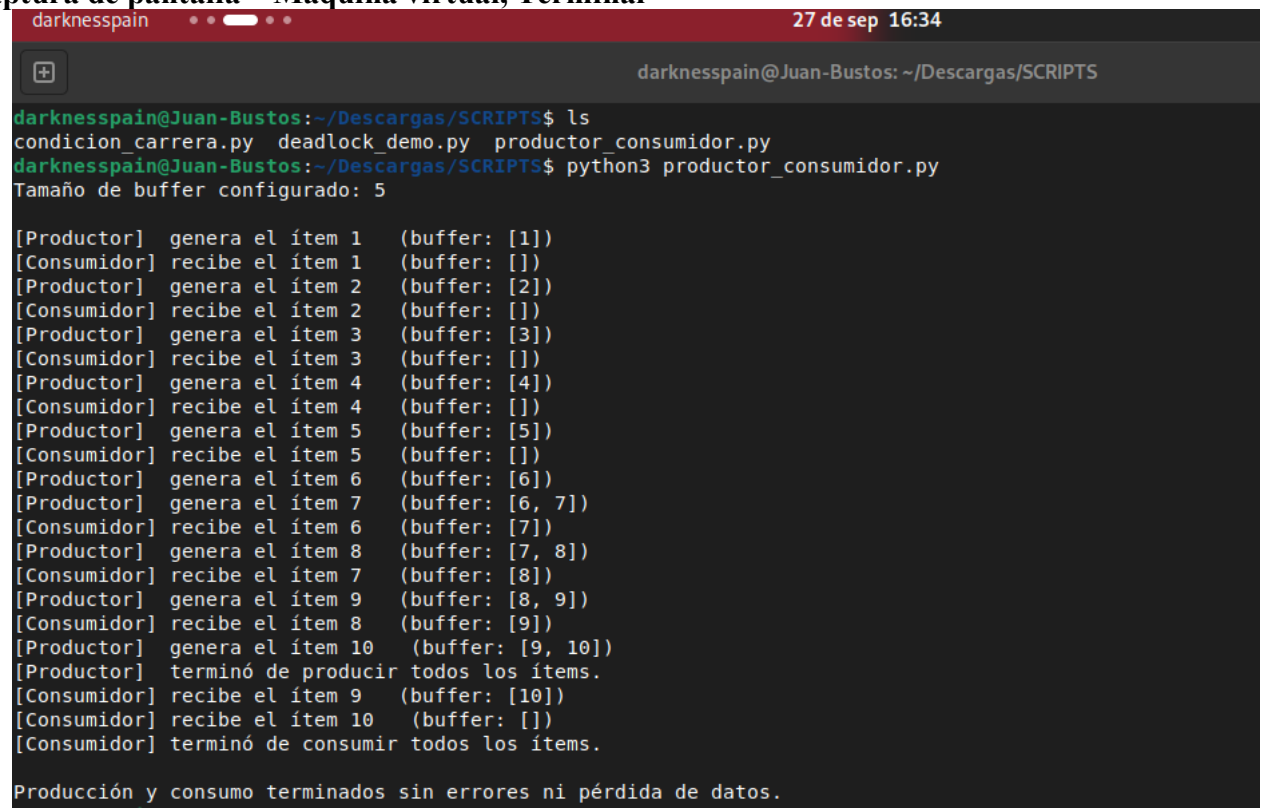
¿Qué predijeron que iba a pasar?	**¿Qué pasó realmente?**	**¿Por qué ocurrió esa diferencia (o coincidencia)?**
Por cada repetición realizada, los hilos leerán la variable "valor_leído" y ambas incrementarán el valor +1. Al ser 2 hilos, cada uno aumentando la variable 5 veces, el valor final del contador será de 10. cumpliendo la condición de que el contador es igual al número de repeticiones.	Al realizar 3 veces la ejecución del script, los hilos leyeron la variable de "valor_leído" y ambas incrementaron +1 el valor de la variable en las 5 repeticiones, a la vez guardando el mismo valor la otra variable "contador". Dando como resultado: • Valor final del contador = 5 • Valor esperado = 10 Generando un resultado incorrecto, incumpliendo la condición.	La diferencia se generó debido a que ambos hilos acceden de "manera simultánea" y leen la variable "valor_leído". Al no haber un orden de ejecución, un hilo toma el valor y le incrementa +1 y el otro hilo hace lo mismo, pero con el valor original sin modificar, al ya haber leído el dato inicial y no el modificado; generando así que cada hilo genere 2 valores de la variable original. En este caso ese valor solo llega hasta 5 para ambos hilos.
Por cada repetición realizada, los hilos leerán la variable "valor_leído" y ambas incrementarán el valor +1. Al ser 2 hilos, cada uno aumentando la variable 5 veces, el valor final del contador será de 10. Generando este resultado al ya estar activa la protección. cumpliendo la condición de que el contador es igual al número de repeticiones.	Al realizar la última ejecución del script, esta vez con la protección activa, los hilos leyeron la variable de valor leído y ambos incrementaron +1 el valor de la variable en las 5 repeticiones. Esta vez los resultados obtenidos fueron: Valor final del contador = 10 Valor esperado = 10 Generando un resultado correcto, cumpliendo la condición.	La coincidencia se generó debido a que la protección activada establece un orden a la ejecución y modificación de la variable "valor_leído", permitiendo que solo 1 hilo genera estas acciones a la vez, evitando la generación de valores desactualizados, para que al momento de que el otro hilo inicie su ejecución tome el valor actualizado.

B.2 Script 2 — productor_consumidor.py

El script «productor_consumidor.py» implementa la solución clásica al problema del productor-consumidor usando la librería threading de Python. La idea aquí es que tenemos un hilo que produce datos y los va guardando en un búfer compartido, y otro hilo que los va sacando para procesarlos. Para que no se armen problemas de concurrencia ni se pisen entre ellos, el programa usa tres semáforos:
• mutex: Un semáforo binario para la exclusión mutua, que asegura que solo un hilo a la vez pueda tocar el búfer.
• espacios_libres: Un semáforo contador que arranca con el tamaño máximo del búfer (TAMANO_BUFFER) y pone a esperar al productor si el búfer se llena.
• elementos_disponibles: Un semáforo contador que arranca en 0 y pone a esperar al consumidor si no hay nada en el búfer.

Paso 1: Ejecución del script con buffer de tamaño 5 (configuración inicial). Se ejecutó el script tal como fue entregado y se observó el orden en que el productor genera los datos y el consumidor los extrae.

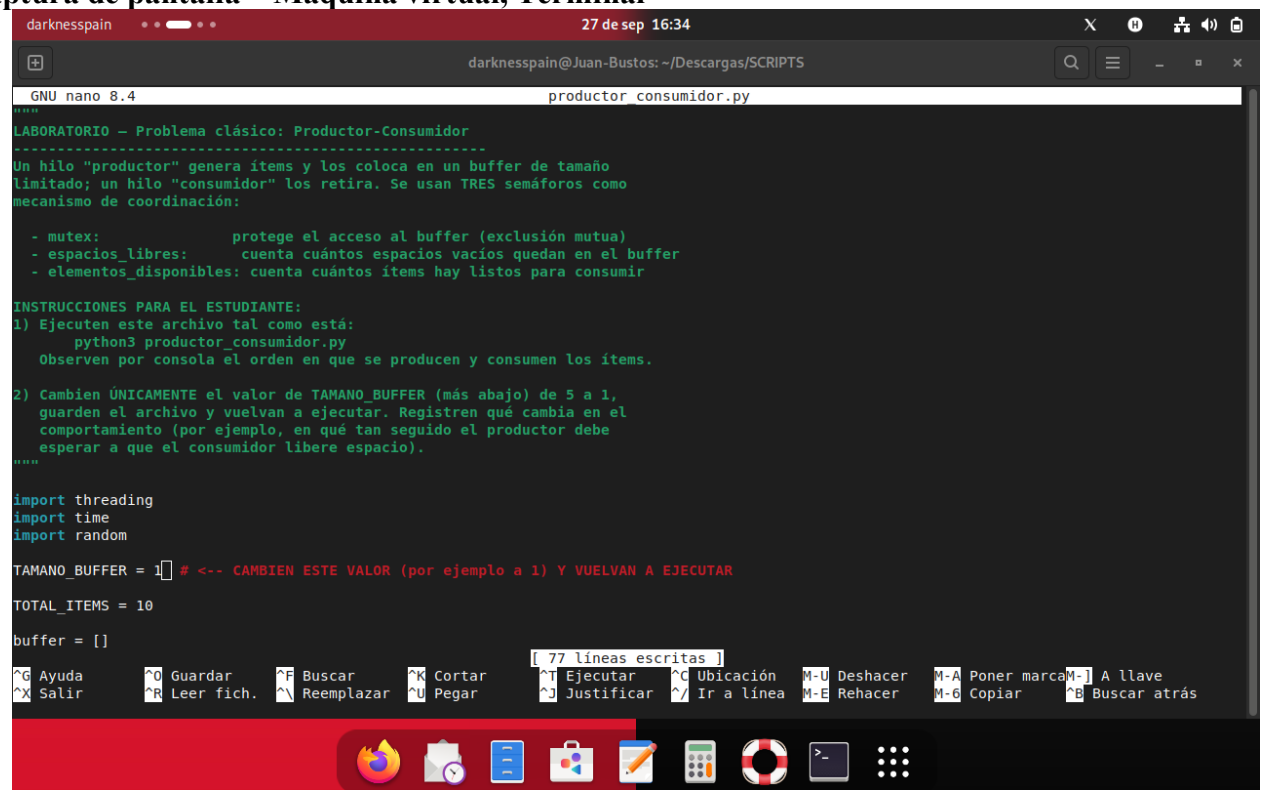
Figura 80
Captura de pantalla – Máquina virtual, Terminal



Nota. Ejecución del script «productor_consumidor.py» con un tamaño de buffer configurado en 5 elementos. Se observa cómo el productor genera varios ítems de forma consecutiva antes de que el consumidor comience a retirarlos, evidenciando el desacoplamiento temporal de ambos hilos.

Paso 2: Modificación del tamaño del buffer en nano (TAMANO_BUFFER = 1). Siguiendo las instrucciones de la guía, se abrió el archivo con el editor nano y se cambió la variable TAMANO_BUFFER de 5 a 1, guardando los cambios con Ctrl+O.

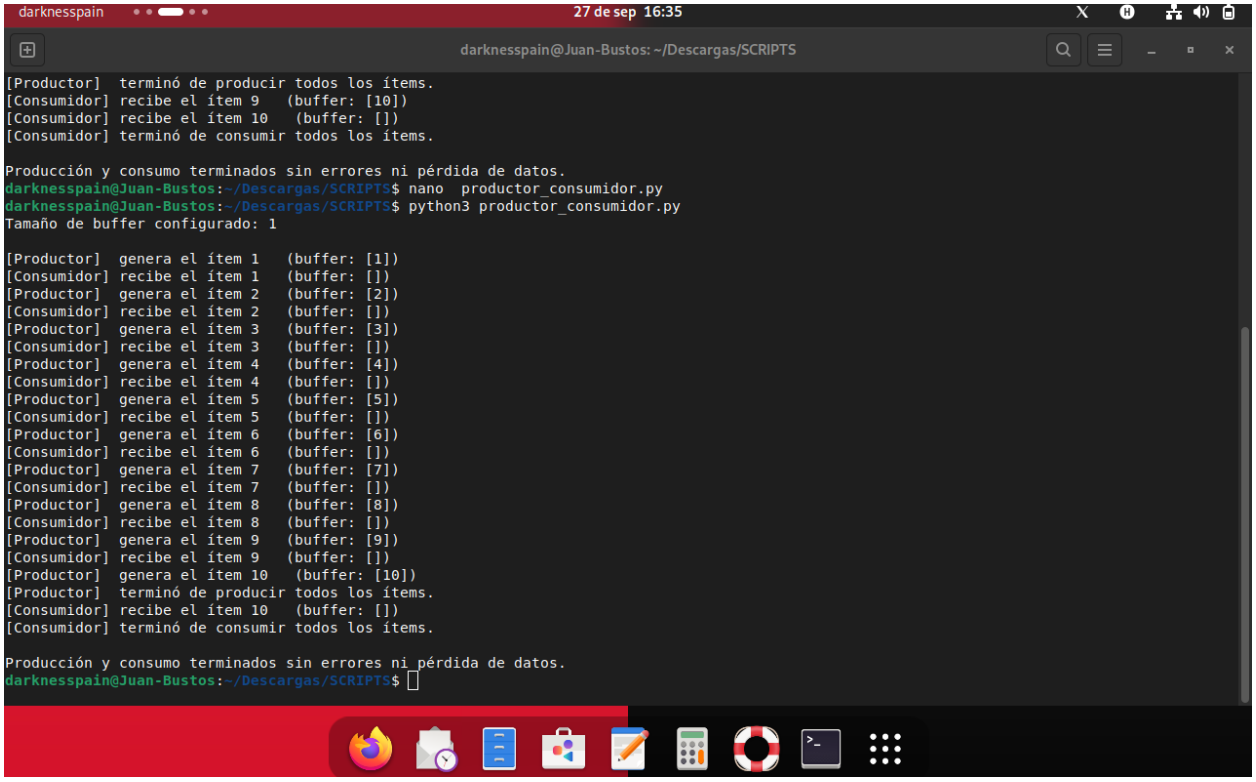
Figura 81
Captura de pantalla – Máquina virtual, Terminal



Nota. Edición del script «productor_consumidor.py» en el editor nano, modificando la variable TAMANO_BUFFER a un valor de 1 para restringir la capacidad máxima del búfer a un único elemento.

Paso 3: Ejecución del script con buffer de tamaño 1. Se ejecutó nuevamente el script para registrar qué cambia en el comportamiento por consola.

Figura 82
Captura de pantalla – Máquina virtual, Terminal



Nota. Ejecución del script «productor_consumidor.py» con TAMANO_BUFFER = 1. Se evidencia una alternancia estricta entre productor y consumidor: cada vez que el productor genera 1 ítem, el búfer se llena de inmediato y el productor debe esperar obligatoriamente a que el consumidor lo retire antes de poder generar el siguiente.

Comparación y análisis de comportamiento:
Al comparar las dos ejecuciones, se nota clarísimo cómo el tamaño del búfer cambia toda la dinámica: Con TAMANO_BUFFER = 5, el productor puede adelantarse y meter hasta 5 elementos seguidos en el búfer sin que el consumidor lo frene. Esto desacopla el trabajo de ambos hilos y hace que el sistema fluya mucho mejor.
En cambio, cuando pusimos TAMANO_BUFFER = 1, la capacidad de amortiguar se redujo al mínimo: el productor mete un ítem y de una queda bloqueado hasta que el consumidor lo saca. Se vuelve una sincronización estricta paso a paso (como un pasamanos). En ninguno de los dos casos se perdieron datos porque el mutex protegió siempre el búfer.

¿Qué predijeron que iba a pasar?	¿Qué pasó realmente?	¿Por qué ocurrió esa diferencia (o coincidencia)?
Buffer = 5: Se predijo que el productor generaría varios elementos seguidos antes de que el consumidor tuviera que vaciar el buffer, permitiendo un ritmo asíncrono y fluido entre ambos hilos hasta alcanzar el límite de 5 elementos.	El productor generó varios ítems rápidamente (llenando posiciones en el buffer), y el consumidor los fue extrayendo a su propio ritmo sin bloquearse mutuamente de forma inmediata.	Coincidencia: Al tener una capacidad de 5 elementos, el búfer desacopla temporalmente las velocidades de producción y consumo. El semáforo de espacios vacíos permite hasta 5 inserciones consecutivas antes de obligar al productor a esperar.

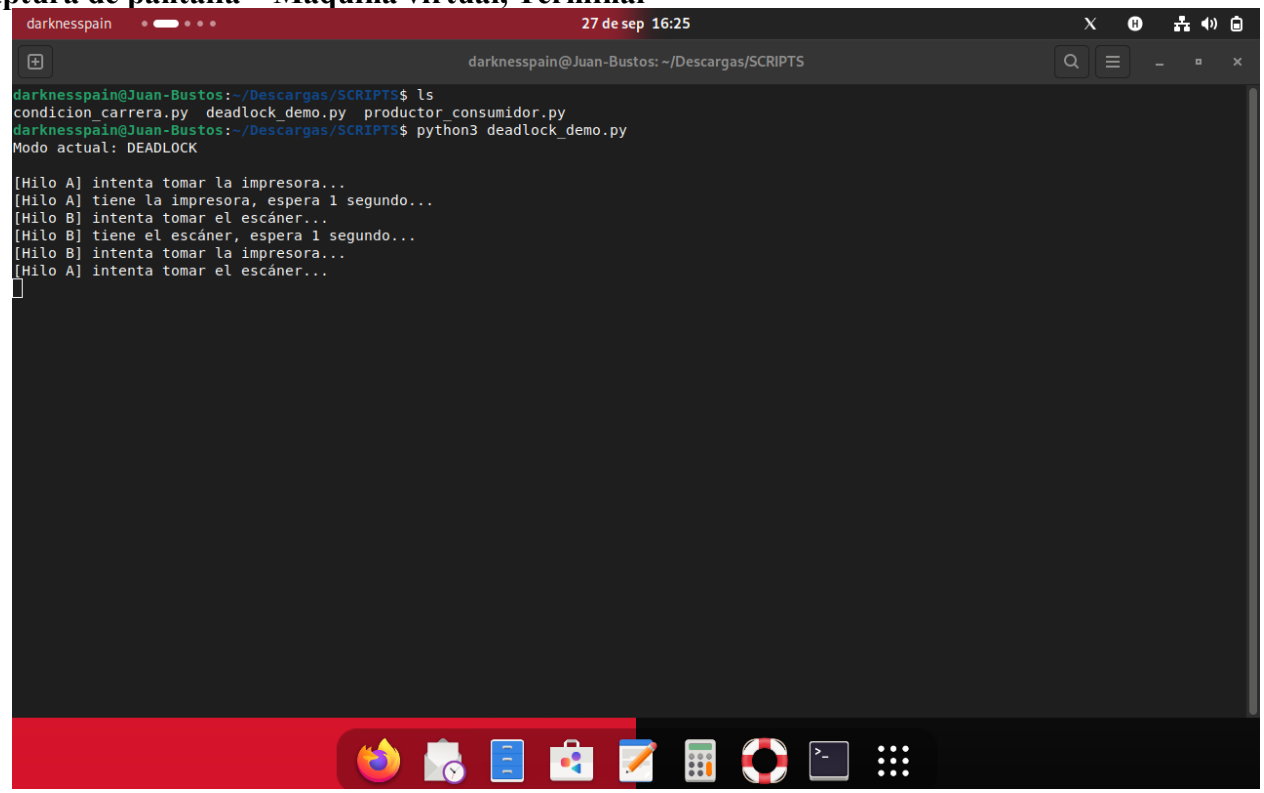
Buffer = 1: Se predijo que al reducir el búfer a 1, el productor y el consumidor quedarían obligados a alternarse estrictamente paso a paso (un ítem producido, un ítem consumido).	La salida por consola mostró una alternancia casi inmediata: cada vez que el productor colocaba 1 elemento, se bloqueaba hasta que el consumidor lo retiraba.	Coincidencia: Con capacidad 1, el espacio disponible se agota con cada inserción individual. El productor entra inmediatamente en espera (bloqueado por semáforo de espacios) hasta que el consumidor retira el dato y envía la señal de espacio libre.

B.3 Script 3 — deadlock_demo.py

El script «deadlock_demo.py» sirve para ver en vivo cómo se produce un interbloqueo (deadlock) y cómo evitarlo en la práctica. El programa simula dos hilos (Hilo A e Hilo B) que necesitan usar dos recursos exclusivos al mismo tiempo: una impresora y un escáner, ambos protegidos con candados de exclusión mutua (threading.Lock()).

Paso 1: Ejecución en modo DEADLOCK (provocación del interbloqueo). Se ejecutó el archivo configurado con MODO = "DEADLOCK". Como se esperaba, el programa quedó colgado indefinidamente al llegar al punto de retención y espera mutua.

Figura 83
Captura de pantalla – Máquina virtual, Terminal

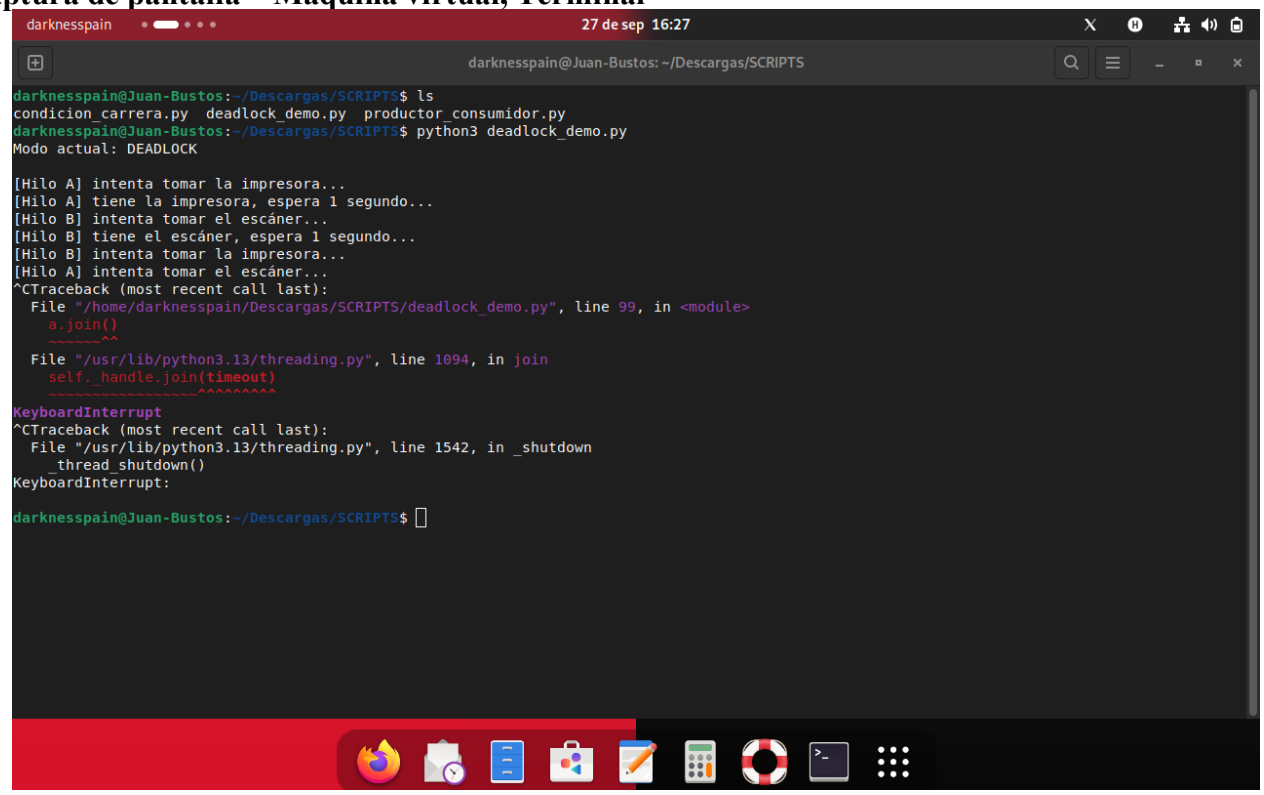


Nota. Ejecución del script «deadlock_demo.py» en modo DEADLOCK. El Hilo A toma la impresora y espera el escáner, mientras que el Hilo B toma el escáner y espera la impresora. Ambos hilos quedan permanentemente bloqueados a la espera del recurso retenido por el otro.

Paso 2: Interrupción de la ejecución tras 15 segundos de espera. Habiendo transcurrido más de 15

segundos sin actividad ni respuesta por consola, se detuvo la ejecución forzadamente mediante la combinación Ctrl+C.

Figura 84
Captura de pantalla – Máquina virtual, Terminal

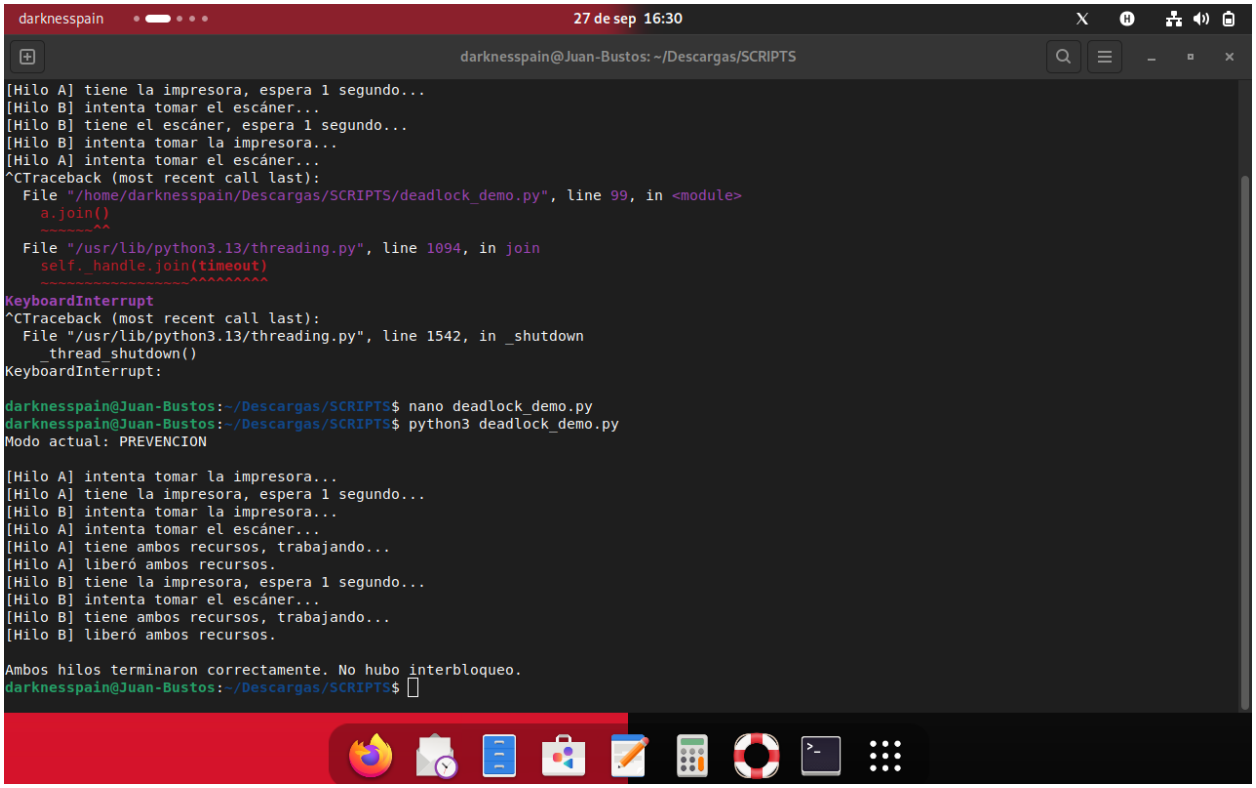


Nota. Interrupción forzada del programa interbloqueado mediante Ctrl+C en la terminal, evidenciándose la traza de excepción KeyboardInterrupt al detener la espera de los hilos.

Identificación de las cuatro condiciones de Havender en el código:
Revisando el archivo «deadlock_demo.py», identificamos con exactitud dónde se da cada una de las cuatro condiciones necesarias de Havender:
1. Exclusión Mutua: La impresora y el escáner están creados como threading.Lock(). Esto significa que si un hilo toma el recurso, nadie más puede usarlo al mismo tiempo.
2. Retención y Espera: El Hilo A agarra la impresora (impresora.acquire()), se queda esperando 1 segundo (time.sleep(1)) con la impresora retenida, y luego intenta agarrar el escáner. Al mismo tiempo, el Hilo B retiene el escáner y pide la impresora. Ninguno suelta lo que ya tiene.
3. No Expropiación: En el código no hay ninguna instrucción para quitarle a la fuerza un recurso a un hilo. La única forma de soltarlo es que el mismo hilo llame voluntariamente a lock.release().
4. Espera Circular: En modo DEADLOCK, los hilos piden los recursos al revés: Hilo A pide primero impresora y luego escáner; mientras Hilo B pide primero escáner y luego impresora. Se forma un círculo donde A espera a B y B espera a A.

Paso 4: Activación de la prevención de interbloqueo (MODO = "PREVENCION"). Se abrió el archivo con nano, se modificó la variable a MODO = "PREVENCION" y se ejecutó nuevamente el script.

Figura 85
Captura de pantalla – Máquina virtual, Terminal



Nota. Ejecución de «deadlock_demo.py» en modo PREVENCION. Ambos hilos adquieren los recursos en el mismo orden jerárquico (primero la impresora, luego el escáner), permitiendo que el Hilo A trabaje y libere ambos recursos, para que posteriormente el Hilo B los use sin interbloqueos.

Análisis de la prevención del interbloqueo:
Al cambiar la variable a MODO = "PREVENCION", aplicamos la regla de oro de Havender: el orden jerárquico de recursos. Hicimos que ambos hilos pidan los recursos exactamente en el mismo orden (primero impresora, luego escáner). Con esto rompemos por completo la Espera Circular, porque si el Hilo A gana la impresora, el Hilo B no puede agarrar el escáner antes de tiempo; tiene que esperar la impresora primero. Al romperse esa sola condición, el interbloqueo queda totalmente eliminado y el programa corre sin trabarse.

¿Qué predijeron que iba a pasar?	¿Qué pasó realmente?	¿Por qué ocurrió esa diferencia (o coincidencia)?
Modo "DEADLOCK": Se predijo que el programa quedaría congelado indefinidamente debido a que dos hilos intentarían adquirir dos recursos (locks) en orden inverso, bloqueándose mutuamente.	El programa imprimió los primeros mensajes de adquisición parcial y se quedó completamente colgado sin avanzar ni finalizar, teniendo que forzar su cierre tras 15 segundos con Ctrl+C.	Coincidencia: Se cumplieron simultáneamente las 4 condiciones de Havender (exclusión mutua, retención y espera, no expropiación y espera circular). El Hilo A tomó la impresora y esperó el escáner, mientras el Hilo B tomó el escáner y esperó la impresora.

Modo "PREVENCION": Se predijo que al cambiar la variable a "PREVENCION", el programa ejecutaría todas sus tareas y terminaría de forma exitosa mostrando el mensaje final.	El programa ejecutó ambos hilos sin trabarse, imprimió todos los pasos de trabajo de los recursos y finalizó normalmente retornando al prompt de la terminal.	Coincidencia: Se rompió la condición de Espera Circular mediante una jerarquía u orden global en la adquisición de los locks (ambos hilos adquieren primero la impresora y luego el escáner), eliminando la posibilidad de interbloqueo.

Conclusiones y Análisis Crítico

Reflexiones del Equipo

- En mi experiencia, Juan Sebastián Tafurth Cortes, gracias a esta practica amplie mis conocimientos acerca del uso del sistema operativo de debian.

  Profundizando mas en temas como lo son el monitorio del procesador y los recursos del sistema con los comandos de <<Top>> y <<Htop>>, la finalización de procesos controlada gracias a los comando <<Kill> que permiten terminarlos y el caso de la condición de carrera que me pareció interesante debido a que ocurre solo por falta de orden de ejecución.

En mi experiencia, Juan Esteban Bustos Chilatra, esta segunda parte del laboratorio estuvo genial porque pasamos de ver la teoría a cacharrearle directamente a los procesos y la concurrencia en la terminal. Ya teniendo experiencia previa con Linux, me pareció muy bacano ver en vivo cómo interactúan las señales con el sistema: por ejemplo, la diferencia real entre mandar un kill -15 (donde el proceso se entera y puede cerrar sus cosas con calma) y clavarle un kill -9, que lo fulmina de inmediato desde el kernel sin dejarle decir ni pío. Por el lado de la concurrencia en Python, ver la condición de carrera en vivo te abre los ojos: si dos hilos intentan modificar la misma variable al tiempo sin un Lock, los datos se corrompen y se pierden incrementos de una forma absurda; pero apenas activas la exclusión mutua, todo cuadra a la perfección. Y finalmente con el tema del deadlock, es muy satisfactorio ver cómo se cumplen al pie de la letra las condiciones de Havender para congelar el programa, y cómo con solo cambiar el orden en que se piden los recursos (rompiendo la espera circular), el problema desaparece por completo sin complicarse la vida. Esta práctica de verdad te aterriza todo lo que uno lee en las diapositivas sobre cómo el sistema operativo administra la CPU y sincroniza las tareas.

Preguntas de Reflexión

1. **¿**Por **qué el resultado de condicion_carrera.py cambiaba entre ejecuciones antes de activar la protección, y dejó de cambiar después??**

Debido a que el código no tenía establecido un orden de ejecución claro para las tareas que ejecutaba cada hilo, ambos leían y modificaban la variable casi al mismo tiempo. Esto provocaba que cada hilo tomara el valor inicial, pero, al momento en que el primer hilo realizaba la modificación, el segundo —al haber leído previamente el valor original— no tomaba el valor actualizado por el primero, incrementando el contador de manera independiente. En cambio, al activar la protección, se establecía un orden de ejecución que evitaba este comportamiento y prevenía el uso de datos desactualizados.

2. **¿Qué relación existe entre el estado de un proceso mostrado por ps (running, sleeping, zombie) y los estados teóricos vistos en clase (nuevo, listo, ejecutando, bloqueado, terminado)?**

La relación es directa, solo que Linux en la práctica agrupa un par de estados que en la teoría vemos por separado:

• Running (R) en ps: En clase nos enseñan dos estados distintos: Listo (cuando el proceso está en la cola esperando que le toque la CPU) y Ejecutando (cuando la CPU ya lo está corriendo). En Linux, ps mete a ambos dentro de la letra R (de runnable), porque para el planificador un proceso que ya está listo para correr o que está corriendo en ese microsegundo se consideran prácticamente activos.

• Sleeping (S o D) en ps: Corresponde exactamente al estado teórico de Bloqueado o En espera. El proceso no está gastando CPU porque está esperando que pase algo externo: que el usuario presione una tecla, que termine un sleep, que llegue un paquete de red o que el disco termine de leer un archivo.

• Zombie (Z) en ps: Es la etapa final del estado Terminado. El proceso ya acabó todo su trabajo y liberó su memoria, pero sigue apareciendo como un 'fantasma' en la tabla de procesos porque su proceso padre todavía no ha leído cómo terminó (mediante la llamada wait()). Una vez el padre se entera, el zombie desaparece por completo.

• El estado Nuevo de la teoría vendría siendo ese instante exacto cuando el sistema hace el fork() para crear el proceso y le asigna su PID y sus estructuras, justo antes de meterlo a la cola de listos (R).

3. **En el script del productor-consumidor, ¿qué pasaría si el buffer tuviera tamaño 0? Relacionen su respuesta con el concepto de región crítica.**

Si el buffer tuviera tamaño 0, significaría que no hay ningún cajón ni espacio en memoria para guardar nada de forma intermedia. Básicamente se pierde toda la ventaja de que el productor y el consumidor trabajen a su propio ritmo.

Esto obligaría a que ambos hilos tengan que hacer lo que en sistemas operativos se conoce como una

'cita previa' o rendezvous: el productor no puede soltar el dato si el consumidor no está ahí pegado con la mano abierta para recibirlo al mismo instante.

Respecto a la región crítica, el cambio es total: ya no funciona como un depósito donde el productor entra, deja el paquete y se va tranquilo a fabricar otro; ahora la región crítica se vuelve un punto de transferencia mano a mano estricto. Si el consumidor se demora, el productor se queda totalmente congelado esperando; y si el productor se retrasa, el consumidor no tiene nada que hacer. Todo el sistema se frena y queda amarrado a la velocidad del hilo más lento.

4. **De las cuatro condiciones de Havender, ¿cuál fue rota por el cambio a modo "PREVENCION" en el script de deadlock? ¿Por qué alcanzó con romper solo esa?**

La condición que se rompió fue la de Espera Circular.

En el modo DEADLOCK teníamos un círculo vicioso clásico: el Hilo A agarraba la impresora y se quedaba esperando el escáner, mientras que el Hilo B agarraba el escáner y se quedaba esperando la impresora. Ninguno soltaba lo suyo y los dos se quedaban esperando lo que tenía el otro, armando un ciclo cerrado donde nadie avanzaba.

Al cambiar a modo PREVENCION, lo que se hizo fue imponer una regla de juego fija (orden jerárquico): obligar a que ambos hilos pidan los recursos exactamente en el mismo orden (primero la impresora y después el escáner). Con esa sola regla, si el Hilo A agarra primero la impresora, el Hilo B se frena de una al intentar pedir la impresora, y no puede adueñarse del escáner a traición. Así es imposible que se forme un ciclo.

¿Por qué bastó con romper solo esa? Porque Havender demostró que para que haya un interbloqueo se tienen que cumplir las cuatro condiciones al mismo tiempo, como si fueran los cuatro candados de una puerta. Con que revientes uno solo de esos candados (en este caso, la espera circular), ya es matemáticamente imposible que el sistema se quede bloqueado.

5. **¿Qué dificultades se presentaron durante la práctica y qué solución encontraron?**

Durante la práctica se nos presentaron principalmente tres problemas y así los solucionamos:

1. Filtrar bien los procesos en segundo plano: Al mandar los procesos de prueba a correr en el fondo con el ampersand (&), no era tan fácil ver si seguían vivos o si ya habían terminado. Nos tocó apoyarnos bastante en ps aux | grep sleep y jobs para cazar el PID exacto antes de meterle el kill -15 o el kill -9 y verificar que de verdad murieran.

2. El lío de los permisos con renice: Cuando intentamos bajarle el valor de nice de 15 a 5 para darle más prioridad a un proceso, la terminal nos rebotó diciendo que no teníamos permisos. Ahí recordamos que un usuario normal solo puede bajarse la prioridad a sí mismo (aumentar el número nice), pero para subírsela (bajar el número) la única forma es tener permisos de root. Lo solucionamos metiéndole sudo al comando (sudo renice -n 5 -p <PID>) y revisamos en htop que el cambio se aplicara de verdad.

3. Los espacios y la indentación en nano: Al editar los scripts de Python directo en la terminal con nano para descomentar las líneas del Lock y cambiar las variables de TAMANO_BUFFER y MODO, tuvimos que tener mucho cuidado con no meter espacios de más ni tabulaciones chuecas, porque Python es súper delicado con la indentación y si dejas una línea mal alineada te tira un IndentationError de una.

Memoria, Archivos, Dispositivos y Administración del Servidor

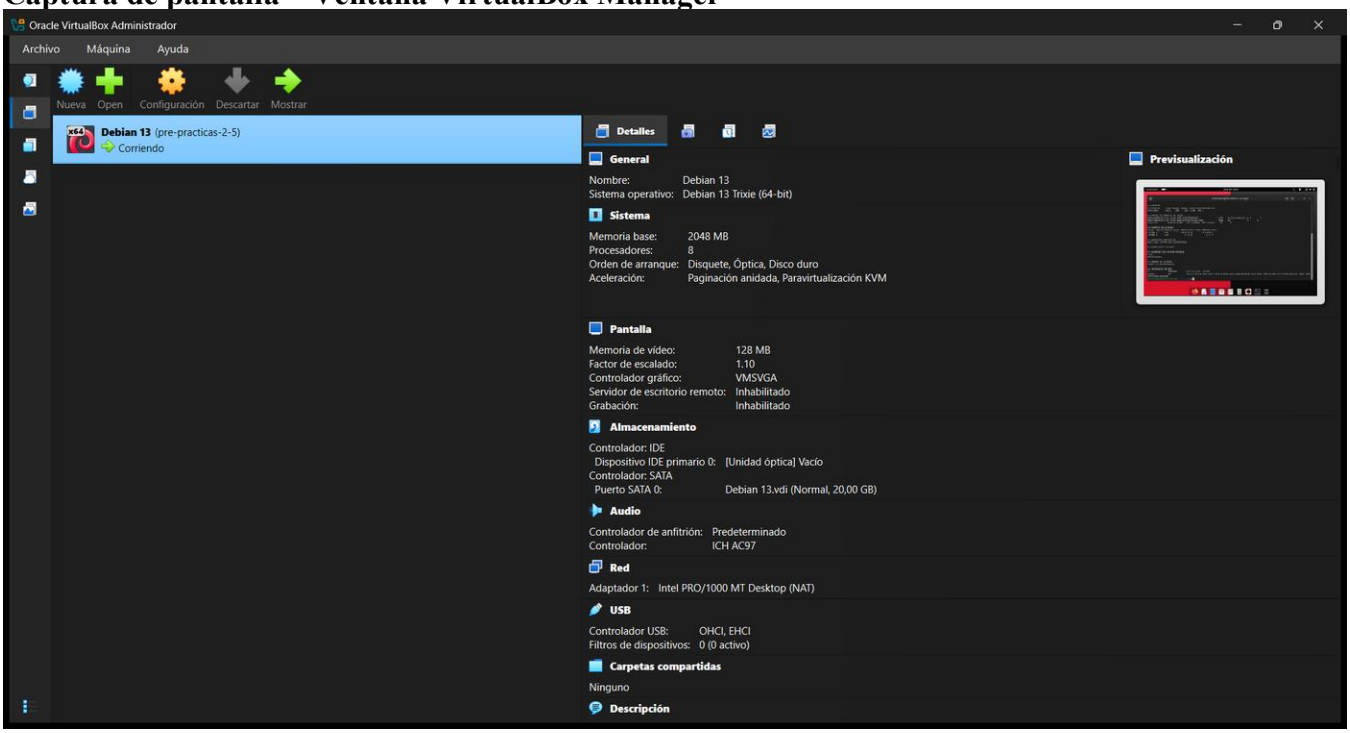
Esta entrega convierte la máquina virtual de las Entregas 1 y 2 en un servidor Debian 13 operable y documentado, y añade una segunda máquina (cliente) para administrarlo a distancia. Trabajamos sobre la máquina virtual servidor (hostname servidor, 2048 MB de RAM) y un clon llamado cliente (1024 MB), unidas por la red interna redso con las direcciones 192.168.100.10 y 192.168.100.11. El método de la guía es siempre el mismo: predecir antes de ejecutar, contrastar con la salida real y explicar la diferencia. A lo largo de la sección medimos la memoria y el swap, comparamos los algoritmos de reemplazo de páginas, creamos un sistema de archivos ext4 montado por UUID, explicamos inodos, enlaces y ACL, y dejamos el servidor protegido con ufw, SSH sin root y respaldos automáticos. Cada figura es una captura real de nuestra máquina virtual y cada cifra citada aparece en la captura correspondiente.

Parte 0 — Preparación de las dos máquinas

0.1 Fotografía inicial del servidor

Antes de tocar nada tomamos una fotografía del estado del servidor con el script «linea_base.sh», que será el punto de comparación de la Parte G. La Figura 86 muestra la configuración de la máquina virtual servidor en VirtualBox: 2048 MB de memoria base, un disco SATA de 20 GB y el Adaptador 1 en NAT, que es el que da salida a internet.

Figura 86
Captura de pantalla – Ventana VirtualBox Manager

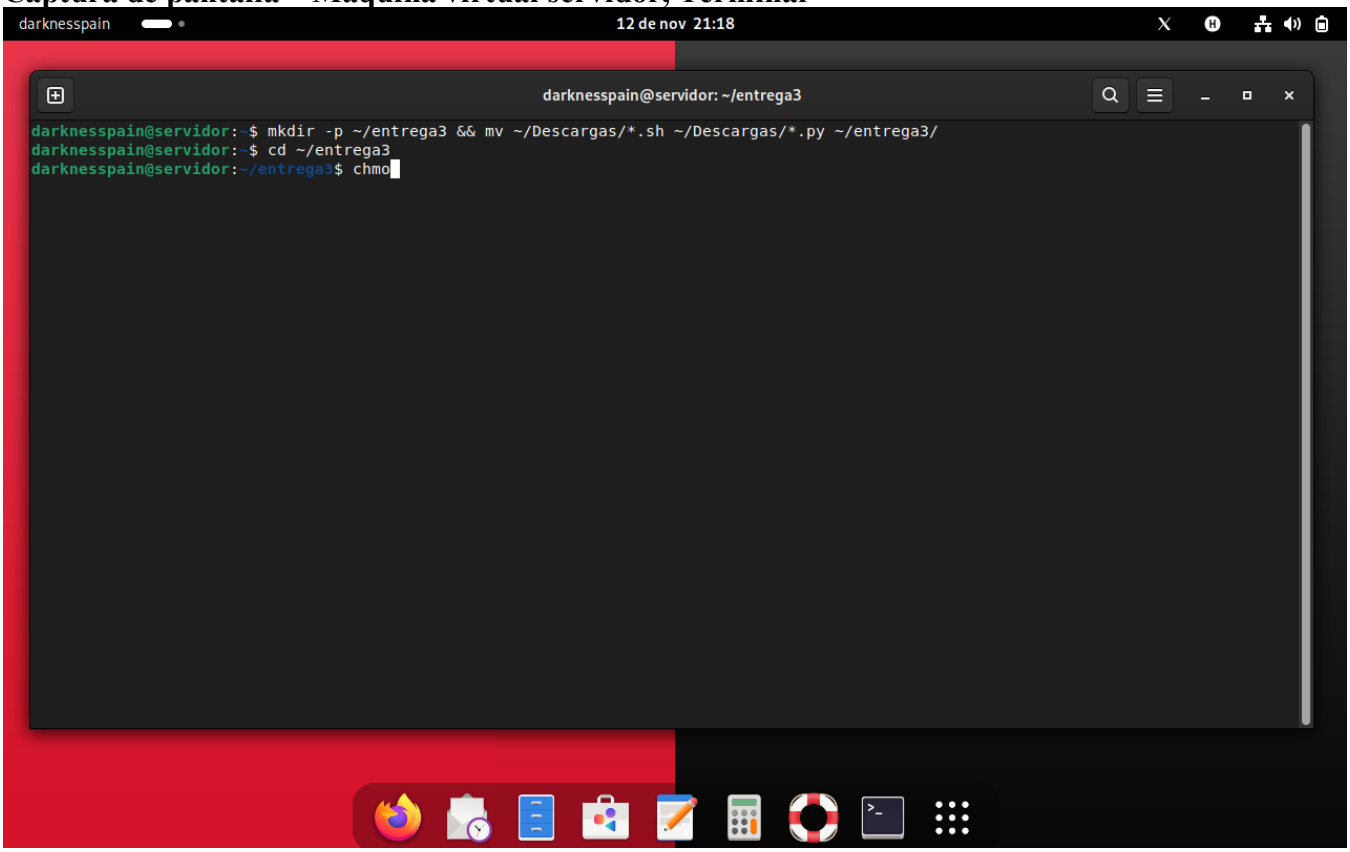


Nota. En la lista de máquinas virtuales aparece «Debian 13» y, en el panel de detalles, la memoria base de 2048 MB y el disco SATA de 20 GB. La máquina estaba apagada al tomar la captura.

La captura fija el punto de partida: la VM que veníamos usando desde la Entrega 1 y que ahora será el servidor. Los 2048 MB de RAM son la restricción física de todo el laboratorio: cualquier experimento de memoria tiene que caber en 1972 MB reales, porque el kernel reserva parte de esa cifra para sí mismo. Ese detalle explica por qué más adelante «free -m» nunca muestra 2048 MB como memoria total y por qué el swap deja de ser opcional cuando reservamos más de lo disponible.

Con la máquina encendida movimos los seis archivos entregados por la docente desde ~/Descargas a una carpeta de trabajo y les dimos permiso de ejecución.

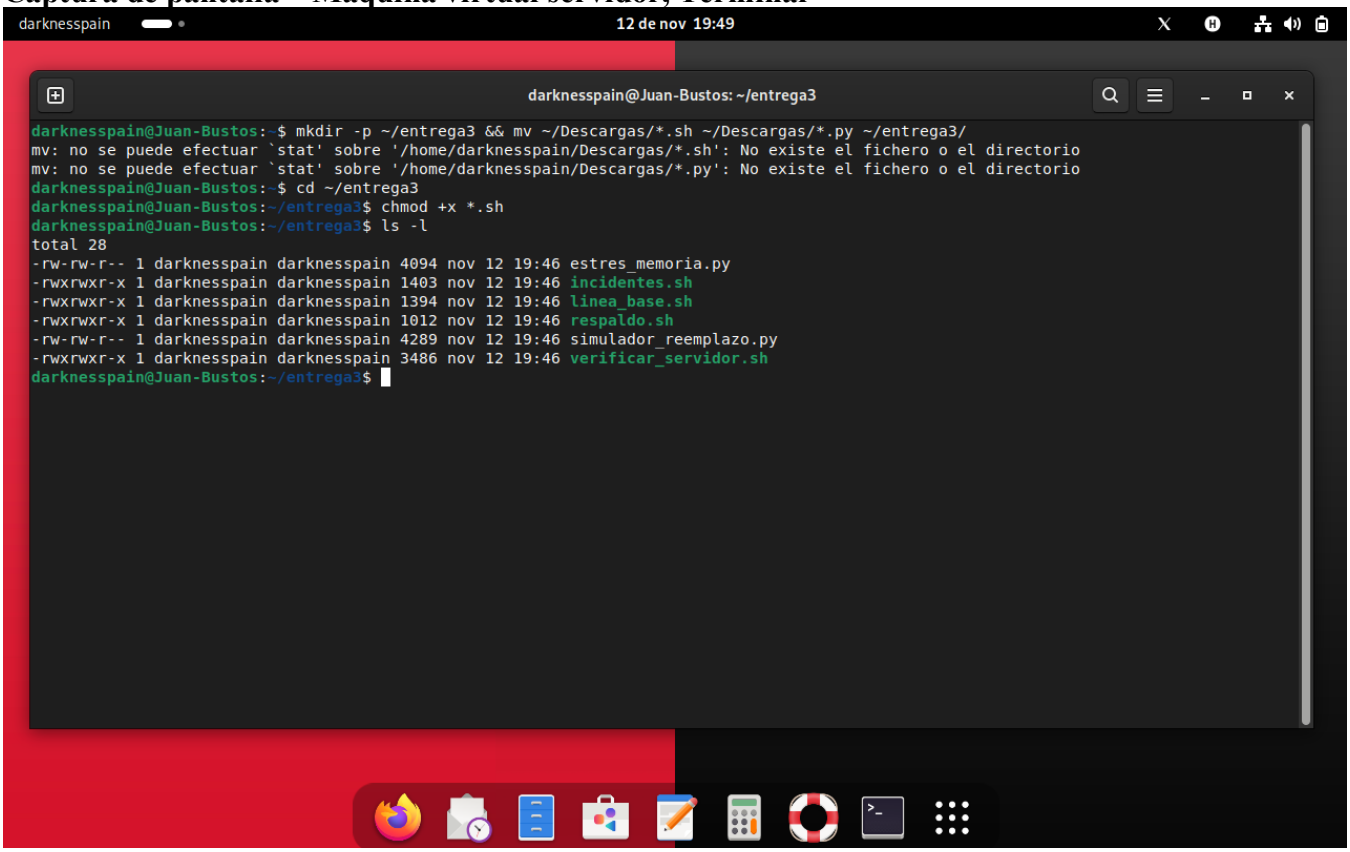
Figura 87
Captura de pantalla – Máquina virtual servidor, Terminal



Nota. Se ejecutaron las tres líneas de preparación de la guía: «mkdir -p ~/entrega3 && mv ~/Descargas/*.sh ~/Descargas/*.py ~/entrega3/», «cd ~/entrega3» y «chmod +x *.sh». No aparece ningún error, así que el traslado y el permiso se aplicaron.

El primer comando crea la carpeta de trabajo y mueve allí los scripts y los dos programas en Python; el segundo entra en esa carpeta y el tercero da permiso de ejecución a los cuatro scripts de shell. Sin el «chmod +x» el sistema se negaría a ejecutar «./linea_base.sh» con un «Permiso denegado», porque en Linux un archivo solo se ejecuta si tiene el bit x para el usuario que lo invoca. Es un ejemplo mínimo de que los permisos son parte del modelo de seguridad del sistema de archivos y no un detalle cosmético.

Figura 88
Captura de pantalla – Máquina virtual servidor, Terminal

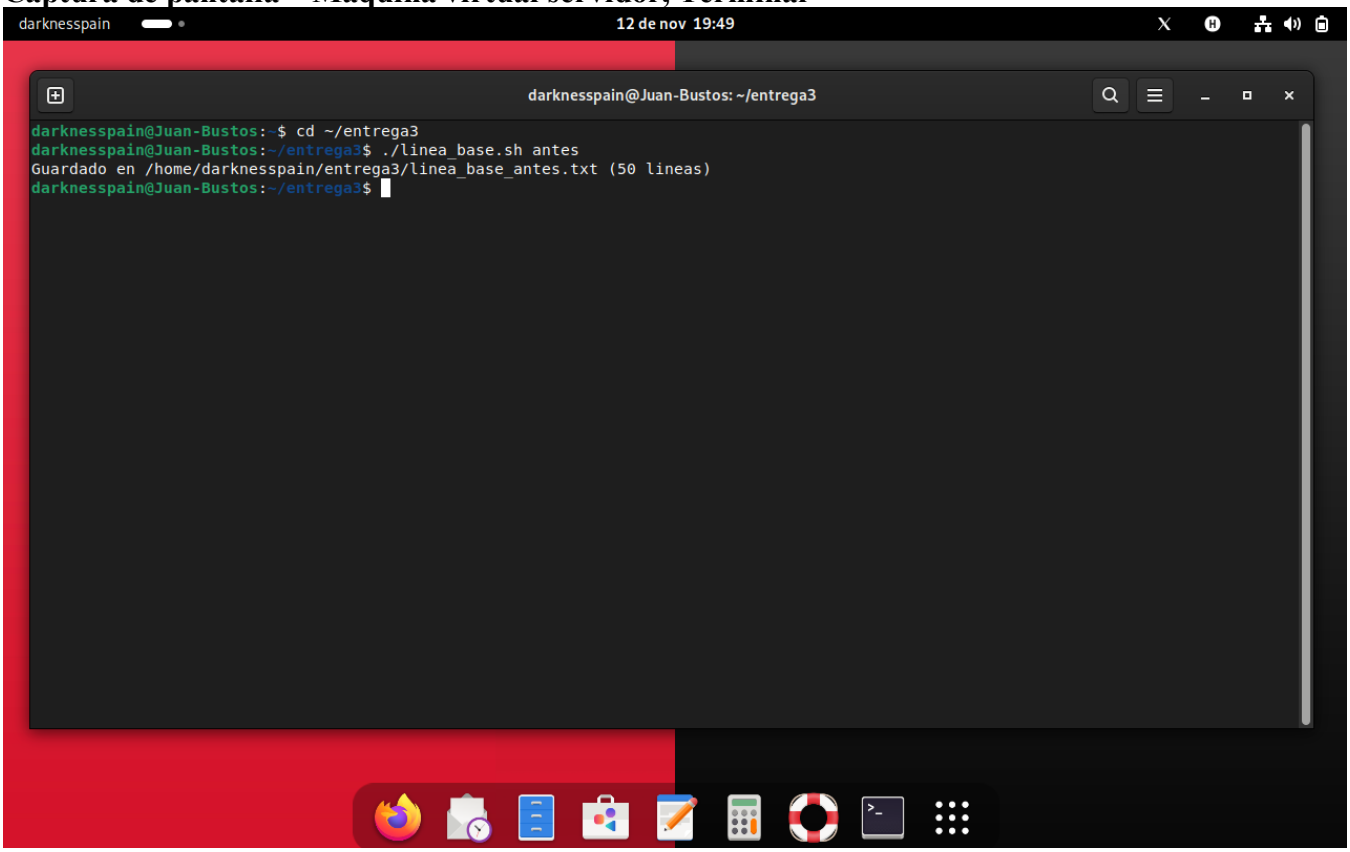


Nota. «ls -l» muestra los seis archivos en ~/entrega3; los cuatro scripts aparecen con «x» en sus permisos y los dos programas de Python con permiso de lectura y escritura.

La verificación con «ls -l» confirma dos cosas: que están los seis archivos (los cuatro scripts y los dos programas) y que los scripts quedaron ejecutables. La primera columna resume el modo: «-rwxrwxr-x» significa archivo regular, lectura/escritura/ejecución para dueño y grupo, y lectura/ejecución para el resto. Esta comprobación es la que evita descubrir el problema tres pasos más adelante, cuando el script falla por permisos.

Tomamos entonces la fotografía inicial con el script de la docente.

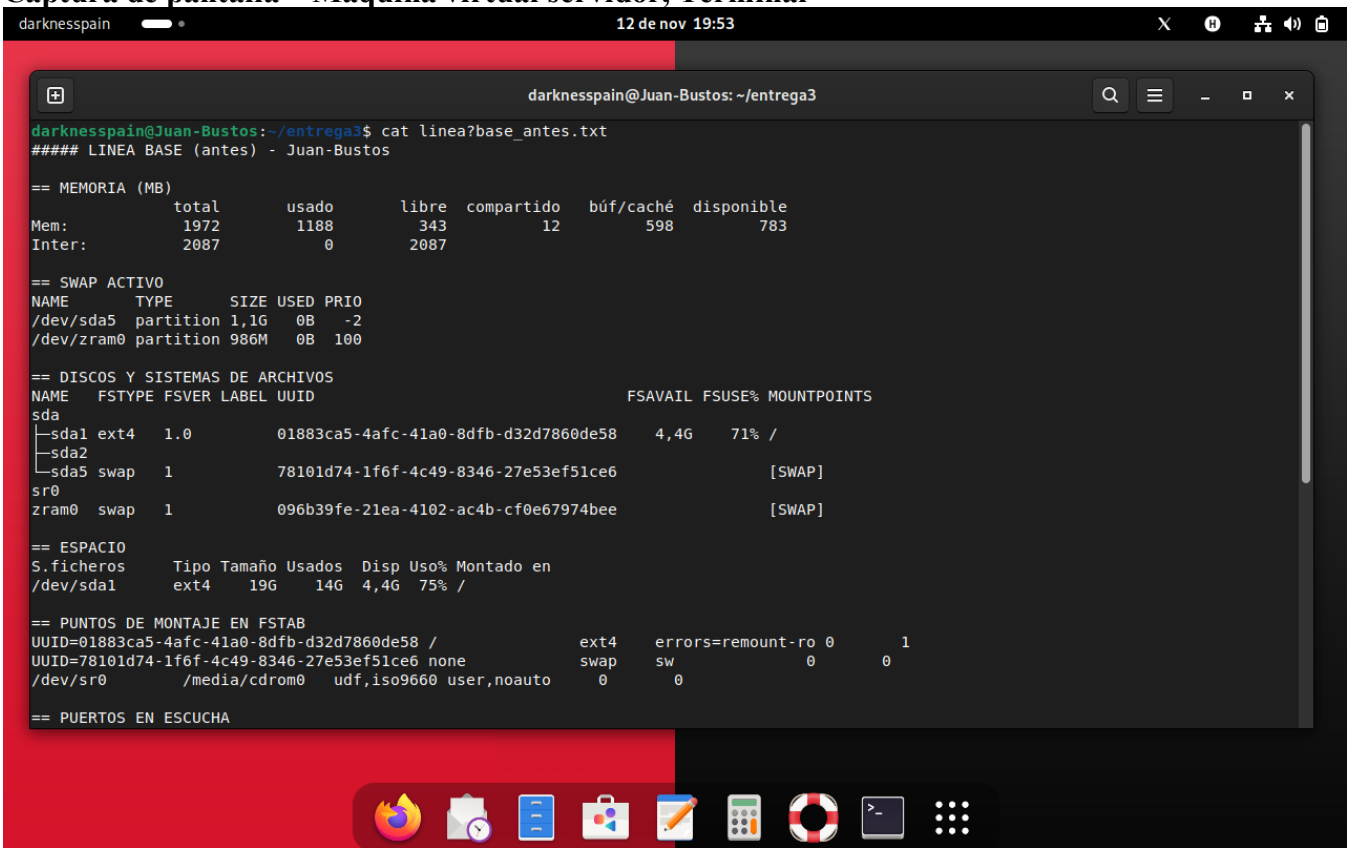
Figura 89
Captura de pantalla – Máquina virtual servidor, Terminal



Nota. «./linea_base.sh antes» guarda el archivo linea_base_antes.txt con 50 líneas y lo informa en pantalla. El prompt muestra el nombre original de la máquina, Juan-Bustos, porque el cambio de hostname se hace después.

El script recorre los subsistemas que vamos a modificar: memoria, swap, discos, espacio, fstab, puertos en escucha, servicios fallidos, usuarios, grupos e interfaces. Guardar ese estado antes de empezar es lo que permite demostrar al final qué cambió; sin línea base, la comparación de la Parte G sería una impresión y no una prueba. El archivo tiene 50 líneas porque varias secciones (por ejemplo la de servicios con fallas) salen vacías.

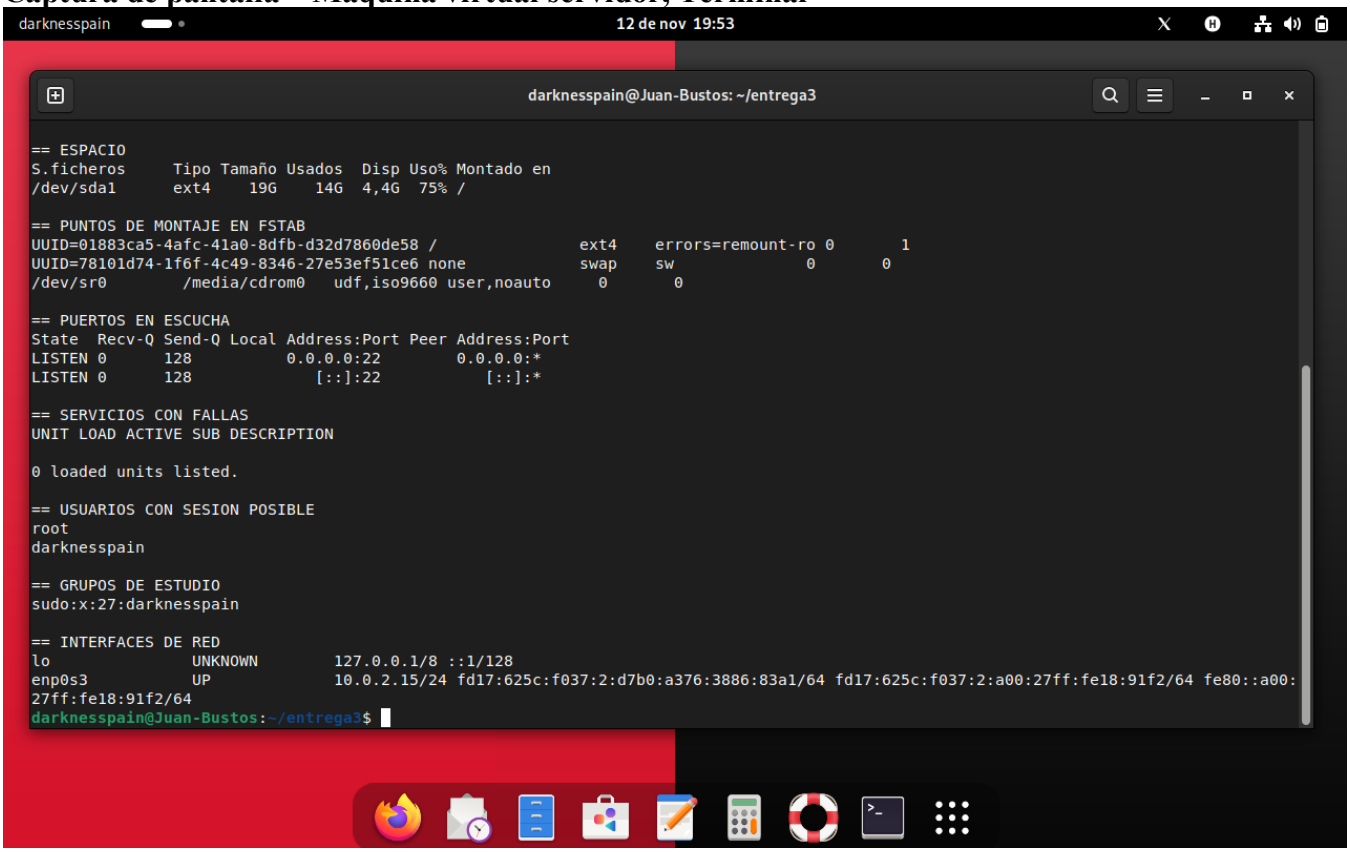
Figura 90
Captura de pantalla – Máquina virtual servidor, Terminal



Nota. Primera parte del contenido de linea_base_antes.txt: «free -m» con 1972 MB totales y 783 MB disponibles; «swapon --show» con la partición /dev/sda5 de 1,1 GB (PRIO -2) y zram0 de 986 MB (PRIO 100); y «lsblk -f» con la raíz ext4 en /dev/sda1.

La fotografía inicial confirma que el sistema ya traía dos áreas de intercambio antes de esta práctica: una partición de swap creada por el instalador y un dispositivo zram, que es swap comprimido en RAM. Los tamaños importan: 1,1 GB y 986 MB suman los 2087 MB de «Inter» que muestra «free -m». También se ve que la raíz ocupa 19 GB con un 71 % usado, un dato que explica por qué más adelante hay que vigilar el espacio al crear el archivo de swap.

Figura 91
Captura de pantalla – Máquina virtual servidor, Terminal



Nota. Segunda parte del archivo: espacio por sistema de archivos, líneas activas de /etc/fstab, puertos en escucha (solo el 22), cero servicios fallidos, los usuarios root y darknesspain y la interfaz enp0s3 con 10.0.2.15 por NAT.

Aquí se ve el estado «de fábrica» del servidor: únicamente SSH escuchando, ninguna unidad en estado failed y una sola interfaz con dirección. Cualquier puerto adicional que aparezca en la comparación final tendrá que explicarse por un servicio que instalamos nosotros. La línea de fstab con «errors=remount-ro» en la raíz es la política por defecto de Debian: si el sistema de archivos raíz detecta errores, se remonta en solo lectura para evitar daños mayores.

0.2 Crear la VM cliente y 0.3 segunda tarjeta de red

Apagamos el servidor desde el menú del sistema, tal como pide la guía, y tomamos la captura de la VM apagada antes de clonarla. El clon se creó completo, con reinicialización de las direcciones MAC, y después se le bajó la memoria a 1024 MB. En las dos máquinas se agregó un segundo adaptador en red interna llamado redso.

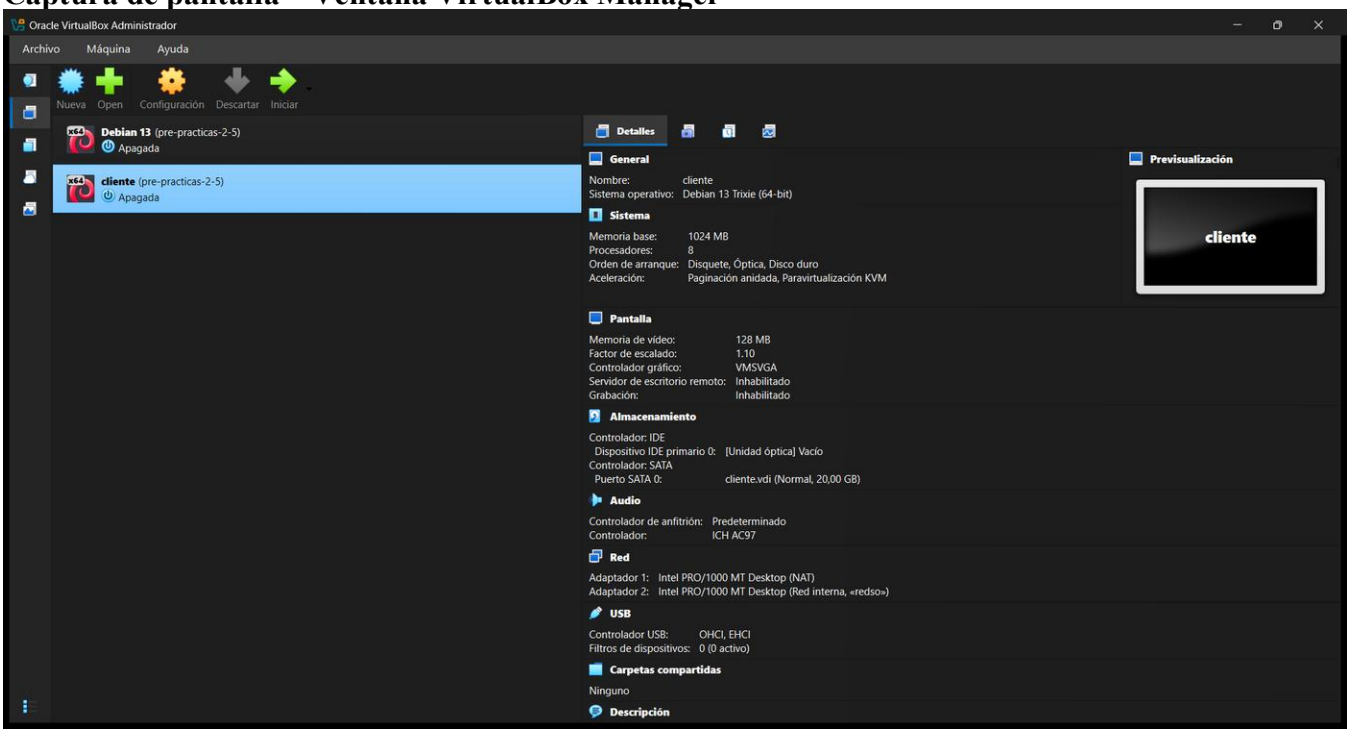
Figura 92
Captura de pantalla – Máquina virtual servidor, Menú de GNOME



Nota. El menú de encendido de GNOME está abierto sobre la terminal: Apagar, Suspender, Reiniciar, Apagar… y Cerrar la sesión. Es el camino que indica la guía para dejar la VM en estado «Apagada» antes de clonarla.

Apagar por el menú —y no por el botón de la ventana de VirtualBox— garantiza un cierre ordenado: systemd detiene los servicios y vacía las cachés de disco. Si se apagara de golpe, el sistema de archivos podría quedar marcado como «no limpio» y el arranque siguiente forzaría una revisión con fsck. Ese cuidado es el mismo que aplica un administrador antes de cualquier operación sobre discos virtuales.

Figura 93
Captura de pantalla – Ventana VirtualBox Manager



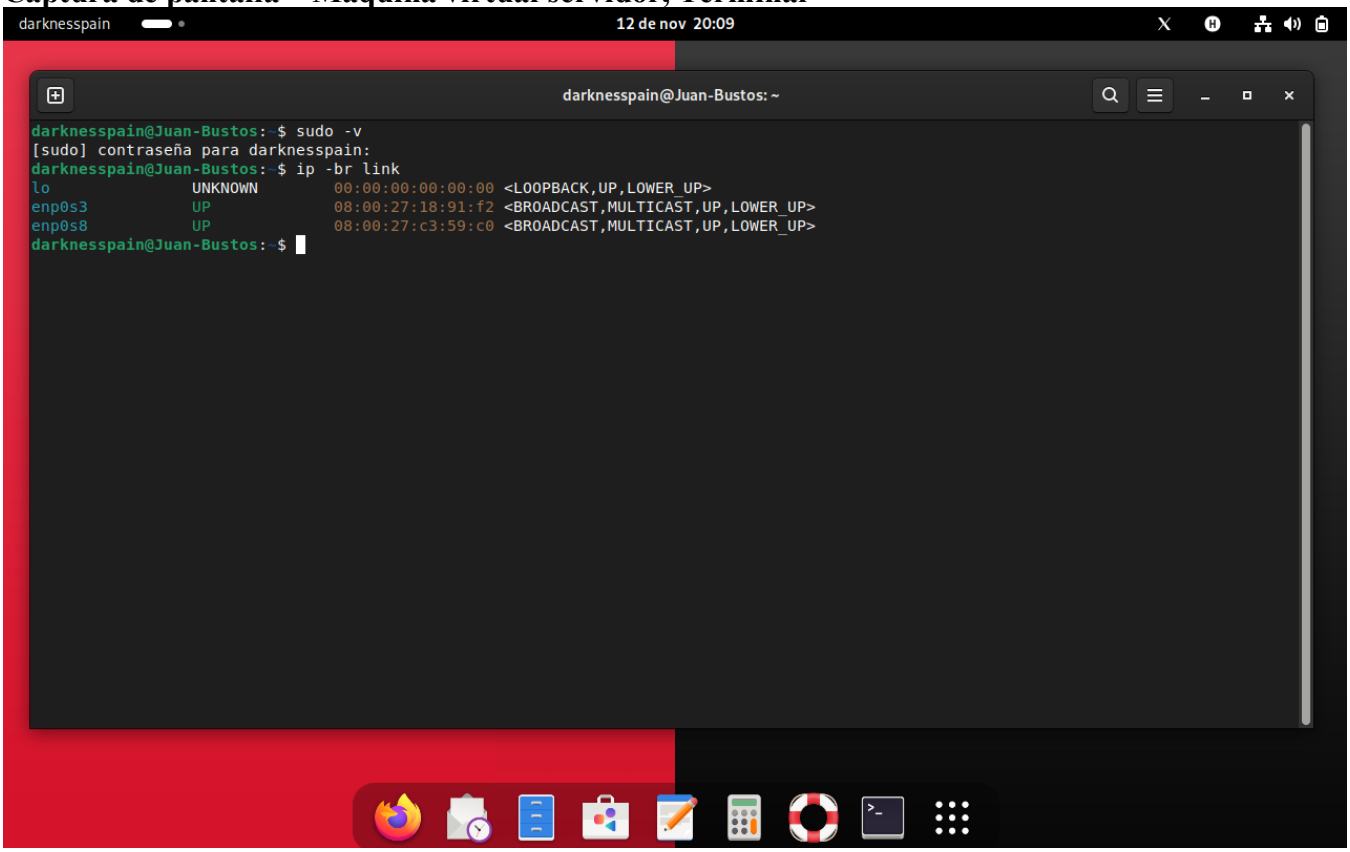
Nota. La lista muestra las dos máquinas: «Debian 13» y «cliente», ambas apagadas. El panel de detalles de cliente indica 1024 MB de memoria base, el disco cliente.vdi de 20 GB, el Adaptador 1 en NAT y el Adaptador 2 en «Red interna, redso».

La captura demuestra las tres decisiones de diseño del montaje: el cliente es un clon completo del servidor (mismo sistema, mismos usuarios), tiene la mitad de memoria para que las dos VM quepan en el equipo anfitrión y comparte con el servidor una red interna aislada. La red interna es la pieza clave del sistema distribuido: no depende de internet ni del NAT, así que el ping y el SSH entre las dos máquinas siguen funcionando aunque el host se quede sin red.

0.4 Nombre y dirección de cada máquina

En el servidor identificamos las interfaces, fijamos el hostname, corregimos /etc/hosts y creamos el perfil de red de la segunda tarjeta con dirección estática 192.168.100.10.

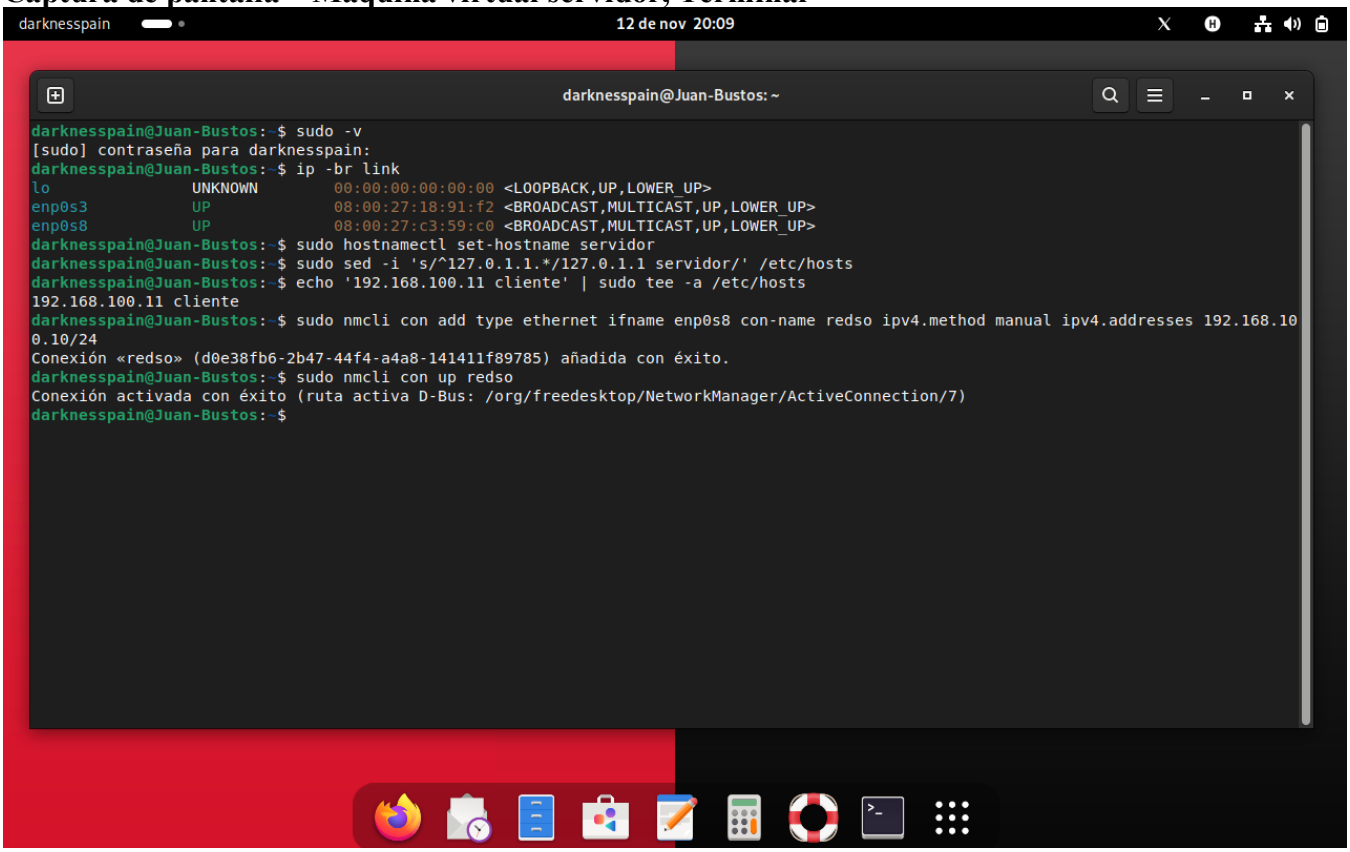
Figura 94
Captura de pantalla – Máquina virtual servidor, Terminal



Nota. «ip -br link» muestra tres interfaces: lo, enp0s3 y enp0s8. La segunda tarjeta ya existe con estado DOWN porque todavía no tiene configuración.

El comando sirve para averiguar el nombre real de la interfaz antes de escribirla en los comandos siguientes: en Debian los nombres no son eth0 ni eth1 sino enp0s3 y enp0s8, derivados de la posición del bus PCI. Si nos equivocáramos de nombre, «nmcli» crearía un perfil que nunca se activaría y el ping entre las dos máquinas fallaría sin un error evidente. Verificar antes de configurar es más barato que diagnosticar después.

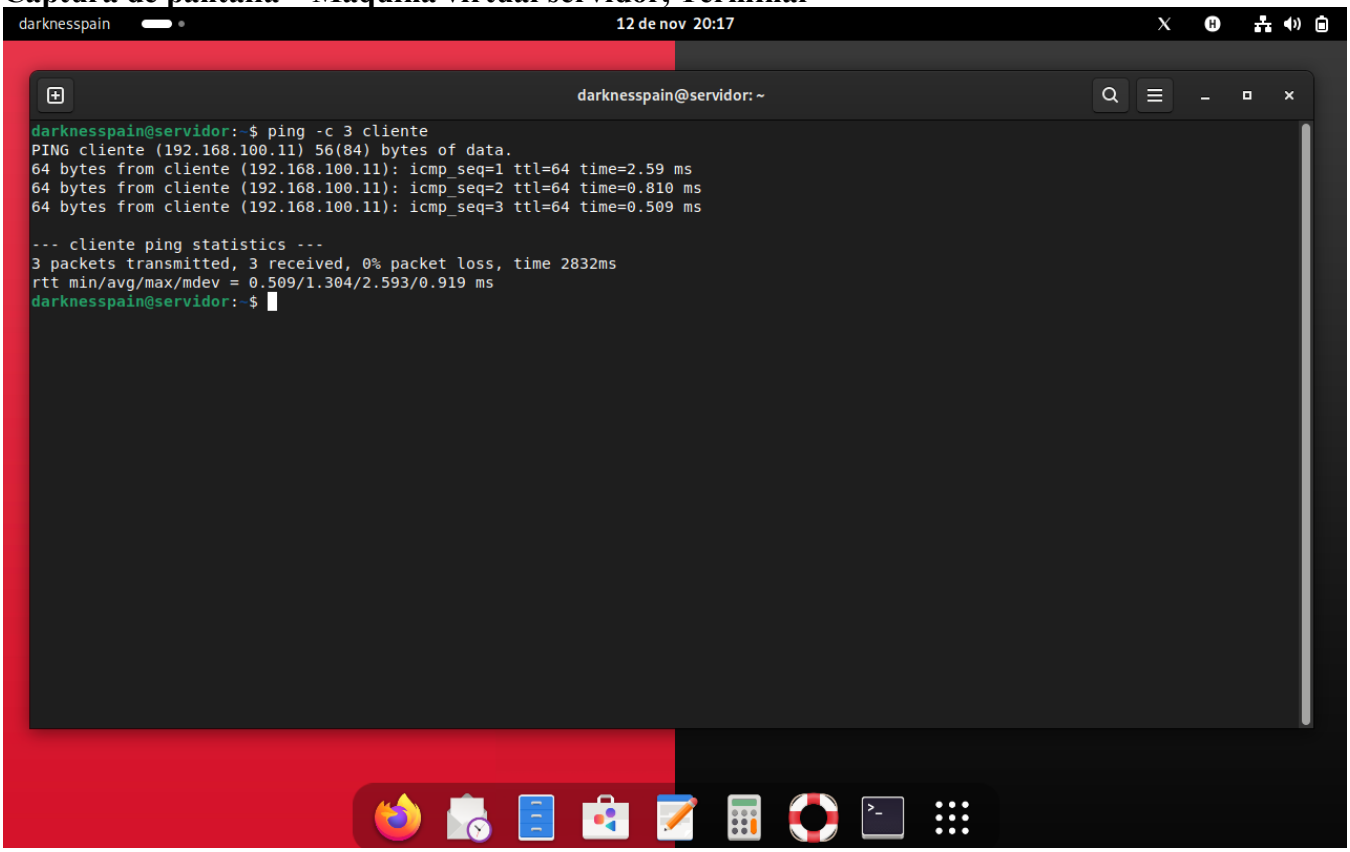
Figura 95
Captura de pantalla – Máquina virtual servidor, Terminal



Nota. Se ejecutaron «hostnamectl set-hostname servidor», el «sed» sobre /etc/hosts, la línea de hosts para cliente con «tee», el «nmcli con add» del perfil redso con 192.168.100.10/24 y «nmcli con up redso». La última orden responde que la conexión se activó correctamente.

Estos cinco comandos hacen tres cosas distintas: cambian la identidad de la máquina, resuelven el nombre del vecino sin depender de un DNS y le dan una dirección fija a la tarjeta de la red interna. La dirección se escribe con máscara /24, es decir, las dos máquinas están en la misma red 192.168.100.0/24 y se alcanzan directamente. Esa es la condición para que el ping funcione sin router: el tráfico va por el segmento interno que VirtualBox conecta en memoria.

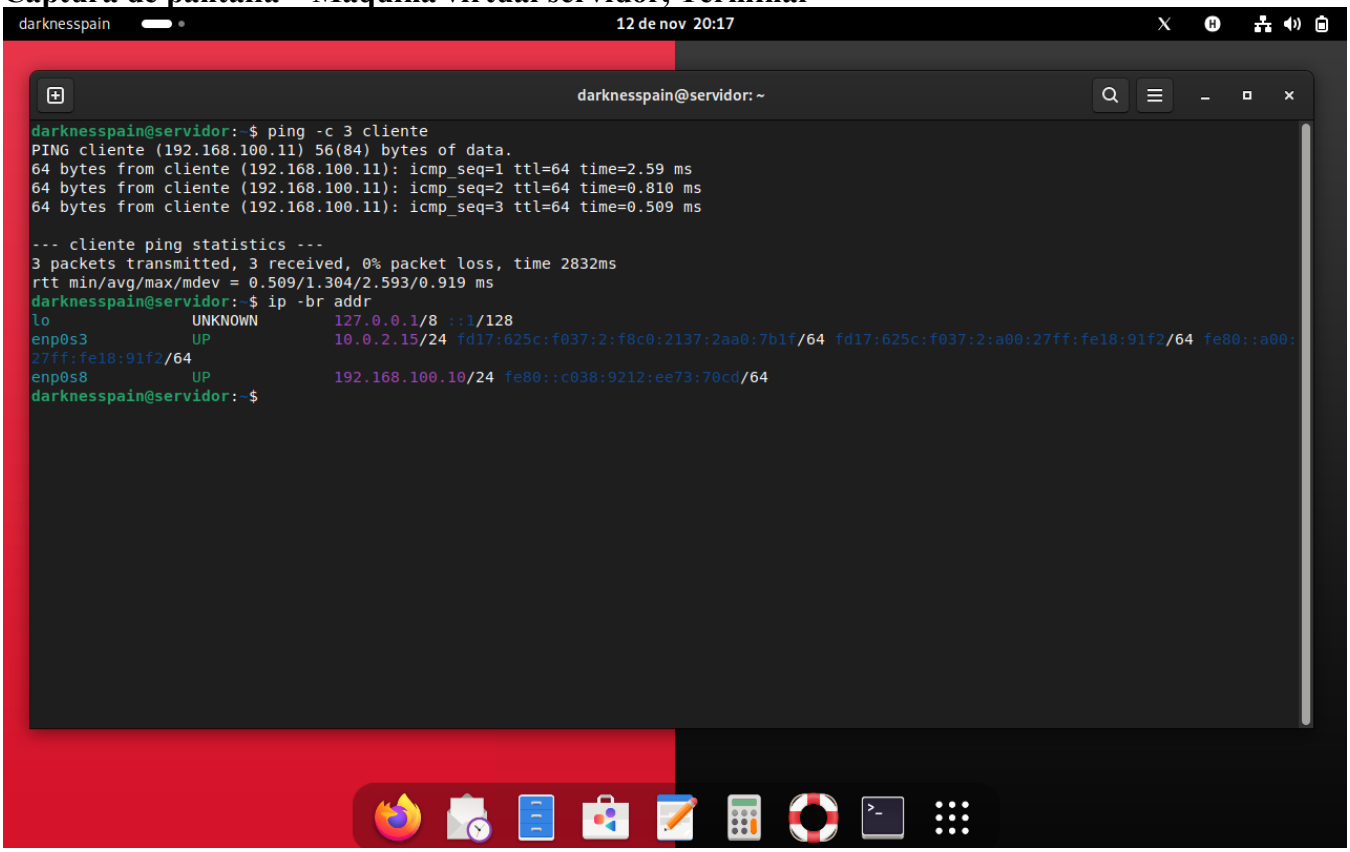
Figura 96
Captura de pantalla – Máquina virtual servidor, Terminal



Nota. «ping -c 3 cliente» resuelve el nombre a 192.168.100.11 y recibe las tres respuestas, con tiempos de 2,59 ms, 0,810 ms y 0,509 ms y cero por ciento de pérdida.

El ping confirma de una sola vez tres cosas: que el nombre «cliente» se resuelve por /etc/hosts, que la red interna está conectada en las dos VM y que la pila IP funciona. Los tiempos son de fracciones de milisegundo porque el paquete nunca sale del equipo anfitrión: VirtualBox lo entrega directamente a la otra máquina virtual. Esta prueba es la que separa un problema de red de un problema de servicio cuando más adelante el SSH deje de responder.

Figura 97
Captura de pantalla – Máquina virtual servidor, Terminal



Nota. «ip -br addr» muestra enp0s3 con 10.0.2.15 (NAT) y enp0s8 con 192.168.100.10/24, además de las direcciones IPv6 de enlace local.

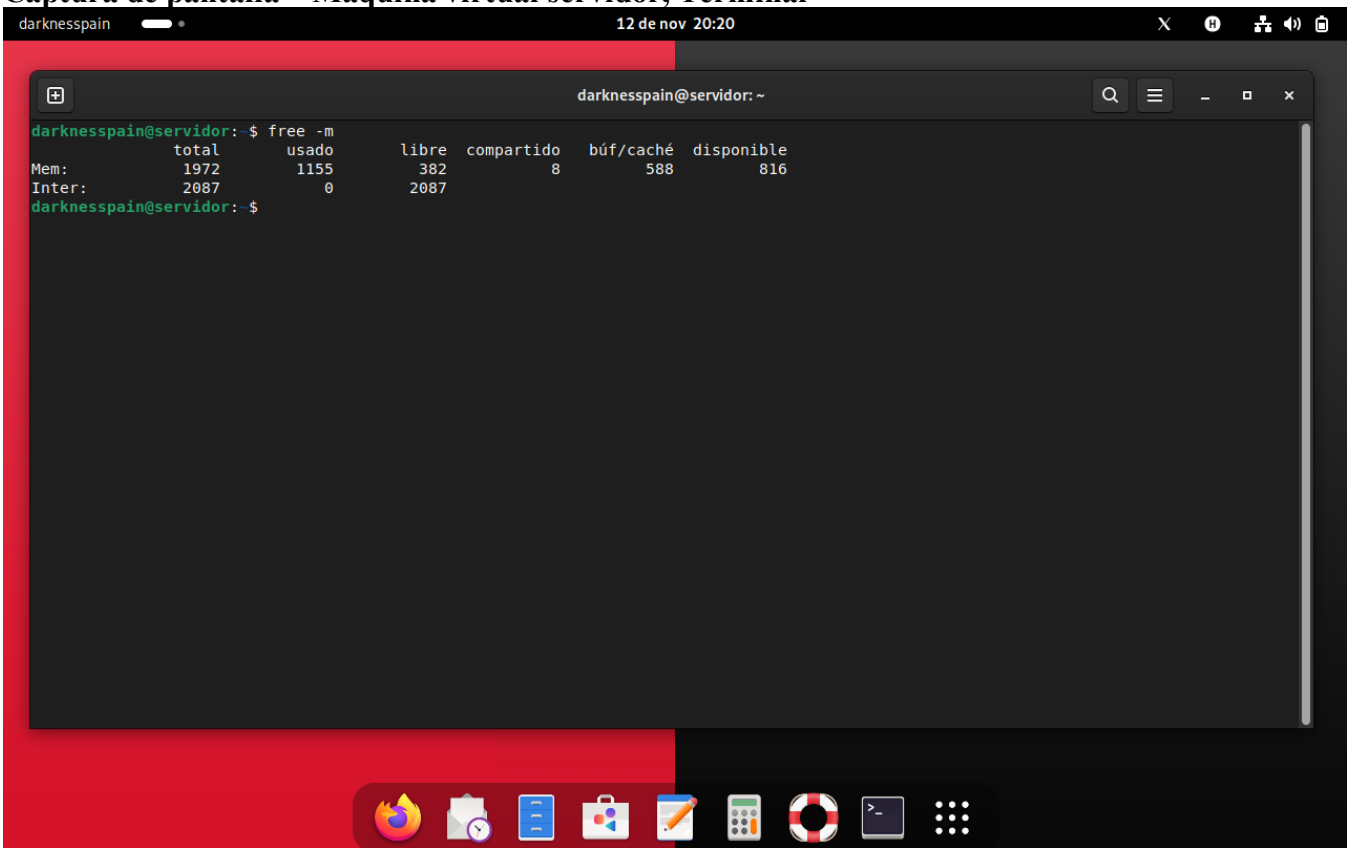
La captura verifica el resultado de la configuración: cada tarjeta tiene su dirección y cumplen funciones distintas. enp0s3 da salida a internet a través del NAT de VirtualBox y es la que permite instalar paquetes; enp0s8 es la red privada del laboratorio y por ella viajarán el ping y los respaldos. Separar el tráfico de administración del tráfico de internet es una práctica habitual en servidores reales, donde la interfaz de gestión suele estar en una VLAN aparte.

Parte A — Gestión de la memoria

A.1 Mirar la memoria como la ve el sistema

Empezamos observando la memoria con cuatro vistas distintas: el resumen de «free», los contadores crudos de /proc/meminfo, las áreas de intercambio activas y los procesos que más memoria residente consumen.

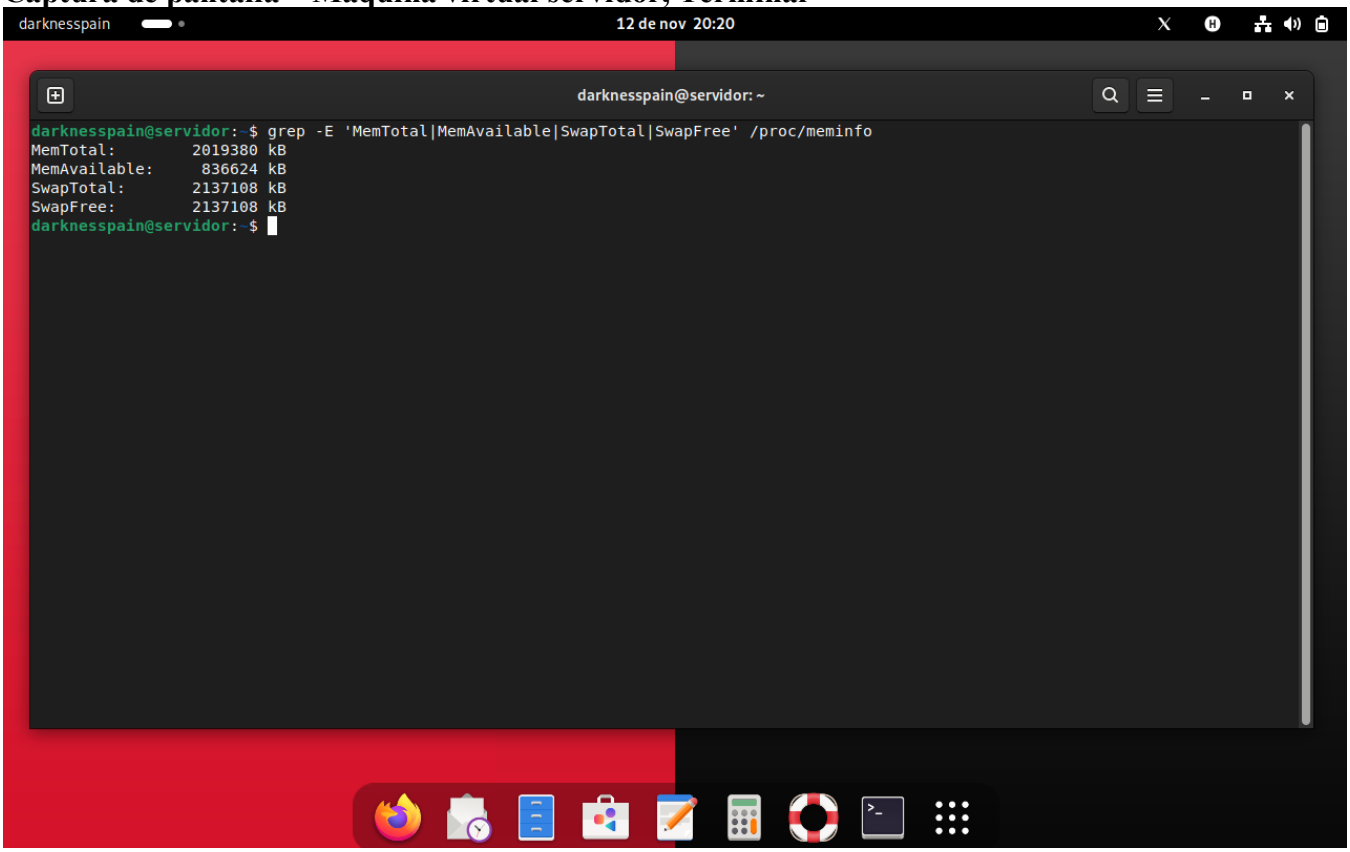
Figura 98
Captura de pantalla – Máquina virtual servidor, Terminal



Nota. «free -m» informa 1972 MB de RAM total, 1142 MB en uso, 395 MB libres y 829 MB disponibles; la línea «Inter» muestra 3111 MB de swap total y ninguno usado.

La columna que importa no es «libre» sino «disponible»: Linux usa la RAM sobrante como caché de disco y la devuelve cuando un proceso la necesita. Por eso hay 395 MB libres pero 829 MB disponibles. La diferencia entre ambas cifras —unos 590 MB— es exactamente la caché que el kernel puede descartar sin escribir nada al disco. Esa distinción es la que permite predecir si un programa que pide 400 MB cabrá sin tocar el swap.

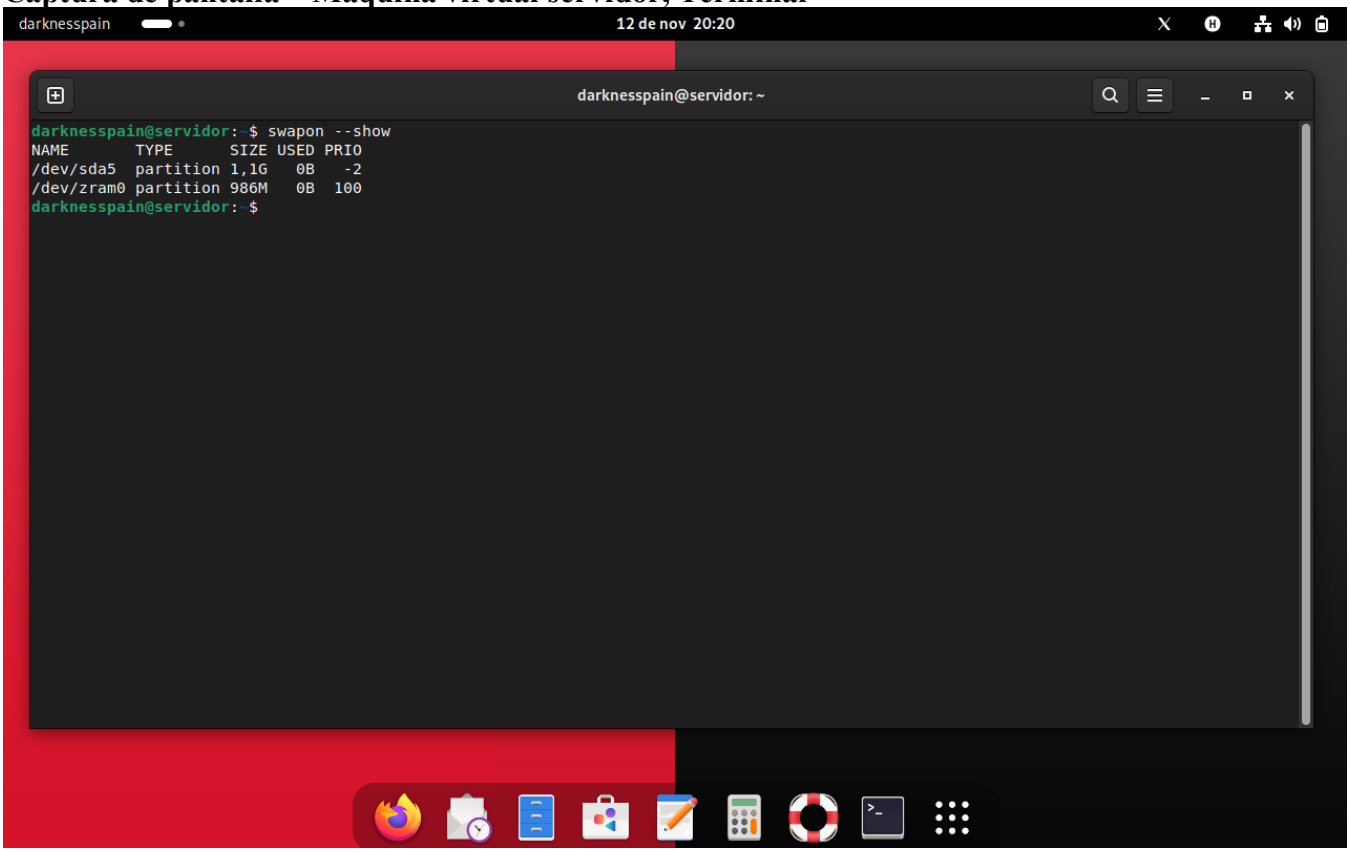
Figura 99
Captura de pantalla – Máquina virtual servidor, Terminal



Nota. «grep -E 'MemTotal|MemAvailable|SwapTotal|SwapFree' /proc/meminfo» devuelve MemTotal 2019380 kB, MemAvailable 845092 kB, SwapTotal 2137108 kB y SwapFree 2137108 kB.

/proc/meminfo es la fuente de la que «free» toma sus números; aquí se ven en kilobytes y sin redondeos. MemTotal (2019380 kB ≈ 1972 MB) es menor que los 2048 MB asignados a la VM porque el kernel reserva memoria para sus propias estructuras. MemAvailable (845092 kB ≈ 825 MB) es la estimación de cuánta memoria puede asignarse sin provocar swapping, y es el valor que usamos como D en el experimento A.3.

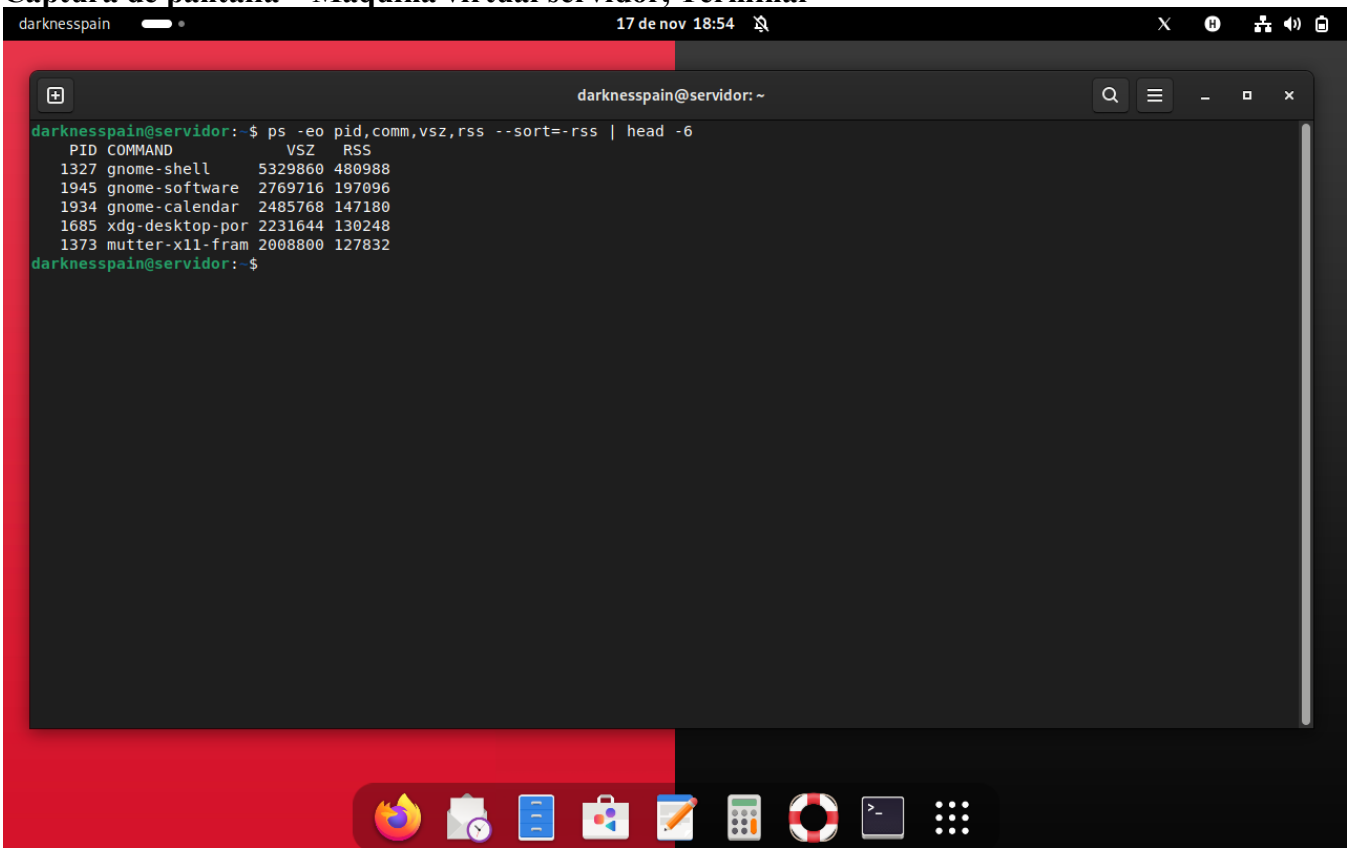
Figura 100
Captura de pantalla – Máquina virtual servidor, Terminal



Nota. «swapon --show» lista tres áreas: /dev/sda5 (partición, 1,1 GB, PRIO -2), /dev/zram0 (partición, 986 MB, PRIO 100) y /swapfile (archivo, 1024 MB, PRIO -3).

La columna PRIO ordena el uso: el kernel recurre primero al área de mayor prioridad, que aquí es zram (100), el swap comprimido en RAM. Solo cuando esa área se llena pasa a las de prioridad menor (-2 y -3). Tener tres áreas con prioridades distintas es una estrategia real: zram absorbe primero porque comprimir en RAM es mucho más rápido que escribir en disco, y el archivo y la partición quedan como respaldo cuando la compresión no alcanza.

Figura 101
Captura de pantalla – Máquina virtual servidor, Terminal



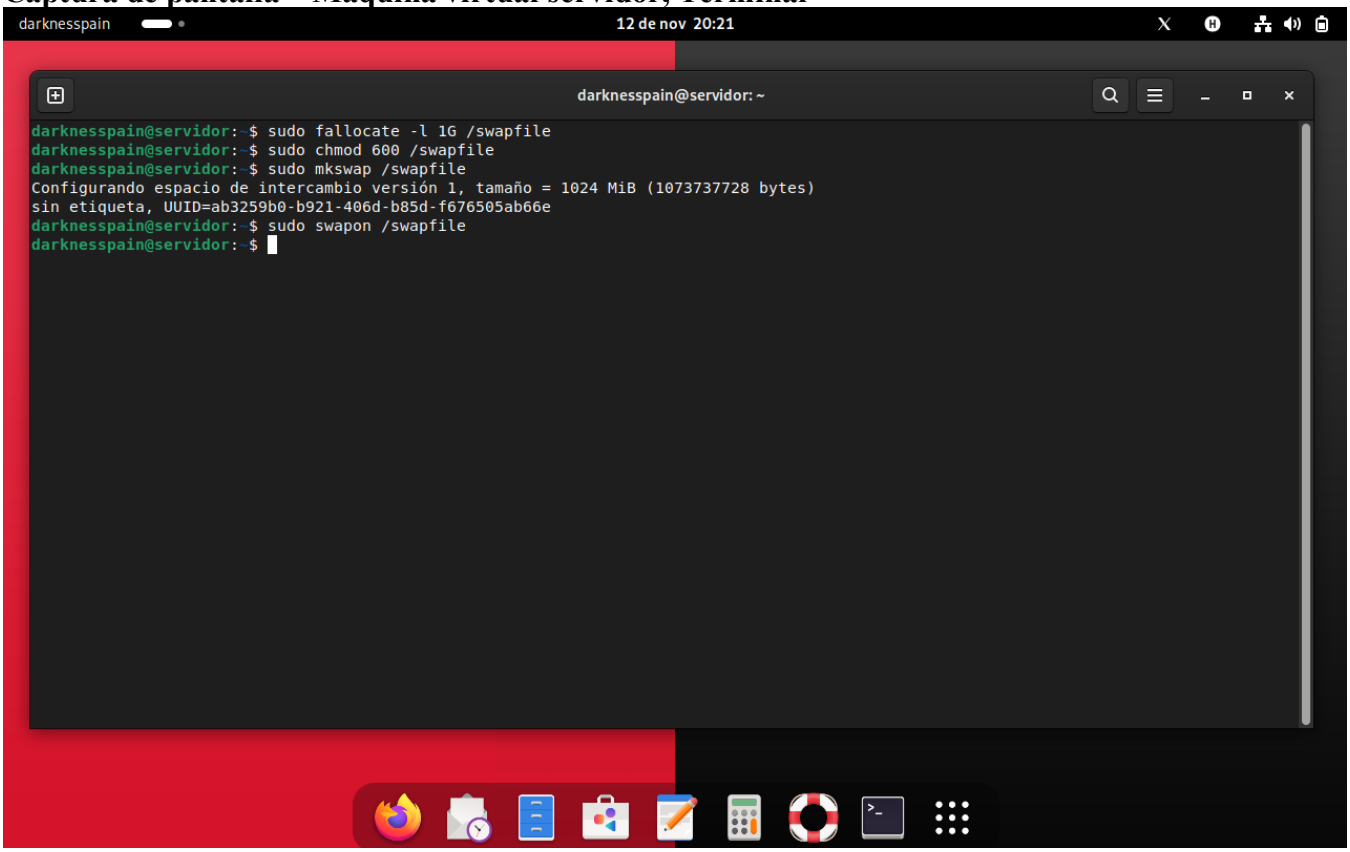
Nota. «ps -eo pid,comm,vsz,rss --sort=-rss | head -6» muestra en cabeza a gnome-shell (PID 1176) con VSZ 5329056 kB y RSS 479508 kB, seguido de gnome-software con 244404 kB de RSS.

La comparación entre VSZ y RSS es la mejor ilustración del espacio de direcciones lógico: gnome-shell «cree» tener 5,3 GB de memoria virtual, pero solo ocupa 479 MB de RAM. Esa diferencia no es un error ni una fuga: el proceso reserva direcciones para bibliotecas, pilas y regiones que aún no ha tocado, y el sistema solo le asigna marcos físicos cuando escribe en ellas (paginación por demanda). Por eso un servidor puede ejecutar procesos cuyo VSZ sumado supere varias veces la RAM instalada.

A.2 Crear un área de intercambio (swap) en un archivo

Creamos un archivo de 1 GB, le quitamos permisos de lectura al resto de usuarios, lo formateamos como área de swap, lo activamos y lo dejamos en /etc/fstab para que se active solo en cada arranque.

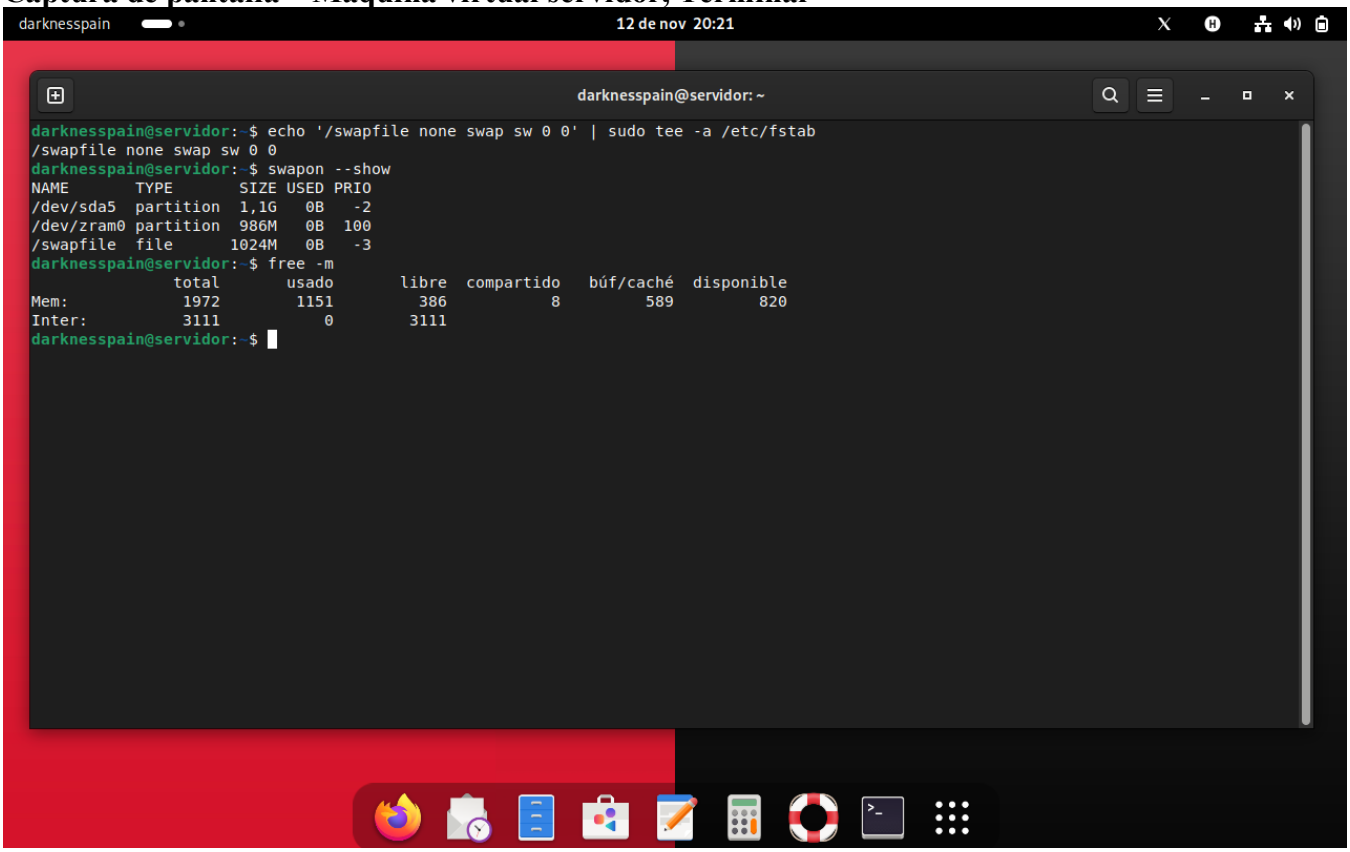
Figura 102
Captura de pantalla – Máquina virtual servidor, Terminal



Nota. «fallocate -l 1G /swapfile», «chmod 600», «mkswap» y «swapon» se ejecutan sin error. mkswap informa un área de 1024 MiB con UUID 3cc37ce4-1017-425e-846e-620699ccd03e.

«fallocate» reserva los bloques de golpe, sin escribirlos, que es mucho más rápido que «dd». El «chmod 600» no es opcional: si cualquier usuario pudiera leer el archivo de swap podría recuperar contraseñas y claves que el kernel haya expulsado de la RAM. mkswap escribe la firma que permite reconocer el área y swapon la conecta al sistema. El UUID que imprime es el identificador del área, no el del archivo.

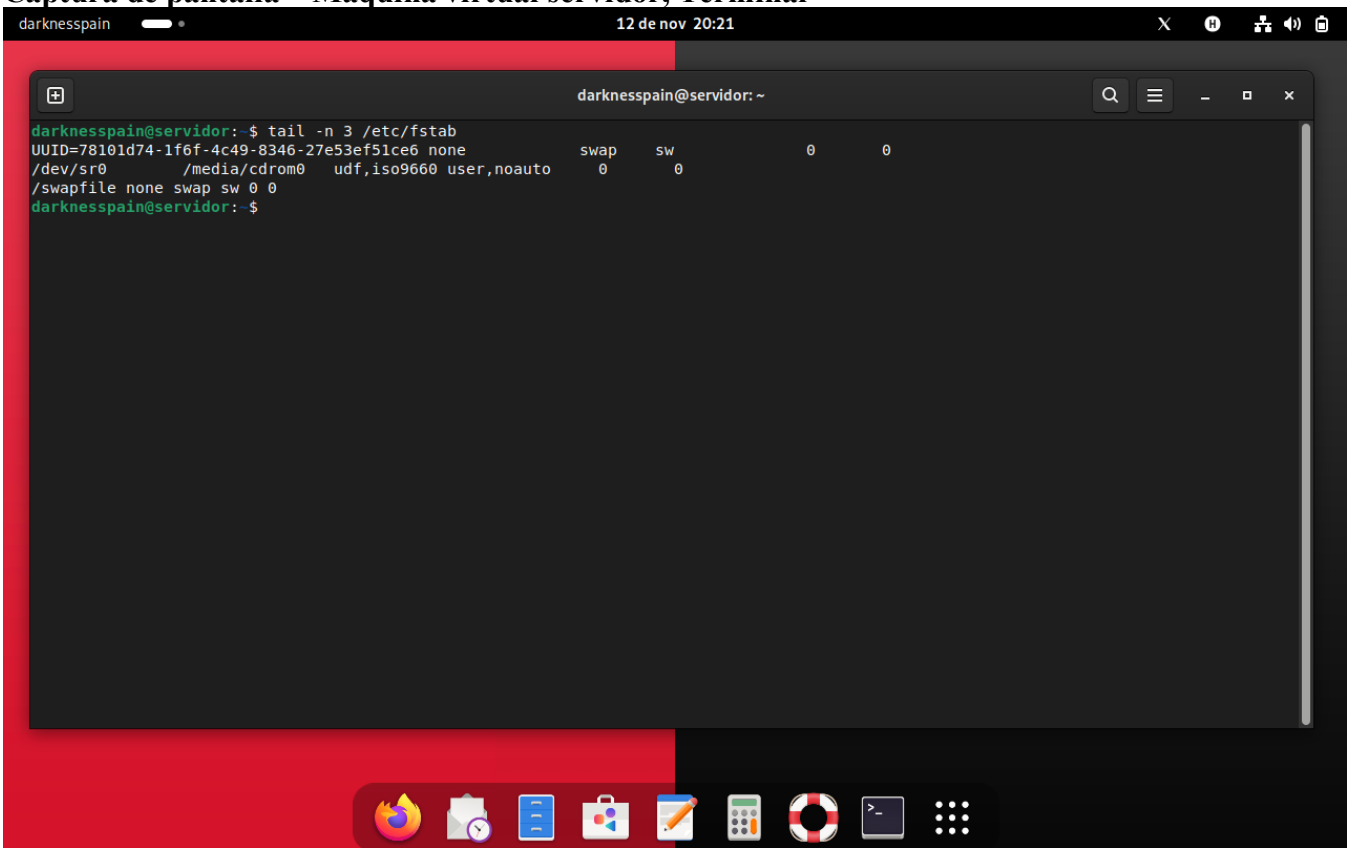
Figura 103
Captura de pantalla – Máquina virtual servidor, Terminal



Nota. La línea «/swapfile none swap sw 0 0» queda añadida a /etc/fstab con «tee». Después, «swapon --show» lista las tres áreas y «free -m» muestra 3111 MB de swap total (2087 + 1024), con 0 MB usados.

La línea de fstab es la que convierte el swap en permanente: sin ella, el archivo se desactivaría en el siguiente arranque y el sistema volvería a tener solo la partición y zram. Los dos ceros finales son el volcado (dump) y el orden de comprobación con fsck; en una entrada de swap ambos van a cero porque no hay nada que volcar ni que revisar. El total de 3111 MB confirma que las tres áreas conviven y que el archivo se sumó a las dos que ya existían.

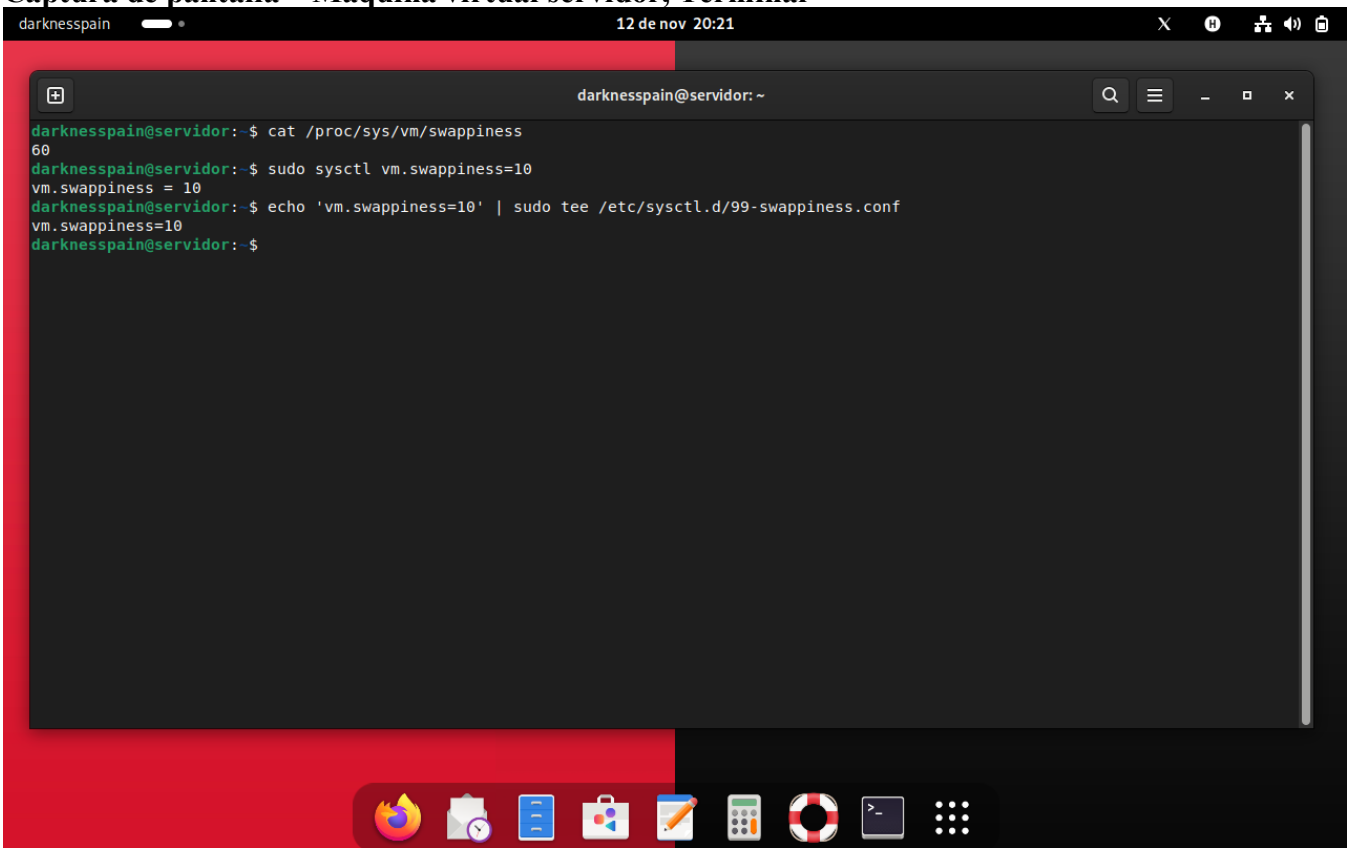
Figura 104
Captura de pantalla – Máquina virtual servidor, Terminal



Nota. «tail -n 3 /etc/fstab» muestra las dos últimas entradas originales (la partición swap por UUID y la unidad de CD) y, al final, la línea «/swapfile none swap sw».

Esta comprobación es puramente defensiva: un error de sintaxis en fstab puede dejar la máquina en modo de emergencia en el próximo arranque. Revisar el archivo con «tail» —y probar siempre con «mount -a» antes de reiniciar— es la rutina que evita quedarse fuera del sistema. La línea quedó al final y con los cuatro campos en el orden correcto: dispositivo, punto de montaje, tipo y opciones.

Figura 105
Captura de pantalla – Máquina virtual servidor, Terminal



Nota. «cat /proc/sys/vm/swappiness» devuelve 60; «sysctl vm.swappiness=10» lo cambia en caliente y el «tee» escribe «vm.swappiness=10» en /etc/sysctl.d/99-swappiness.conf.

swappiness es el peso que el kernel da a expulsar páginas anónimas frente a descartar caché. Con 60 el sistema es más propenso a mandar memoria de procesos al swap; con 10 prefiere conservar los procesos en RAM y liberar caché. El cambio con sysctl solo dura hasta el reinicio, por eso se escribe además el archivo en /etc/sysctl.d: los archivos de ese directorio se aplican al arrancar. Es el patrón habitual para cualquier parámetro del kernel que deba persistir.

A.3 Experimento: ¿qué pasa cuando la RAM no alcanza?

Con D = 825 MB de memoria disponible escribimos primero la predicción y después ejecutamos tres escenarios con «estres_memoria.py», observando el swap y los fallos de página en una segunda terminal con «vmstat 1».

Tabla 1
Datos de memoria observados en A.1 y A.2

Dato	Valor observado	Comando que lo muestra
RAM total (MB)	1972	free -m
Memoria disponible (MB)	825 (845092 kB)	grep MemAvailable

		/proc/meminfo
Swap total (MB). ¿Ya existía swap?	3111; sí, partición de 1,1 GB y zram de 986 MB	swapon --show
Proceso con más RSS	gnome-shell, PID 1176, 479508 kB	ps -eo pid,comm,vsz,rss --sort=-rss
VSZ y RSS de ese proceso	VSZ 5329056 kB / RSS 479508 kB	ps -eo pid,comm,vsz,rss --sort=-rss

Nota. D = 825 MB es el valor que se usó para calcular los tamaños de los experimentos de A.3.

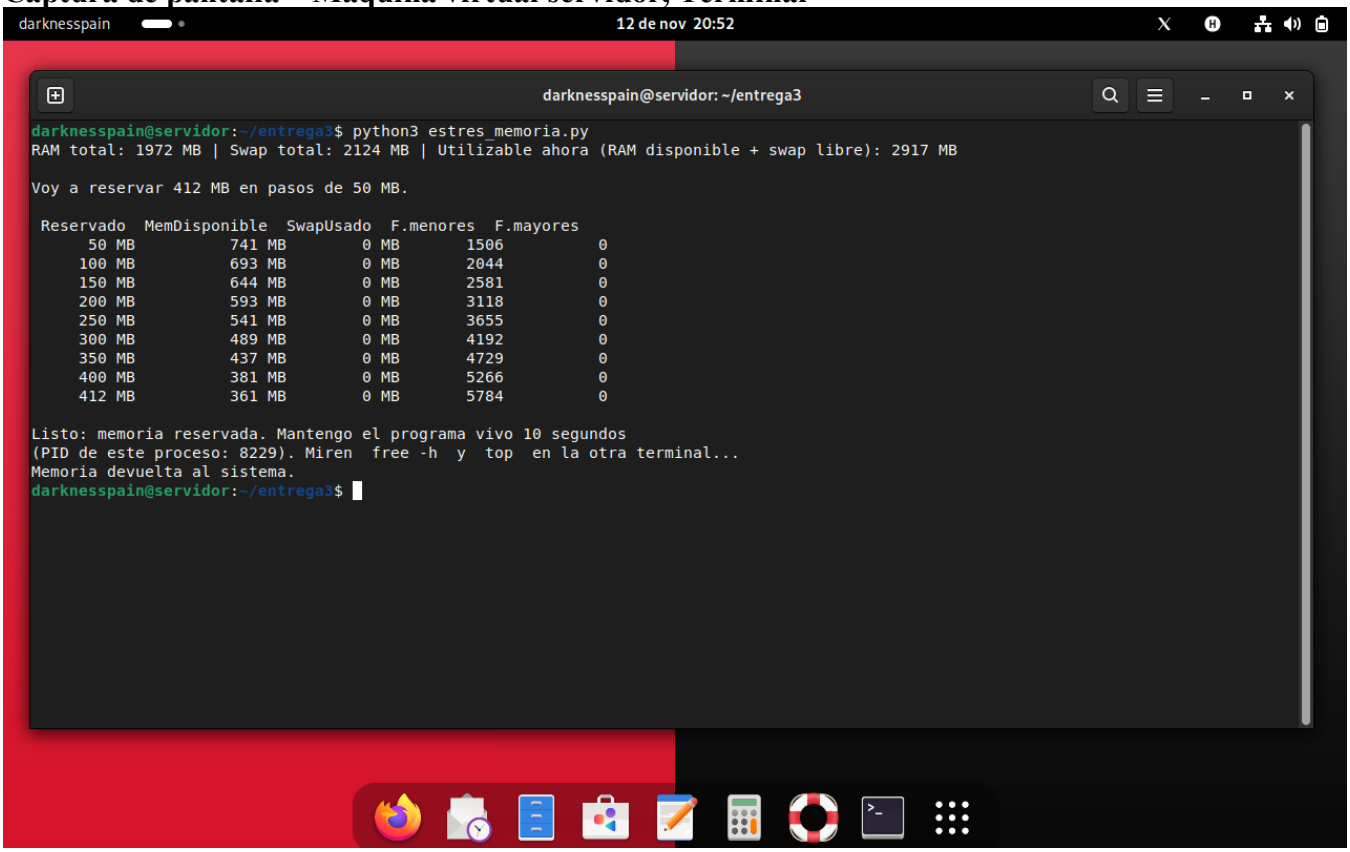
Figura 106
Captura de pantalla – Máquina virtual servidor, Editor nano



Nota. El archivo estres_memoria.py abierto en nano con la línea «MEGAS_A_RESERVAR = 412» resaltada, que corresponde a D/2. El resto del archivo no se modificó.

La guía pide cambiar una sola línea para que el experimento sea reproducible: el programa reserva memoria en pasos de 50 MB, escribe una página cada 4 KB para forzar la asignación real y luego informa, paso a paso, la memoria disponible, el swap usado y los fallos de página del proceso. Al fijar 412 MB estamos pidiendo poco menos de la mitad de lo disponible, que es el escenario «cómodo» del experimento.

Figura 107
Captura de pantalla – Máquina virtual servidor, Terminal



Nota. La tabla del programa muestra el avance de 50 a 412 MB: la memoria disponible baja de 741 a 361 MB, la columna SwapUsado queda en 0 MB en todos los pasos y los fallos mayores no aumentan (se mantienen en 0).

El resultado coincide con la predicción: reservar 412 MB con 825 MB disponibles no obliga al kernel a expulsar páginas anónimas, así que el swap no se toca. Los fallos menores sí crecen (de 1506 a 5784) porque cada página nueva provoca un fallo menor al asignarse por primera vez: el dato ya está en RAM, solo hay que crear la traducción en la tabla de páginas. Distinguir «fallo menor» de «fallo mayor» es la clave para interpretar la tabla.

Figura 108

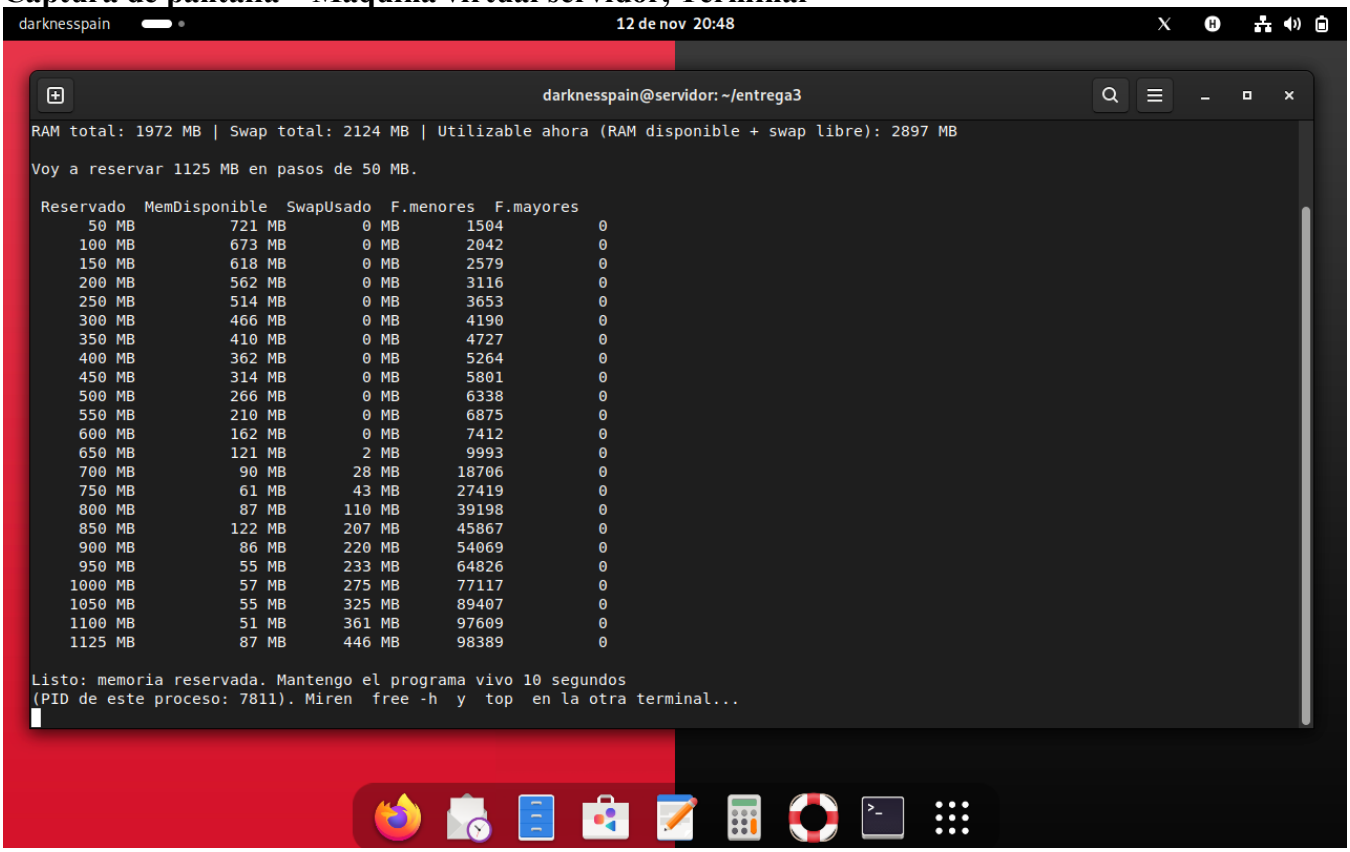
Captura de pantalla – Máquina virtual servidor, Editor nano



Nota. La misma línea del programa, ahora con «MEGAS_A_RESERVAR = 1125», que es D + 300. Es el valor usado en los experimentos 2 y 3.

Pedir 1125 MB cuando solo hay unos 825 MB disponibles es exactamente el escenario que queremos provocar: el sistema tendrá que recuperar caché primero y, cuando no alcance, expulsar páginas de los procesos al swap. El programa tiene una protección —no reserva más del 80 % de la memoria utilizable (RAM disponible + swap libre)— que aquí no se activa porque el total utilizable ronda los 3,8 GB y el 80 % son unos 3 GB.

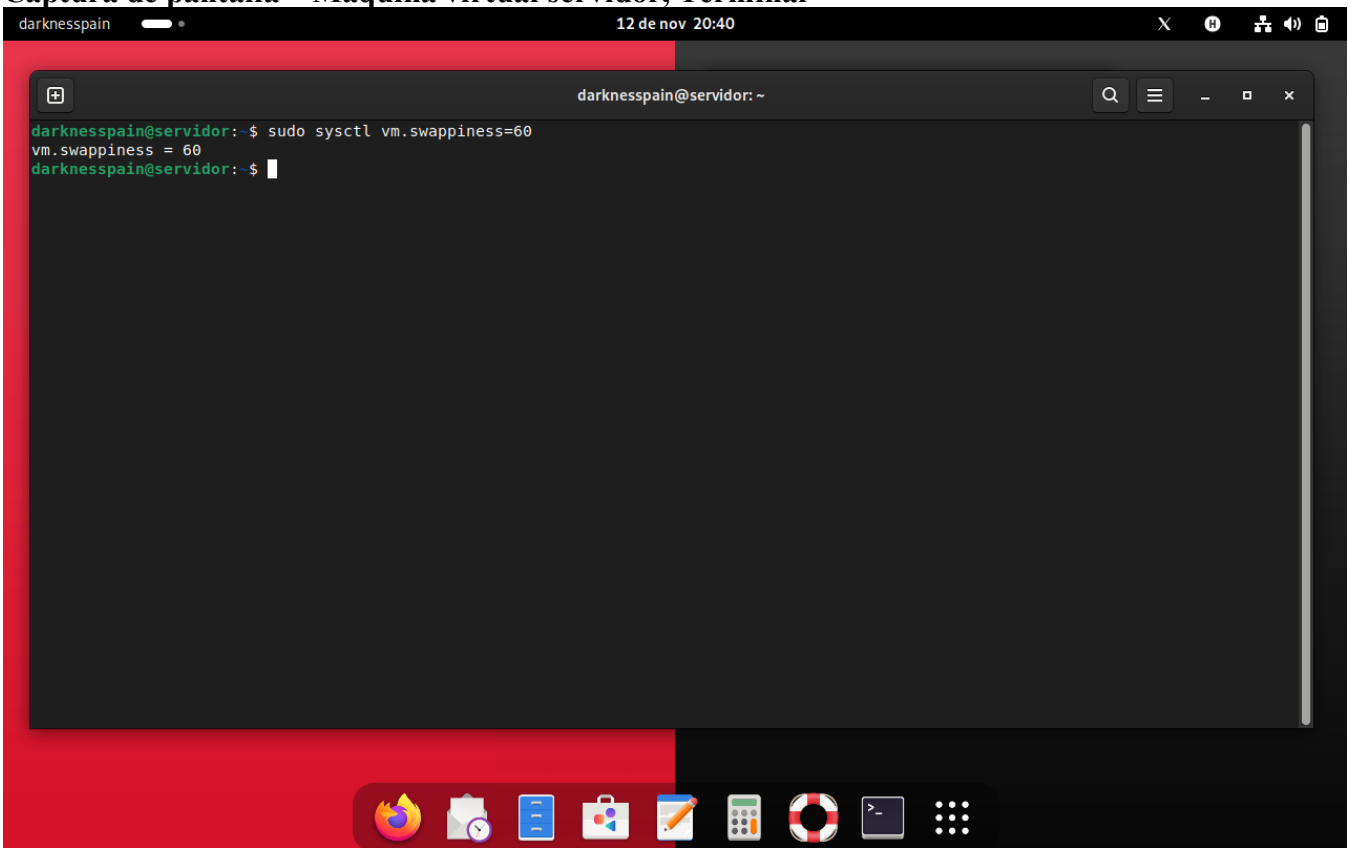
Figura 109
Captura de pantalla – Máquina virtual servidor, Terminal



Nota. Con 1125 MB solicitados, la memoria disponible cae hasta 87-61 MB y el swap usado sube paso a paso hasta 446 MB. Los fallos menores llegan a 98389 y el sistema se vuelve lento durante la reserva.

Aquí se ve el swapping en acción: como la demanda supera la RAM disponible, el kernel empieza a escribir páginas anónimas en las áreas de intercambio y la columna SwapUsado deja de ser cero. El sistema sigue funcionando porque el swap amplía la memoria, pero a un coste enorme: cada acceso a una página expulsada provoca un fallo mayor y una lectura de disco. Ese intercambio continuo es la hiperpaginación, y es la razón por la que «agregar swap» no equivale a «agregar RAM».

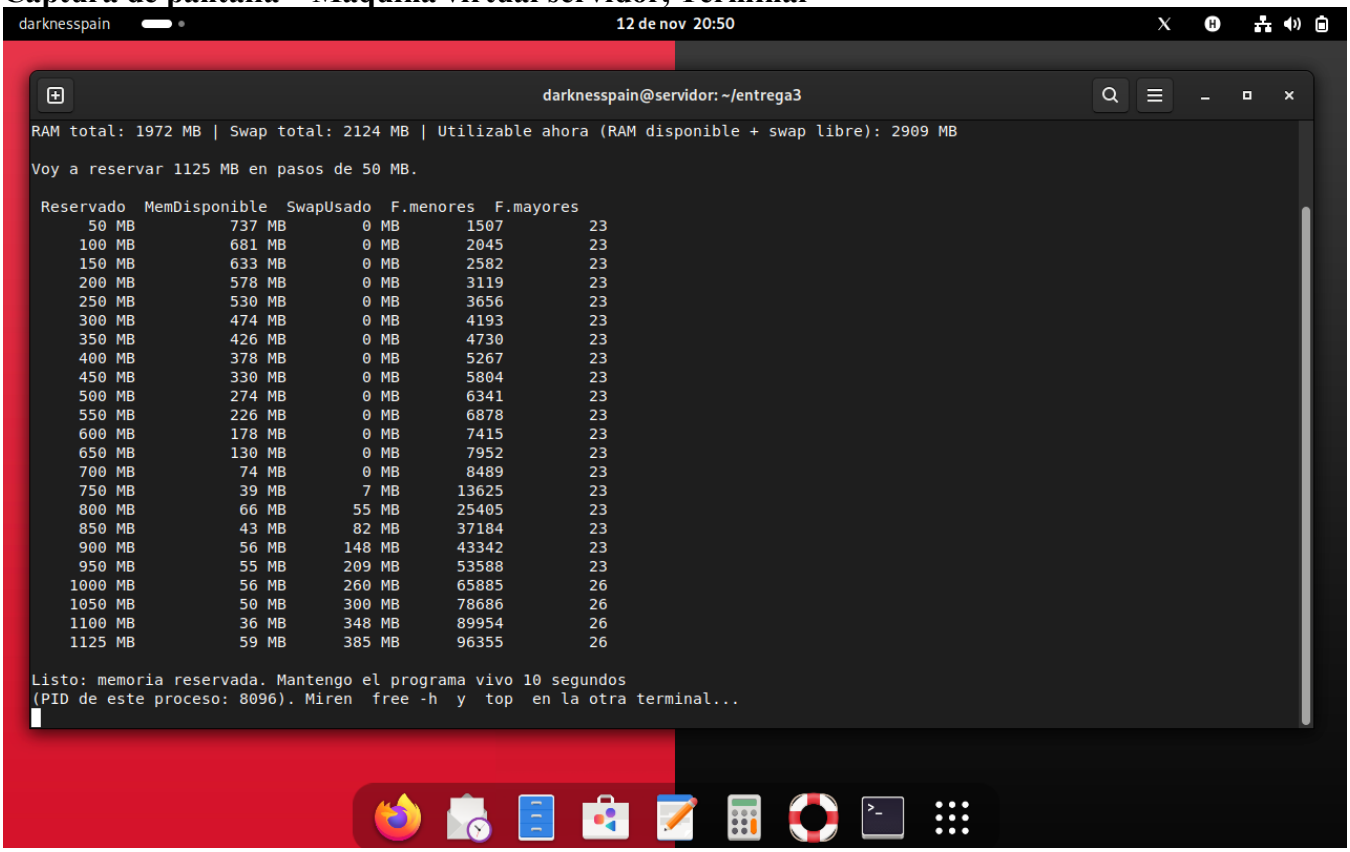
Figura 110
Captura de pantalla – Máquina virtual servidor, Terminal



Nota. «sudo sysctl vm.swappiness=60» responde «vm.swappiness = 60», el valor que la guía pide para el tercer experimento.

El tercer escenario repite el segundo cambiando solo la inclinación del kernel a usar swap. Con swappiness 60 el sistema prefiere mover páginas anónimas al disco antes que descartar caché, así que esperábamos más swap ocupado que en el experimento 2. Cambiar una sola variable y volver a medir es la forma correcta de atribuir una diferencia a una causa: si hubiéramos cambiado también el tamaño, no sabríamos a qué se debe el resultado.

Figura 111
Captura de pantalla – Máquina virtual servidor, Terminal



Nota. Con swappiness 60 y los mismos 1125 MB, el swap usado llega a 385 MB y la memoria disponible termina en 59 MB; los fallos mayores del proceso suben hasta 26.

El resultado no coincidió con nuestra predicción: esperábamos más swap que en el experimento 2 y quedó menos (385 MB frente a 446 MB). La explicación que encontramos es que, con la memoria tan justa, el kernel termina recurriendo al swap en los dos casos y la diferencia depende de cuánta caché pudo recuperar antes y de la compresión de zram, no solo del valor de swappiness. Lo dejamos escrito porque el valor de predecir está en comparar: la hipótesis era razonable y los datos mostraron que el sistema tiene más de una válvula de escape.

Tabla 2
Predicción y resultado de los tres experimentos de A.3

Experimento	MB reservados	Predicción	Swap usado final	F. mayores	¿si/so > 0?
1 (D/2)	412	Sin swap: cabe en RAM	0 MB	0	No
2 (D+300)	1125	Swap crece y aparecen fallos mayores	446 MB	Aumentan	Sí
3 (swappiness 60)	1125	Más swap que en el experimento 2	385 MB	26	Sí

Nota. Los valores salen de las tablas del programa en las Figuras 94, 96 y 98. En el experimento 1 los fallos mayores se mantuvieron en cero; en el 3 llegaron a 26.

- **Pregunta 1. VSZ (memoria virtual que el proceso «cree» tener) casi siempre es**

mucho mayor que RSS (memoria física que realmente ocupa). ¿Por qué el sistema puede darse el lujo de prometer más de lo que tiene?

Porque el espacio de direcciones de un proceso es una abstracción, no una reserva de RAM. Cuando gnome-shell pide memoria, el kernel le entrega un rango de direcciones virtuales y anota en su estructura que esas páginas existen, pero no le asigna marcos físicos hasta que el proceso escribe en ellas. Ese mecanismo se llama paginación por demanda: el primer acceso a una página provoca un fallo menor, el kernel busca un marco libre (o reutiliza uno limpio) y crea la traducción en la tabla de páginas. Así, el VSZ de 5,3 GB que vimos en la Figura 90 convive sin problema con 1972 MB de RAM: la mayoría de esas direcciones corresponden a bibliotecas compartidas y a regiones que nunca se tocan. El sistema solo se compromete de verdad cuando el proceso usa la memoria, y si en ese momento no hay marcos disponibles, recurre al swap. La promesa es posible porque el patrón de uso real de los procesos suele ser mucho menor que su espacio de direcciones; cuando deja de serlo, aparece la hiperpaginación que provocamos en A.3.

A.4 Algoritmos de reemplazo: FIFO, LRU y Óptimo

Resolvimos a mano las tres tablas con 3 marcos para la cadena 2, 3, 2, 1, 5, 2, 4, 5, 3, 2, 5, 2 y después las verificamos con el simulador de la docente.

Tabla 3

Resolución manual FIFO (3 marcos)

Pide	2	3	2	1	5	2	4	5	3	2	5	2
Marco 1	2	2	2	2	5	5	5	5	3	3	3	3
Marco 2		3	3	3	3	2	2	2	2	2	5	5
Marco 3				1	1	1	4	4	4	4	4	2
¿F o A?	F	F	A	F	F	F	F	A	F	A	F	F

Nota. FIFO expulsa la página que entró primero: por eso al pedir 5 sale el 2, y al pedir 4 sale el 1. Total: 9 fallos.

Tabla 4

Resolución manual LRU (3 marcos)

Pide	2	3	2	1	5	2	4	5	3	2	5	2
Marco 1	2	2	2	2	2	2	2	2	3	3	3	3
Marco 2		3	3	3	5	5	5	5	5	5	5	5
Marco 3				1	1	1	4	4	4	2	2	2
¿F o A?	F	F	A	F	F	A	F	A	F	F	A	A

Nota. LRU expulsa la que lleva más tiempo sin usarse: al pedir 5 sale el 3 (usado en el paso 2) y al pedir 4 sale el 1. Total : 7 fallos.

Tabla 5

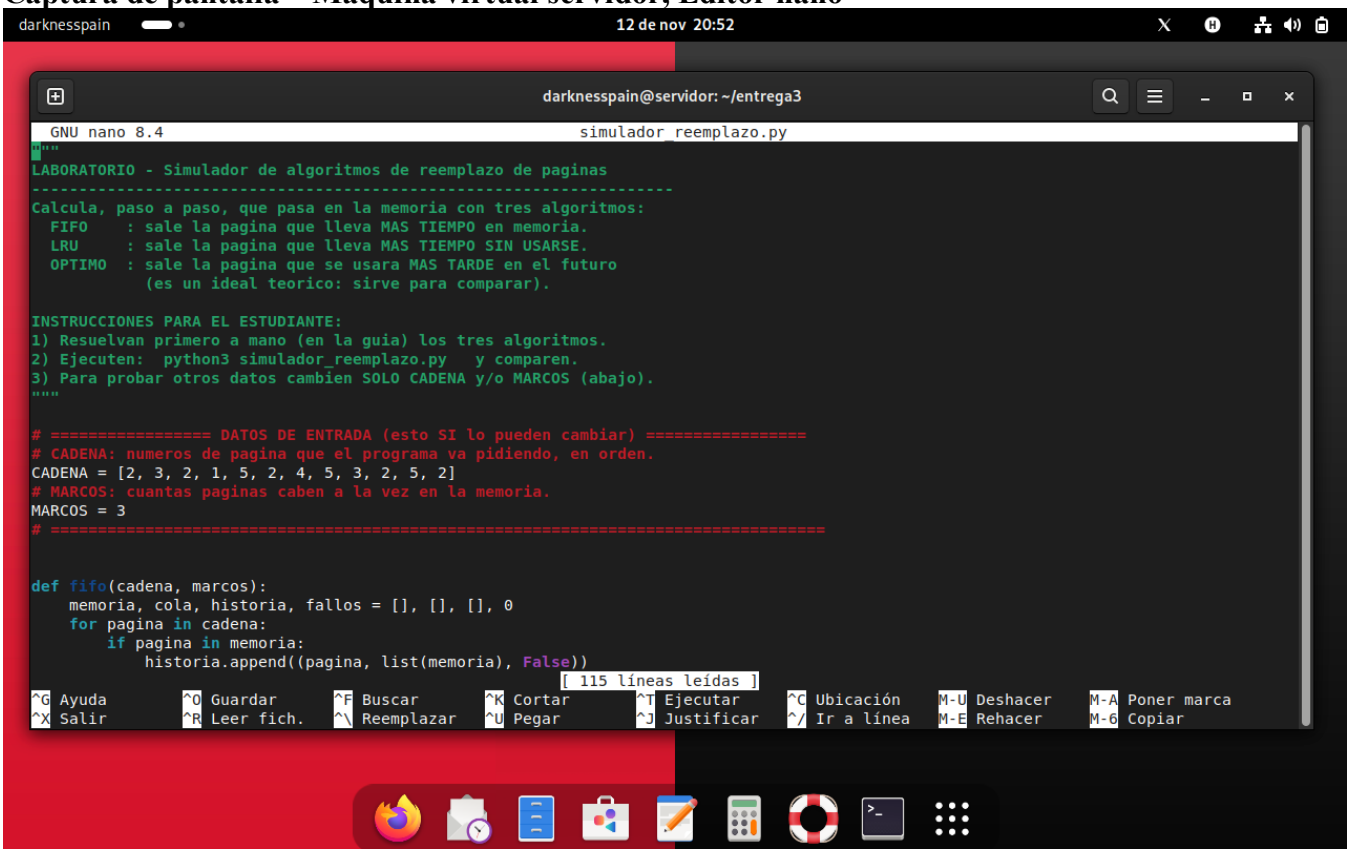
Resolución manual ÓPTIMO (3 marcos)

Pide	2	3	2	1	5	2	4	5	3	2	5	2
Marco 1	2	2	2	2	2	2	2	2	2	2	2	2
Marco 2		3	3	3	3	3	4	4	4	4	4	4
Marco 3				1	5	5	5	5	3	3	5	5
¿F o A?	F	F	A	F	F	A	F	A	F	A	F	A

Nota. El Óptimo expulsa la página que tardará más en volver: al pedir 5 sale el 1, que no se usa nunca más, y al pedir 4 sale el 3, que reaparece en el paso 9. Total: 6 fallos.

Figura 112

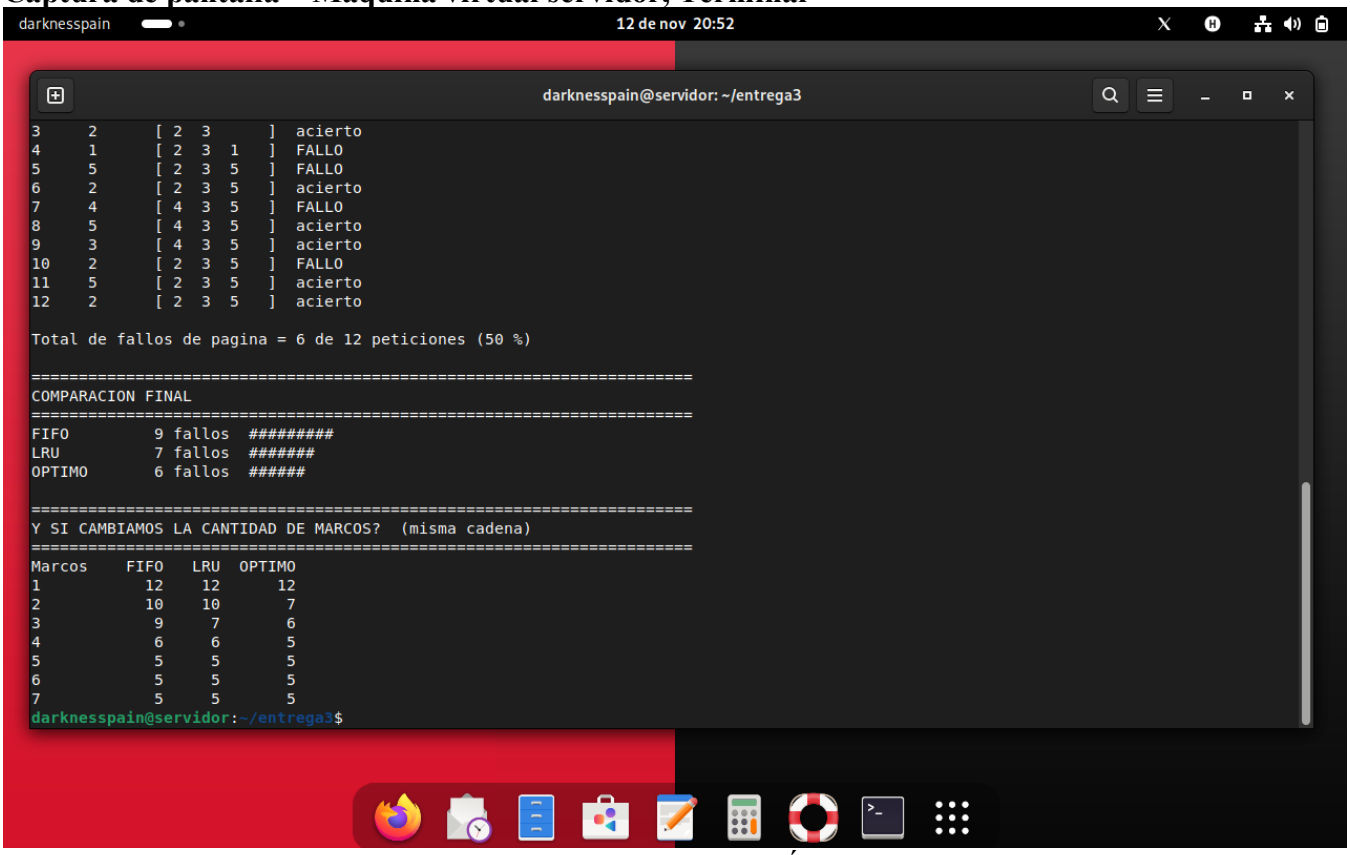
Captura de pantalla – Máquina virtual servidor, Editor nano



Nota. El simulador abierto en nano con «CADENA = [2, 3, 2, 1, 5, 2, 4, 5, 3, 2, 5, 2]» y «MARCOS = 3», que son los datos de la tabla manual.

Cambiar solo la cadena y el número de marcos mantiene el experimento controlado: el algoritmo es el mismo código de la docente y lo único que varía son los datos de entrada. Así la comparación entre nuestra tabla y la salida del programa mide si entendimos el criterio de expulsión, no si acertamos con una implementación distinta. La cadena elegida tiene repeticiones (2 aparece cuatro veces) que hacen visible la diferencia entre FIFO y LRU.

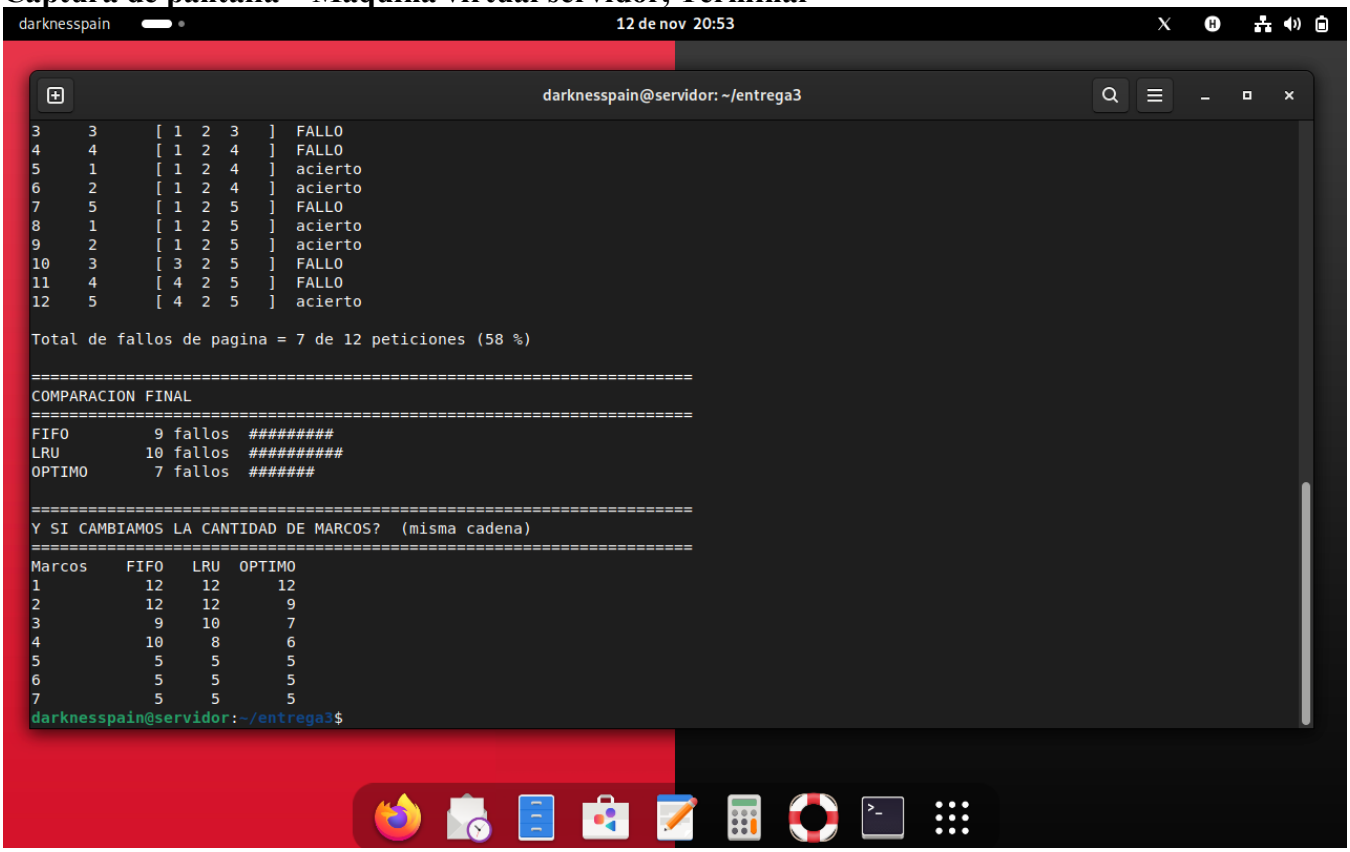
Figura 113
Captura de pantalla – Máquina virtual servidor, Terminal



Nota. El simulador imprime el paso a paso de FIFO, LRU y ÓPTIMO y el resumen final: FIFO 9 fallos, LRU 7 fallos y ÓPTIMO 6 fallos de 12 peticiones. También muestra la tabla de fallos para 1 a 7 marcos.

Los tres totales coinciden exactamente con nuestras tablas manuales, así que el ejercicio a mano estaba bien resuelto. La diferencia entre algoritmos se explica por la información que cada uno usa: FIFO solo mira cuándo entró una página, LRU mira cuándo se usó por última vez y el Óptimo mira el futuro. Cuanta más información, menos fallos: 9, 7 y 6. En un sistema real el Óptimo no se puede implementar porque exigiría conocer las peticiones futuras; se usa como cota inferior para evaluar qué tan bueno es un algoritmo práctico.

Figura 114
Captura de pantalla – Máquina virtual servidor, Terminal



Nota. Con la cadena de Belady 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5 y 3 marcos, el resumen da FIFO 9, LRU 10 y ÓPTIMO 7; la tabla inferior muestra que con 4 marcos FIFO sube a 10, LRU baja a 8 y ÓPTIMO baja a 6.

La captura contiene la anomalía de Belady en una sola pantalla: al pasar de 3 a 4 marcos, FIFO empeora (de 9 a 10 fallos) mientras LRU y Óptimo mejoran. FIFO no cumple la propiedad de pila porque su criterio de expulsión no depende del patrón de uso reciente, y por eso más memoria puede desordenar la cola de entrada. LRU y Óptimo sí son algoritmos de pila: el conjunto de páginas que mantienen con n marcos es siempre un subconjunto del que mantendrían con n+1, de modo que agregar un marco nunca puede aumentar los fallos.

- **Pregunta 2. Si en «swapon --show» aparecen dos áreas (una partición creada por el instalador y su archivo /swapfile), ¿qué significa la columna PRIO? ¿Cuál se usará primero?**

PRIO es la prioridad del área de intercambio y el kernel usa primero la de número más alto. En nuestra máquina conviven tres áreas: zram0 con PRIO 100, la partición /dev/sda5 con PRIO -2 y el archivo /swapfile con PRIO -3 (Figura 89). El sistema llena primero zram porque es

memoria comprimida y no cuesta una escritura a disco; solo cuando esa área se agota recurre a la partición y al archivo. Las prioridades negativas indican áreas de reserva. Este ordenamiento se relaciona directamente con el concepto de swapping: cuando la suma de las páginas activas supera los marcos disponibles, el kernel elige páginas víctimas y las traslada a estas áreas; el conjunto de páginas que un proceso necesita en un intervalo —su conjunto de trabajo— es lo que determina si el intercambio será esporádico o constante.

- **Pregunta 3. Distingan fallo de página menor y mayor. ¿Por qué un fallo mayor es mucho más costoso? Si el sistema pasara casi todo su tiempo moviendo páginas, ¿cómo se llama ese fenómeno y cómo se relaciona con el conjunto de trabajo?**

Un fallo menor ocurre cuando la página que el proceso pide ya está en memoria física pero no en su tabla de páginas: basta con crear la entrada que traduce la dirección virtual en el marco real, y eso cuesta microsegundos. Un fallo mayor ocurre cuando la página no está en RAM y hay que traerla del disco (del área de swap, si es anónima): ahí se paga una operación de E/S que tarda milisegundos, cientos de veces más, y mientras tanto el proceso queda bloqueado. En las tablas de A.3 se ve la diferencia: el experimento 1 solo produjo fallos menores, y en el 3 aparecieron fallos mayores cuando el sistema empezó a expulsar páginas. Si el sistema pasa la mayor parte del tiempo moviendo páginas se llama hiperpaginación o thrashing, y se explica con el conjunto de trabajo: cada proceso necesita un conjunto de páginas activas, y si la suma de esos conjuntos supera los marcos disponibles, cada página que se trae expulsa otra que se necesitará enseguida. La solución no es agregar swap, sino reducir la carga simultánea o aumentar los marcos.

- **Pregunta 4. ¿Qué algoritmo empeoró al agregar un marco? ¿Por qué LRU y Óptimo nunca sufren esta anomalía? ¿Qué le aconsejarían a un administrador que piensa «le agrego más RAM y se arregla»?**

El que empeoró fue FIFO: pasó de 9 a 10 fallos al subir de 3 a 4 marcos. Es la anomalía de Belady, y ocurre porque FIFO no es un algoritmo de pila: su decisión depende solo del orden de llegada, así que un marco más cambia la secuencia de expulsiones y puede sacar antes una página que se volverá a usar. LRU y el Óptimo sí son de pila, porque su criterio se basa en el uso

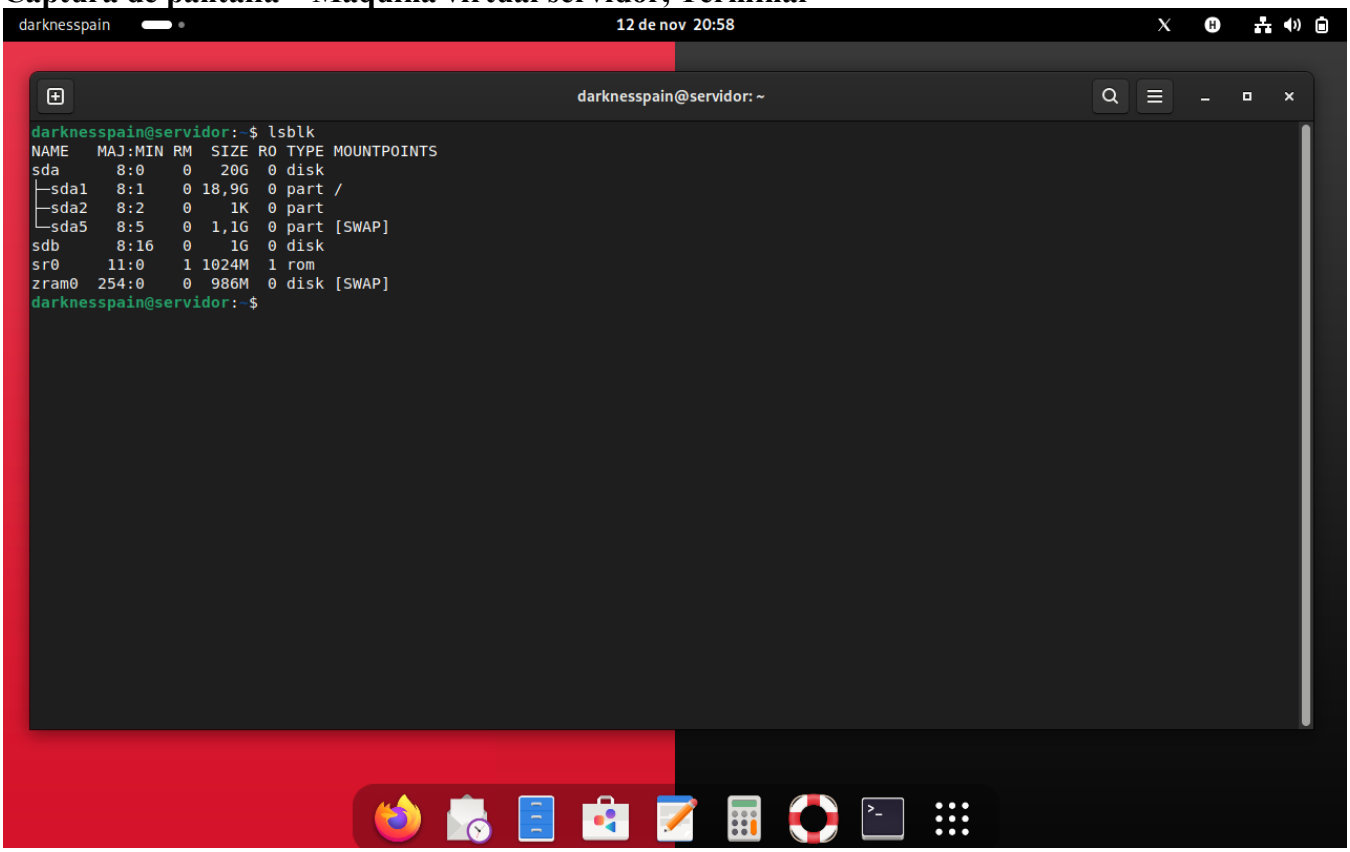
(pasado o futuro): el conjunto de páginas que mantienen con n marcos es siempre un subconjunto del que mantendrían con n+1, y por eso un marco extra nunca puede aumentar los fallos. Al administrador le diría que más RAM casi siempre ayuda, pero no corrige un mal patrón de acceso ni un algoritmo inadecuado: hay que medir antes y después, y vigilar el conjunto de trabajo. En nuestra propia máquina, agregar swap mejoró la supervivencia pero empeoró el rendimiento, que es el mismo malentendido en otro nivel.

Parte B — Sistema de archivos

B.1 Agregar un disco virtual de 1 GB

Con el servidor apagado agregamos un segundo disco SATA de 1 GB reservado dinámicamente, llamado datos.vdi, y lo verificamos al arrancar.

Figura 115
Captura de pantalla – Máquina virtual servidor, Terminal



Nota. «lsblk» muestra el disco nuevo /dev/sdb de 1 GB sin particiones ni puntos de montaje, además del disco del sistema (sda, 20 GB) y la unidad óptica sr0.

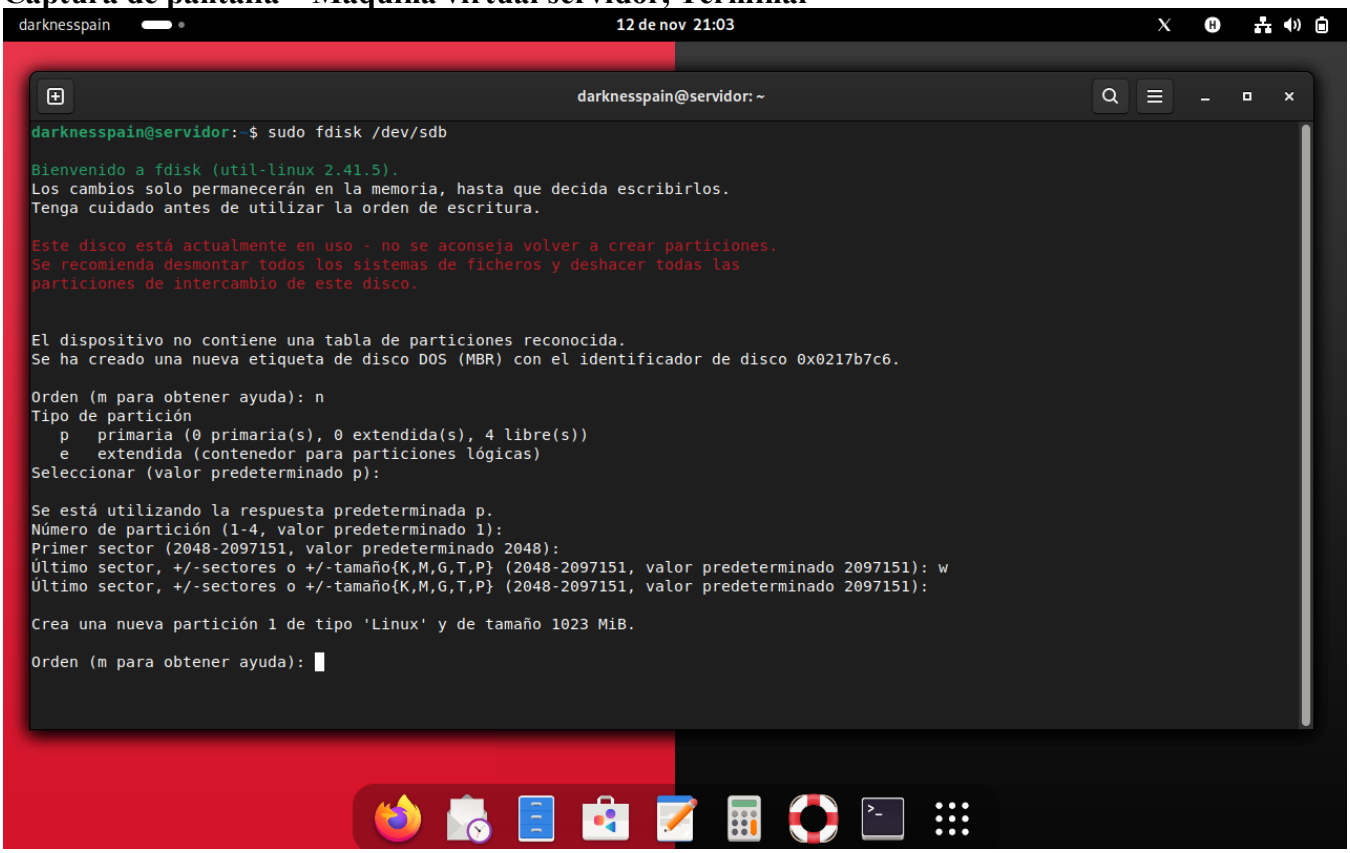
Antes de particionar hay que confirmar tres cosas: que el disco aparece, que es el de 1 GB y que no tiene particiones. Es la comprobación que evita el error más grave posible en esta

práctica, que sería formatear el disco del sistema. El nombre sdb lo asigna el kernel por orden de detección; como el disco se agregó después, queda segundo. Ese detalle es justamente el argumento de la Pregunta 5 sobre por qué fstab debe usar UUID y no el nombre del dispositivo.

B.2 Particionar, formatear y montar

Creamos una partición primaria con fdisk, la formateamos en ext4 con la etiqueta DATOS, la montamos en /datos y dejamos el montaje permanente por UUID.

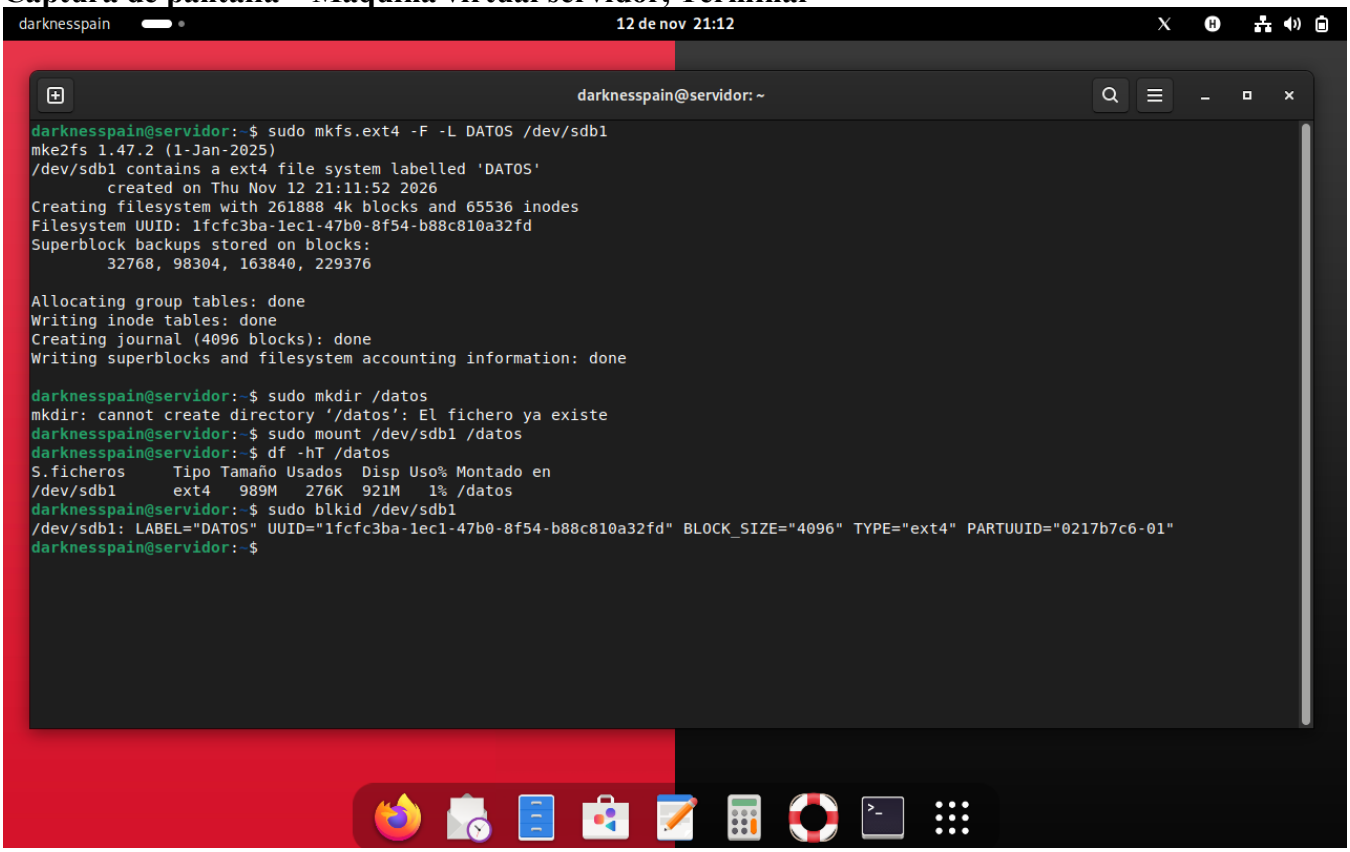
Figura 116
Captura de pantalla – Máquina virtual servidor, Terminal



Nota. Sesión de «fdisk /dev/sdb»: tras la orden «n» y aceptar los valores por defecto, fdisk informa «Crea una nueva partición 1 de tipo 'Linux' y de tamaño 1023 MiB» y vuelve al menú de órdenes, donde se escribe «w» para guardar.

fdisk trabaja sobre la tabla de particiones y no toca los datos hasta que se escribe la orden «w»; hasta ese momento todo está en memoria. Aceptar los valores por defecto toma el primer sector alineado (2048) y el último disponible, con lo que la partición ocupa casi todo el disco. La alineación a 2048 sectores no es un capricho: los discos y los SSD trabajan por bloques de 4 KB y una partición desalineada obliga a leer dos bloques físicos por cada bloque lógico.

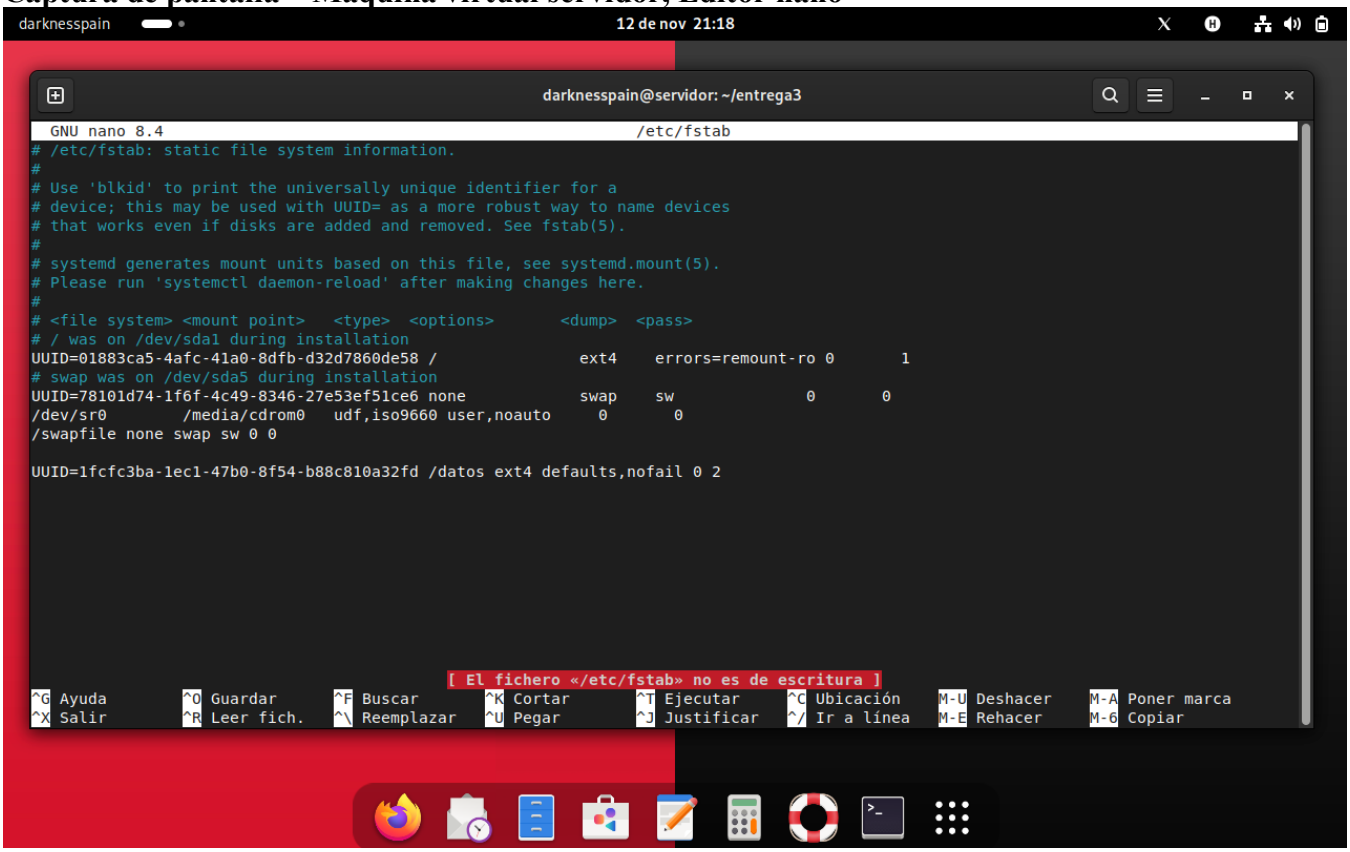
Figura 117
Captura de pantalla – Máquina virtual servidor, Terminal



Nota. «mkfs.ext4 -F -L DATOS /dev/sdb1» crea el sistema de archivos con 261888 bloques de 4 KB y 65536 inodos, UUID 1fcfc3ba-1ec1-47b0-8f54-b88c810a32fd. Después «mount» deja /datos montado, «df -hT» muestra 989 MB con 921 MB disponibles y «blkid» confirma el tipo ext4.

El formateo escribe las estructuras del sistema de archivos: superbloque, tablas de inodos, mapa de bloques y journal. Los 65536 inodos son el límite de archivos que podrá tener el disco, un dato que será decisivo en los incidentes de la Parte F: un disco puede tener bloques libres y aun así rechazar archivos nuevos si se quedó sin inodos. Se usó «-F» porque, tras crear la partición, el automontaje del escritorio la había abierto y mke2fs se negaba a formatear un dispositivo «en uso»; lo anotamos como ajuste.

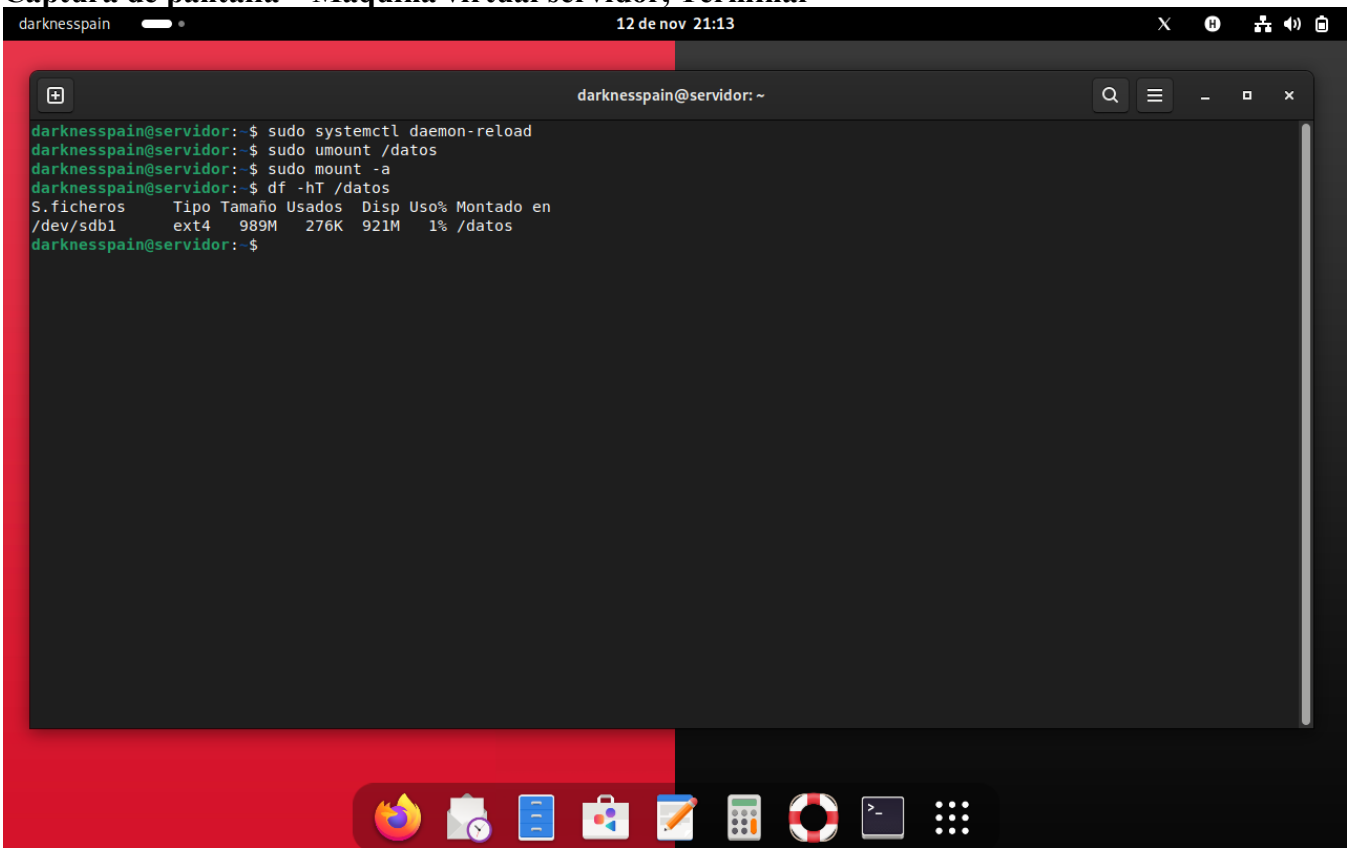
Figura 118
Captura de pantalla – Máquina virtual servidor, Editor nano



Nota. El archivo /etc/fstab abierto en nano, al final, con la línea «UUID=1fcfc3ba-1ec1-47b0-8f54-b88c810a32fd /datos ext4 defaults,nofail 0 2».

La entrada usa el UUID en lugar de /dev/sdb1 y añade la opción «nofail». El UUID identifica el sistema de archivos y no depende del orden en que el kernel detecte los discos, así que seguirá apuntando al mismo disco aunque mañana se agregue otro y la partición pase a llamarse sdc1. «nofail» evita que un fallo de este disco impida el arranque del sistema: sin esa opción, systemd entraría en modo de emergencia si el disco no estuviera disponible.

Figura 119
Captura de pantalla – Máquina virtual servidor, Terminal



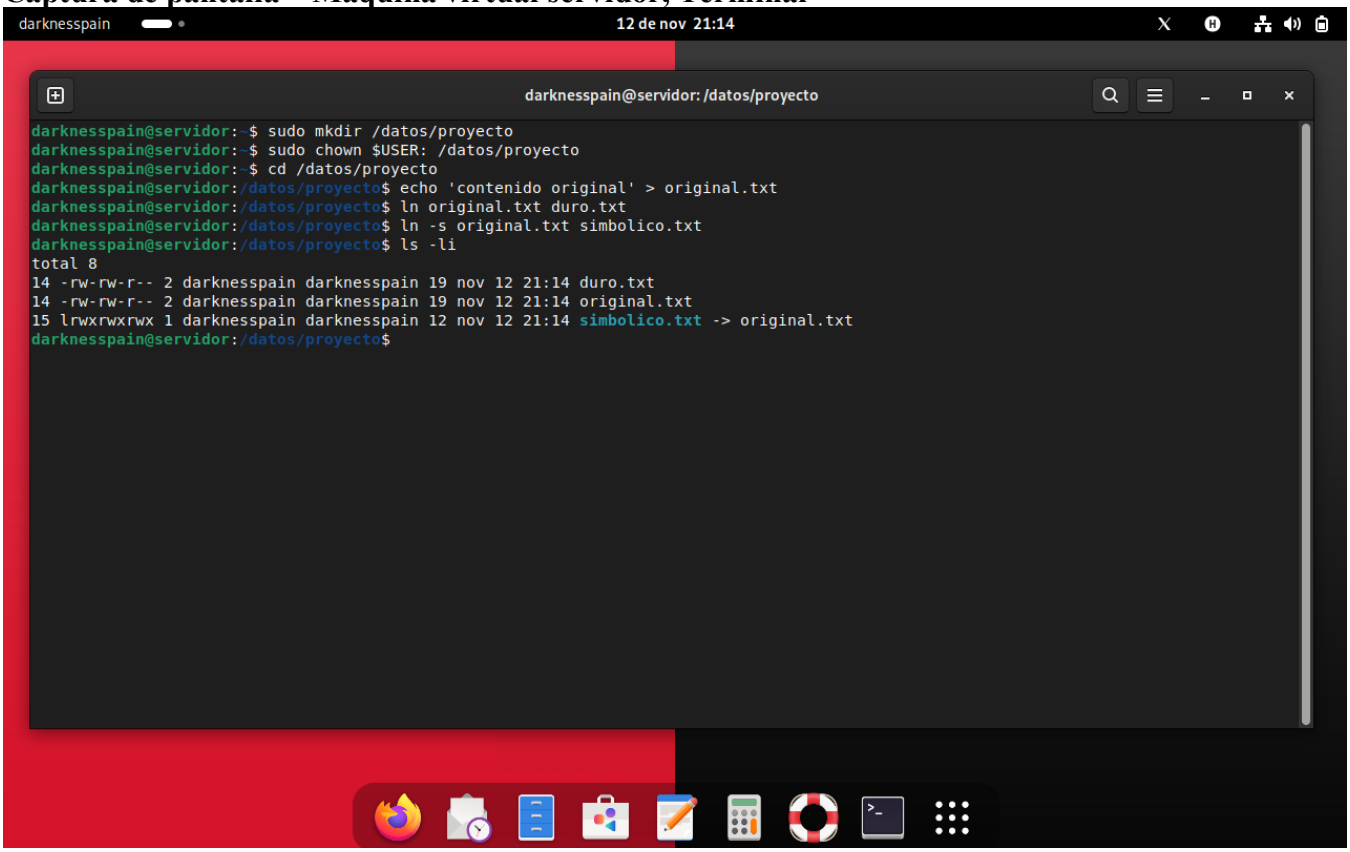
Nota. Tras «systemctl daemon-reload», «umount /datos», «mount -a» y «df -hT /datos», el disco vuelve a aparecer montado en /datos con 989 MB de tamaño y 921 MB disponibles.

«mount -a» monta todo lo que esté en fstab y es la prueba de que la línea quedó bien escrita: si hubiera un error de sintaxis o un UUID equivocado, el comando fallaría aquí y no en el próximo arranque. El orden importa: daemon-reload hace que systemd relea sus unidades de montaje, umount desmonta para probar desde cero y mount -a vuelve a montar leyendo fstab. Que df muestre el disco otra vez demuestra que el montaje ya no depende de un comando manual.

B.3 Inodos y enlaces: ¿qué es realmente un archivo?

Creamos un archivo, un enlace duro y un enlace simbólico al mismo contenido, y comprobamos qué pasa al borrar el nombre original. Antes de ejecutar escribimos la predicción.

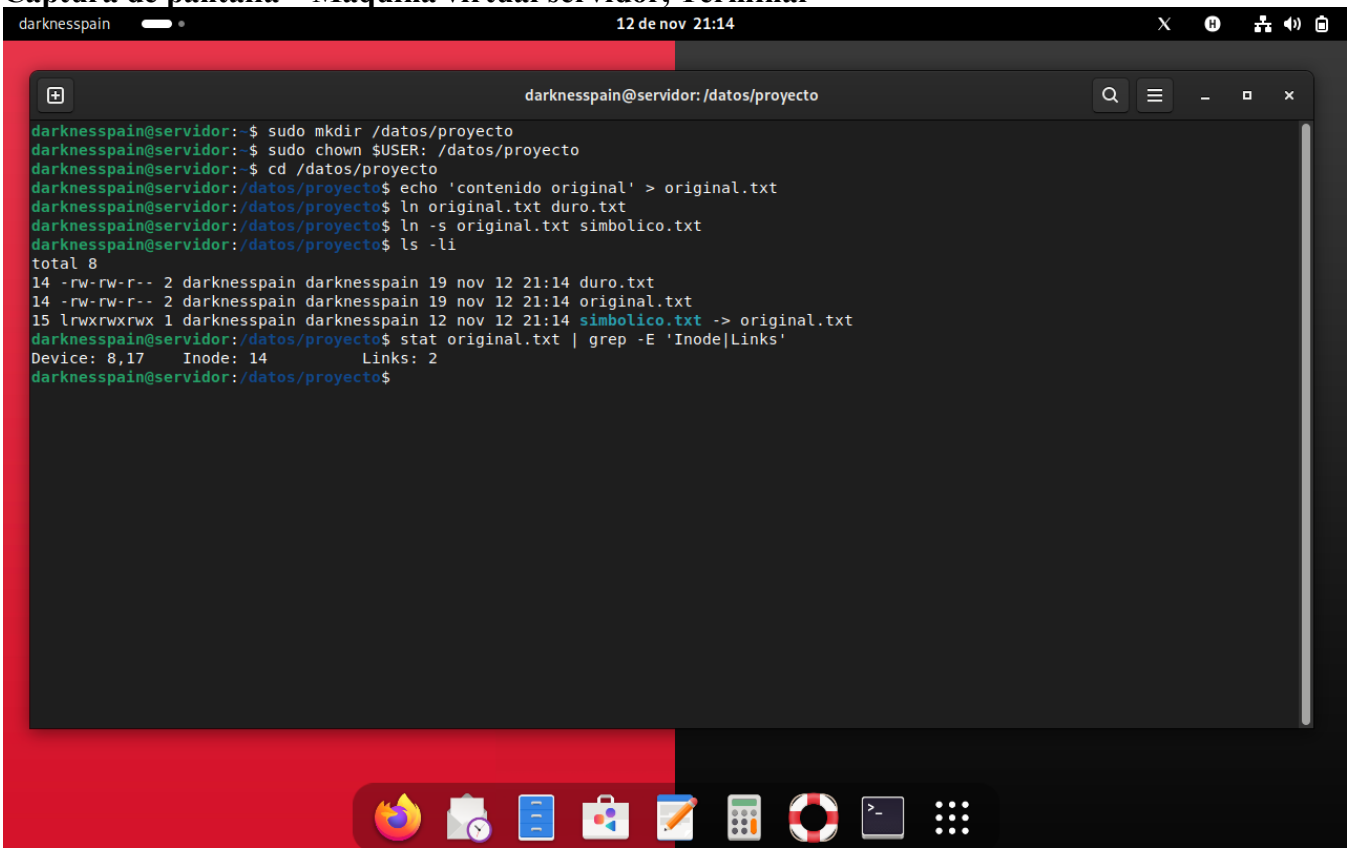
Figura 120
Captura de pantalla – Máquina virtual servidor, Terminal



Nota. «ls -li» muestra original.txt y duro.txt con el mismo inodo (14) y dos enlaces, mientras simbolico.txt tiene otro inodo (15), permisos lrwxrwxrwx y la flecha hacia original.txt.

La columna de la izquierda es el número de inodo, y es la prueba visual de que un enlace duro no es una copia sino un segundo nombre para el mismo inodo. Los tres nombres viven en el directorio, pero solo dos apuntan al mismo objeto: el enlace simbólico crea un inodo nuevo de tipo enlace que guarda una ruta. Por eso los permisos del enlace simbólico son rwxrwxrwx: en Linux los permisos de un symlink no se usan, mandan los del archivo al que apunta.

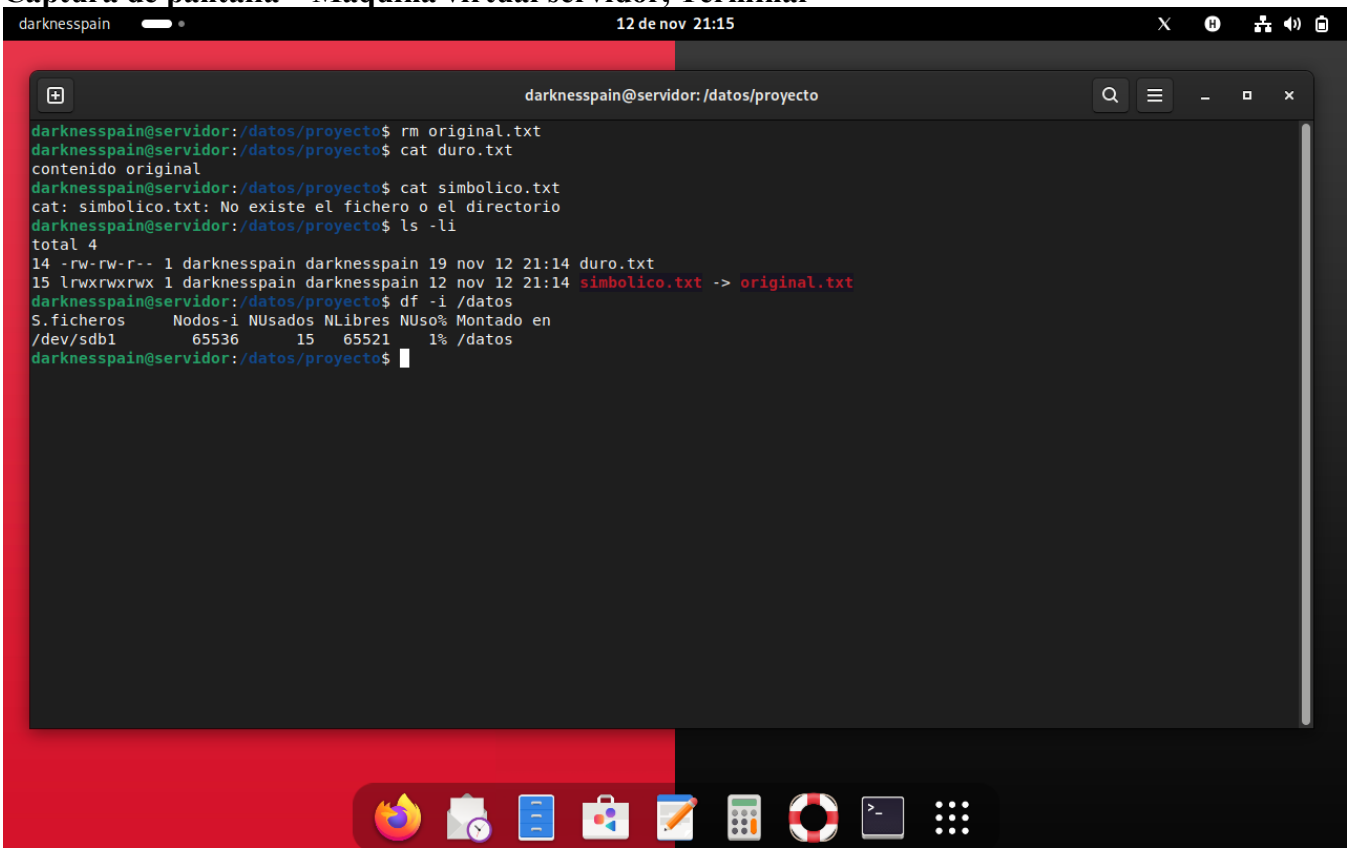
Figura 121
Captura de pantalla – Máquina virtual servidor, Terminal



Nota. «stat original.txt» filtrado muestra «Inode: 14» y «Links: 2».

El contador de enlaces es la pieza que explica todo el comportamiento de los enlaces duros: cada nombre nuevo suma uno y cada borrado resta uno; cuando llega a cero, el inodo se libera y los bloques quedan disponibles. Esa es la razón por la que «rm» no borra datos inmediatamente en un sistema con enlaces: solo elimina una entrada del directorio. En un sistema de archivos con journal, además, la liberación se registra en el diario para poder recuperar el estado tras un corte de energía.

Figura 122
Captura de pantalla – Máquina virtual servidor, Terminal



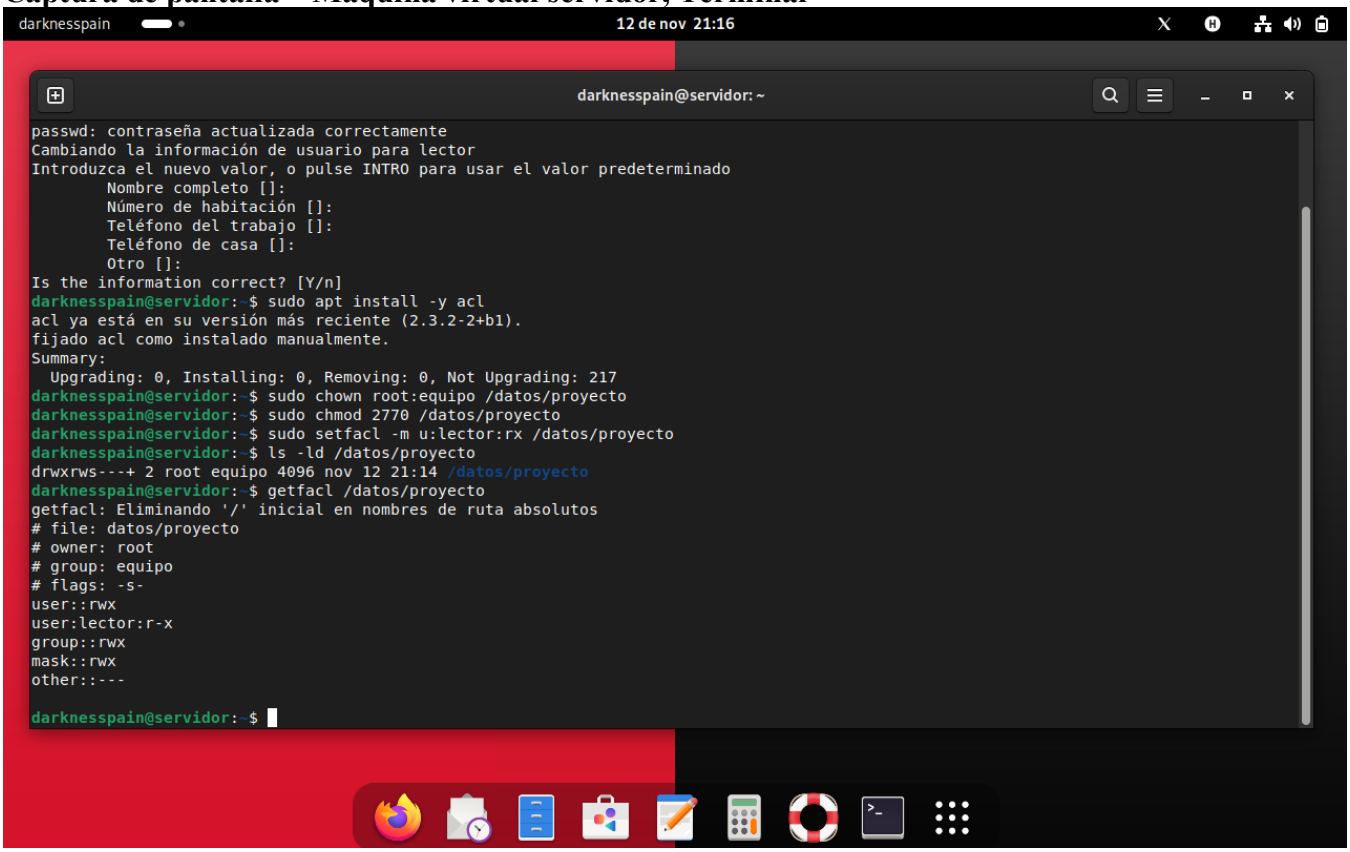
Nota. Tras «rm original.txt», «cat duro.txt» sigue mostrando «contenido original», mientras que «cat simbolico.txt» falla con «No such file or directory». «ls -li» y «df -i /datos» muestran 65536 inodos totales con 15 usados.

El resultado confirma la predicción: el enlace duro sobrevive porque el inodo sigue vivo con un enlace, y el simbólico se rompe porque apunta a un nombre que ya no existe. «df -i» añade la información que «df -h» no da: cuántos inodos quedan libres. En este disco hay 65536 y solo 15 en uso, pero si se crearan cientos de miles de archivos vacíos el espacio seguiría sobrando y aun así no se podría crear ninguno más: ese es el segundo incidente de la Parte F.

B.4 Protección: propietarios, grupos, permisos y ACL

Creamos el grupo equipo, añadimos a nuestro usuario, creamos el usuario lector, instalamos las herramientas de ACL y dejamos /datos/proyecto en modo 2770 con una ACL de lectura para lector. Antes de probar escribimos la tabla de predicción.

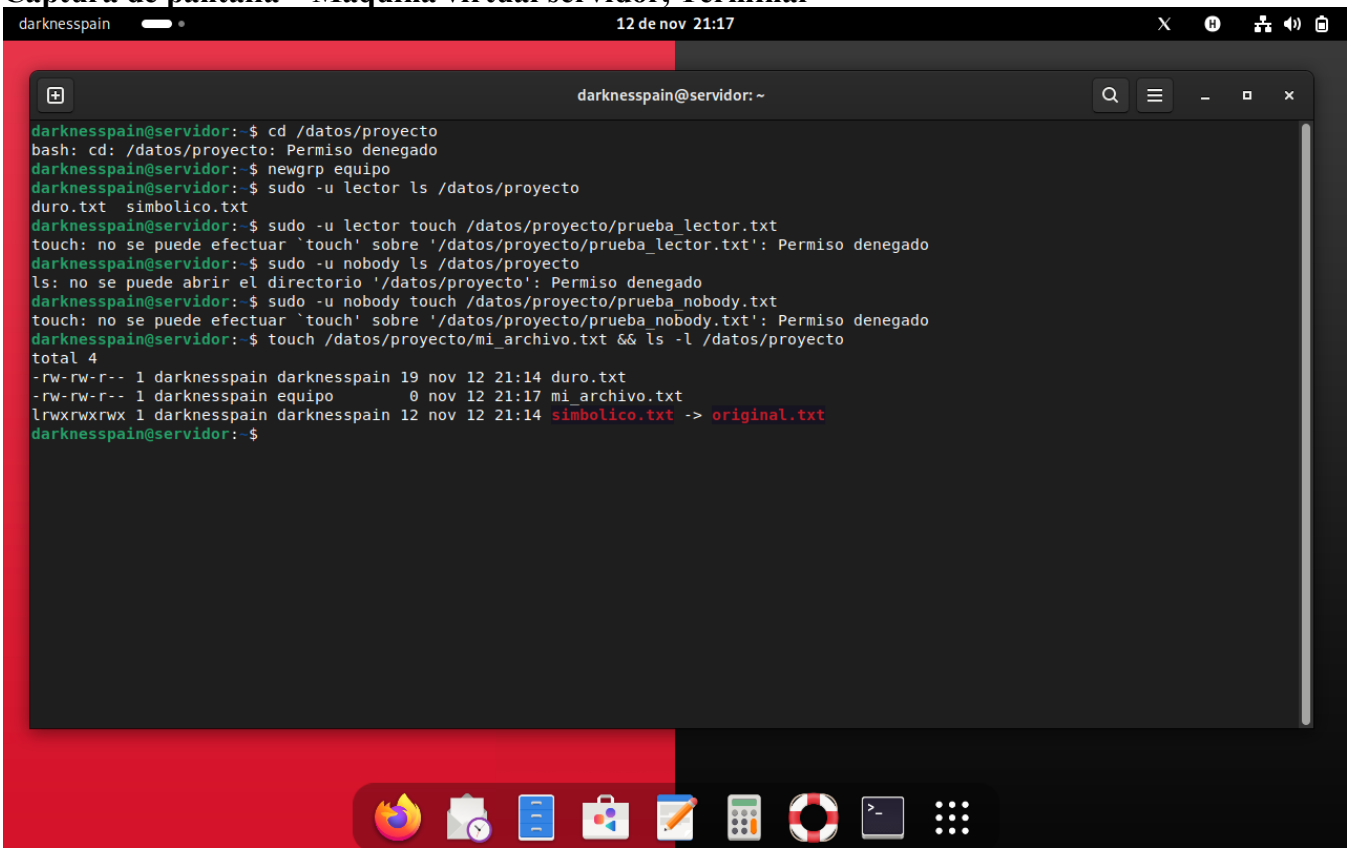
Figura 123
Captura de pantalla – Máquina virtual servidor, Terminal



Nota. Se ve la creación del usuario lector con adduser (contraseña y datos por defecto), la instalación de acl, y al final «ls -ld /datos/proyecto» con «drwxrws---+» y la salida de «getfacl» con user:lector:r-x, group::rwx y other::---.

El modo «drwxrws---+» cuenta la historia completa: el «s» en el grupo es el bit setgid, el «+» indica que hay una ACL extendida y «other» no tiene nada. El setgid hace que todo archivo creado dentro herede el grupo equipo, sin que cada usuario tenga que cambiar el grupo a mano. La ACL añade un permiso para un usuario concreto sin abrir la carpeta al resto, algo imposible con chmod, que solo distingue dueño, grupo y otros. La máscara de la ACL (mask::rwx) actúa como techo de los permisos concedidos a usuarios y grupos nombrados.

Figura 124
Captura de pantalla – Máquina virtual servidor, Terminal



Nota. «sudo -u lector ls» funciona, «sudo -u lector touch» es rechazado, «sudo -u nobody ls» también es rechazado y, tras «newgrp equipo», el archivo mi_archivo.txt se crea con el grupo equipo.

Las cuatro pruebas validan el diseño: lector puede entrar y listar por la ACL, pero no escribir porque su permiso es r-x; nobody no puede ni listar porque «otros» está en cero; y nuestro usuario, una vez activado el grupo con newgrp, crea archivos que heredan el grupo equipo gracias al setgid. Ese detalle del grupo activo es la causa habitual de los «Permiso denegado» que aparecen justo después de un «usermod -aG»: el grupo se asigna a la sesión nueva, no a la que ya estaba abierta.

Tabla 6
Predicción y resultado de los accesos a /datos/proyecto

Acción en /datos/proyecto	Predicción lector	Resultado lector	Predicción nobody	Resultado nobody
Listar la carpeta (ls)	SÍ	SÍ	NO	NO
Crear un archivo (touch)	NO	NO (Permiso denegado)	NO	NO (Permiso denegado)

Nota. La ACL concede r-x a lector; «otros» queda sin permisos, por eso nobody no puede ni listar. El setgid no otorga permisos: solo hereda el grupo.

- **Pregunta 5. ¿Por qué se usa el UUID en fstab y no el nombre /dev/sdb1?**

Porque el nombre /dev/sdX lo asigna el kernel en el orden en que detecta los discos y

puede cambiar entre arranques o al conectar hardware nuevo. En esta misma práctica lo comprobamos: antes de agregar el disco de datos, el disco del sistema era sda y la unidad óptica sr0; al añadir datos.vdi apareció sdb, y si mañana se agregara otro disco antes que él, nuestro sistema de archivos podría pasar a llamarse sdc. El UUID, en cambio, se genera al formatear y viaja con el sistema de archivos: es el mismo aunque cambie el nombre del dispositivo, el orden de los controladores o el puerto SATA. Por eso fstab lo usa y por eso la línea lleva además «nofail», que evita que un disco ausente bloquee el arranque.

- **Pregunta 6. Expliquen por qué duro.txt sobrevivió y simbolico.txt quedó «roto». ¿Qué dato entrega «df -i» que «df -h» no?**

duro.txt sobrevivió porque un enlace duro es una entrada de directorio que apunta al mismo inodo que original.txt: al borrar el nombre original, el contador de enlaces bajó de 2 a 1 y el inodo siguió existiendo con sus bloques intactos. simbolico.txt quedó roto porque guarda una ruta de texto («original.txt») y no un inodo; al desaparecer ese nombre, la ruta ya no resuelve a nada y el sistema devuelve «No such file or directory». «df -i» informa el uso de inodos —cuántos archivos caben— mientras que «df -h» informa el uso de bloques —cuánto pesan—. Son dos recursos distintos y se agotan por separado: un disco puede tener gigabytes libres y cero inodos disponibles, y en ese caso ningún archivo nuevo puede crearse. Ese dato es exactamente el que distingue los dos incidentes de disco de la Parte F.

- **Pregunta 7. En «ls -l», ¿qué grupo tiene mi_archivo.txt? ¿Por qué no es el grupo principal de su usuario? ¿Qué problema resuelven las ACL que chmod no puede resolver?**

mi_archivo.txt quedó con el grupo equipo, no con darknesspain, porque la carpeta tiene el bit setgid activo: en un directorio con setgid, los archivos nuevos heredan el grupo del directorio en lugar del grupo principal de quien los crea. Eso evita que los documentos del equipo queden repartidos entre grupos personales y permite que todos los miembros trabajen sobre ellos. Las ACL resuelven un problema que chmod no puede: dar permisos a un usuario concreto sin tocar los del grupo ni los del resto. En nuestro caso, lector necesitaba leer

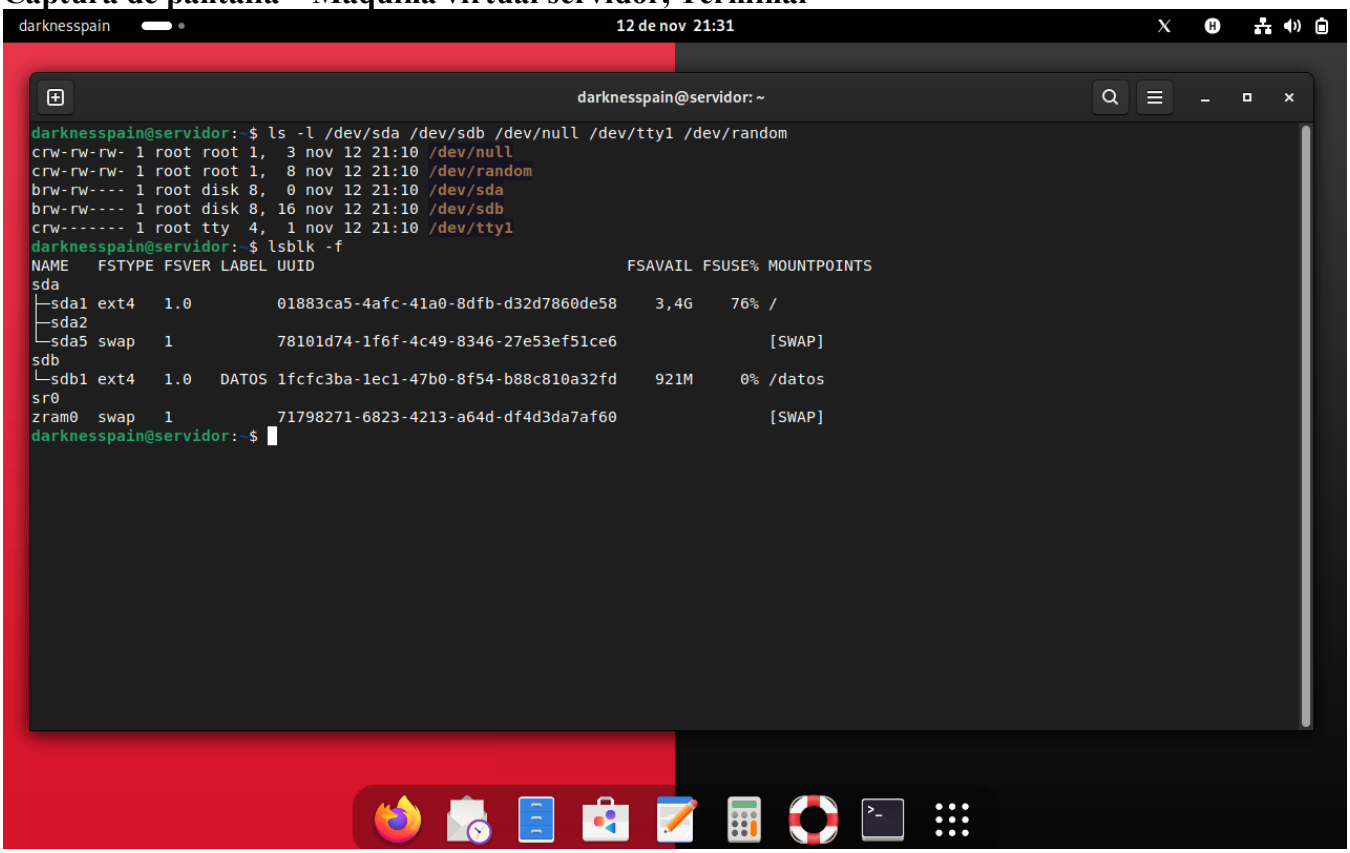
/datos/proyecto pero no escribir, y no pertenece al grupo equipo; con chmod la única alternativa habría sido abrir la carpeta a «otros», con lo que cualquier usuario del sistema habría podido entrar. La ACL permitió conceder r-x solo a lector y dejar «otros» en cero.

Parte C — Dispositivos

C.1 Todo es un archivo: dispositivos de bloques y de caracteres

En Linux los dispositivos no se manejan con llamadas especiales: son archivos dentro de /dev. La primera letra de «ls -l» dice de qué tipo son: «b» para los de bloques, que trabajan por bloques de tamaño fijo y permiten montarse como sistemas de archivos, y «c» para los de caracteres, que se leen y escriben byte a byte.

Figura 125
Captura de pantalla – Máquina virtual servidor, Terminal

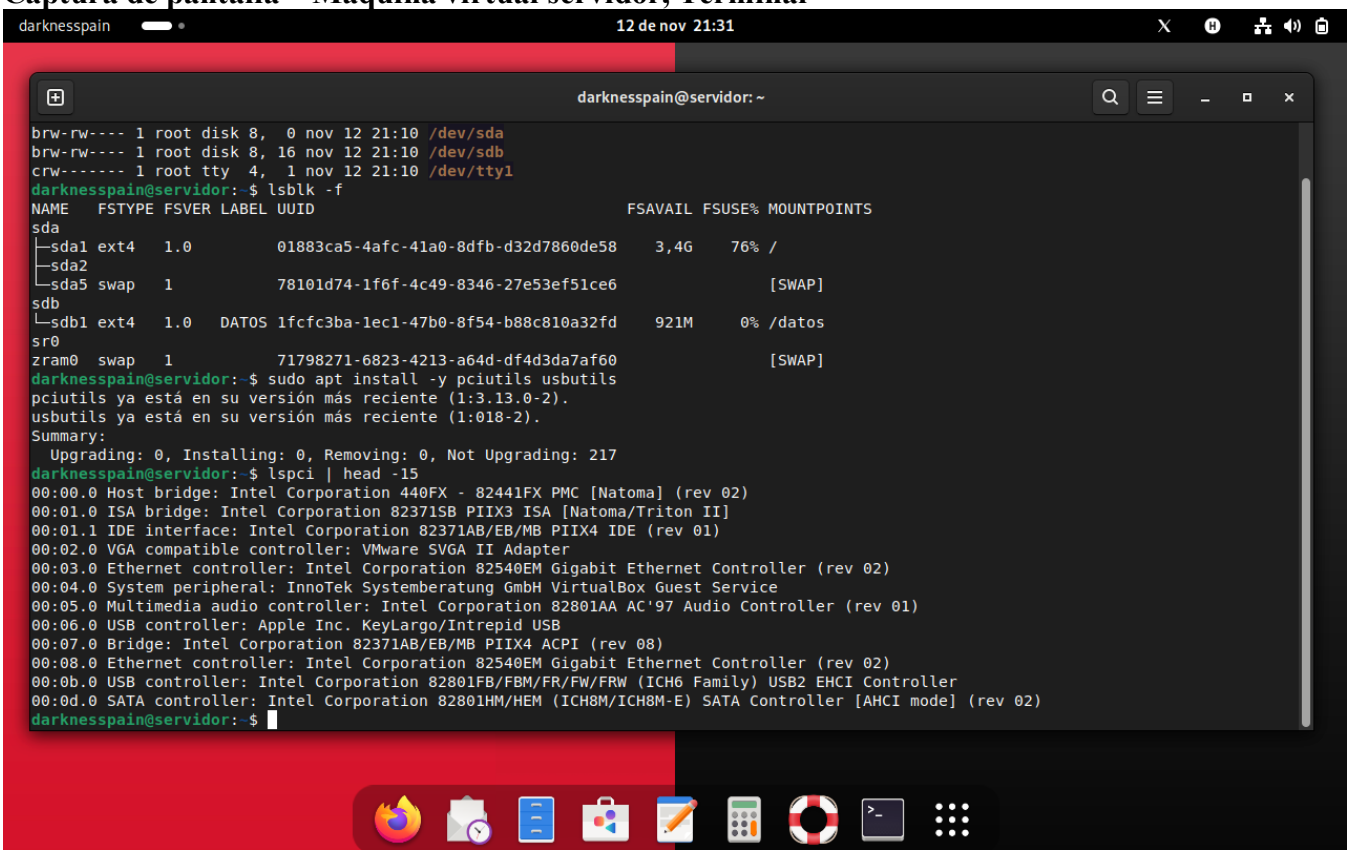


Nota. «ls -l /dev/sda /dev/sdb /dev/null /dev/tty1 /dev/random» muestra sda y sdb con «b» (brw-rw----), y null, tty1 y random con «c». Debajo, «lsblk -f» lista sda1, sdb1 con etiqueta DATOS y la partición de swap.

La letra inicial y el par mayor/menor de cada línea son la dirección del dispositivo: el número mayor identifica al controlador y el menor al dispositivo concreto dentro de él. Por eso

/dev/sda y /dev/sdb empiezan por «b»: el kernel les permite montar un sistema de archivos y acceder con caché de bloques. /dev/null y /dev/random son de caracteres porque no tienen estructura interna: uno descarta lo que se le escribe y el otro devuelve bytes aleatorios. Que ambos sean «archivos» es lo que permite usar «> /dev/null» o leer /dev/random con las mismas herramientas que cualquier otro archivo.

Figura 126
Captura de pantalla – Máquina virtual servidor, Terminal

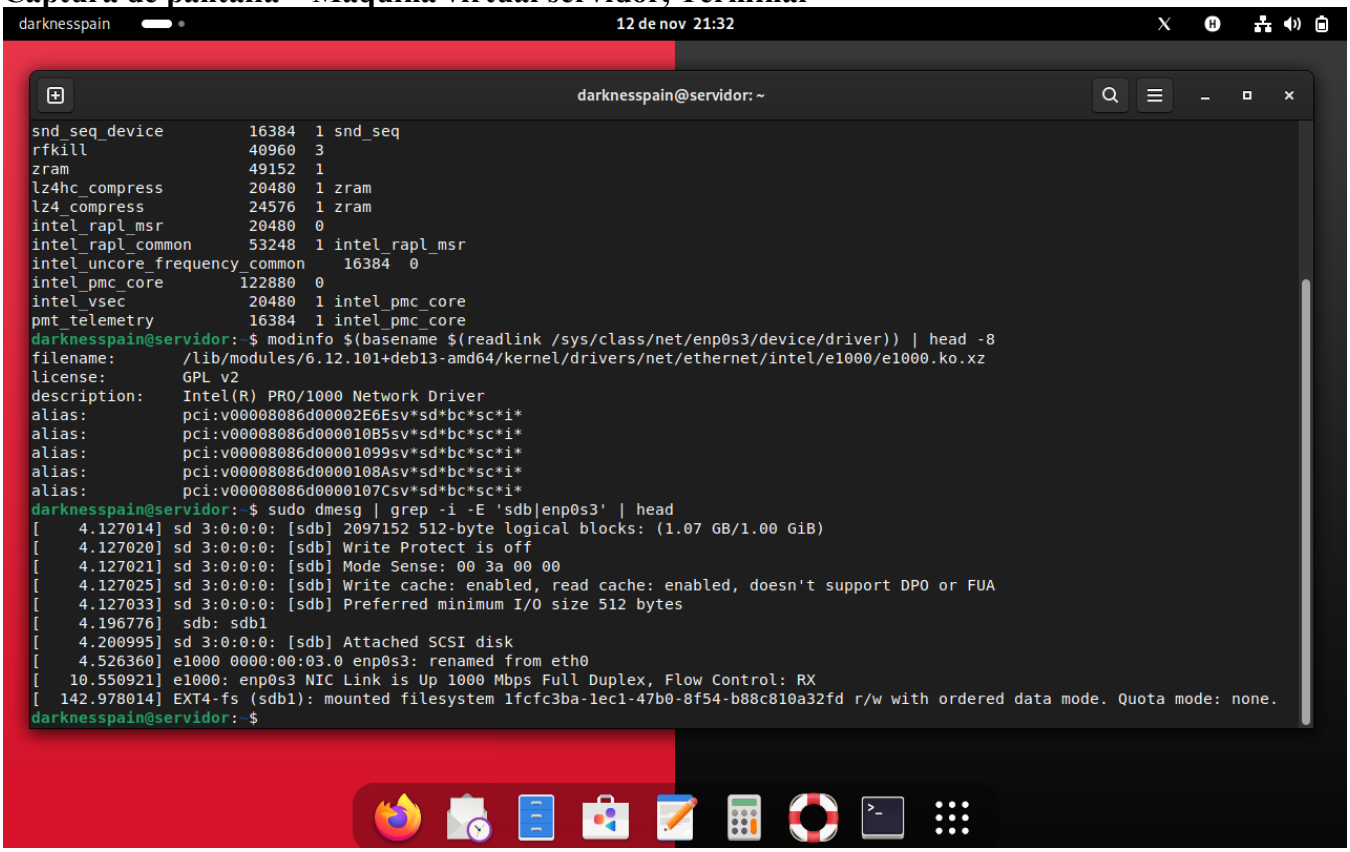


Nota. «lspci | head -15» lista el puente del sistema, el controlador VGA, el controlador SATA y dos «Ethernet controller: Intel Corporation 82540EM Gigabit Ethernet Controller» en 00:03.0 y 00:08.0.

lspci muestra el hardware tal como está colgado del bus PCI: cada línea es una función con su dirección de bus, dispositivo y función (00:03.0). Los dos controladores de red corresponden a las dos tarjetas que agregamos en VirtualBox, la del NAT y la de la red interna; que sean del mismo modelo explica que el driver sea el mismo para ambas. Esta lista es la que permite comprobar si el sistema «ve» un dispositivo antes de pelearse con su configuración.

C.2 ¿Qué driver maneja cada cosa?

Figura 127
Captura de pantalla – Máquina virtual servidor, Terminal



Nota. «ip -br link» muestra lo, enp0s3 y enp0s8; «readlink» del driver de enp0s3 devuelve e1000; «lsmod | head -15» y «modinfo» muestran el módulo cargado, y «dmesg | grep -i 'sdb|enp0s3'» enseña las líneas de detección del disco y de la tarjeta.

El enlace simbólico /sys/class/net/enp0s3/device/driver apunta al driver que el kernel asignó a esa tarjeta: e1000, que es el módulo del emulador Intel 82540EM que VirtualBox presenta al invitado. modinfo confirma que es un módulo cargable, con su versión y sus parámetros, y lsmod muestra que está en memoria. dmesg, por su parte, es el diario del kernel: ahí queda registrado el momento en que apareció sdb y en que se levantó enp0s3, que es la primera cosa que se mira cuando un dispositivo no funciona.

- **Pregunta 8. ¿Cómo se llama el driver de su tarjeta de red y qué hace en el sistema? ¿Es un módulo que se pudo cargar/descargar o está integrado en el kernel? ¿Qué papel cumple el driver entre el programa del usuario y el hardware?**

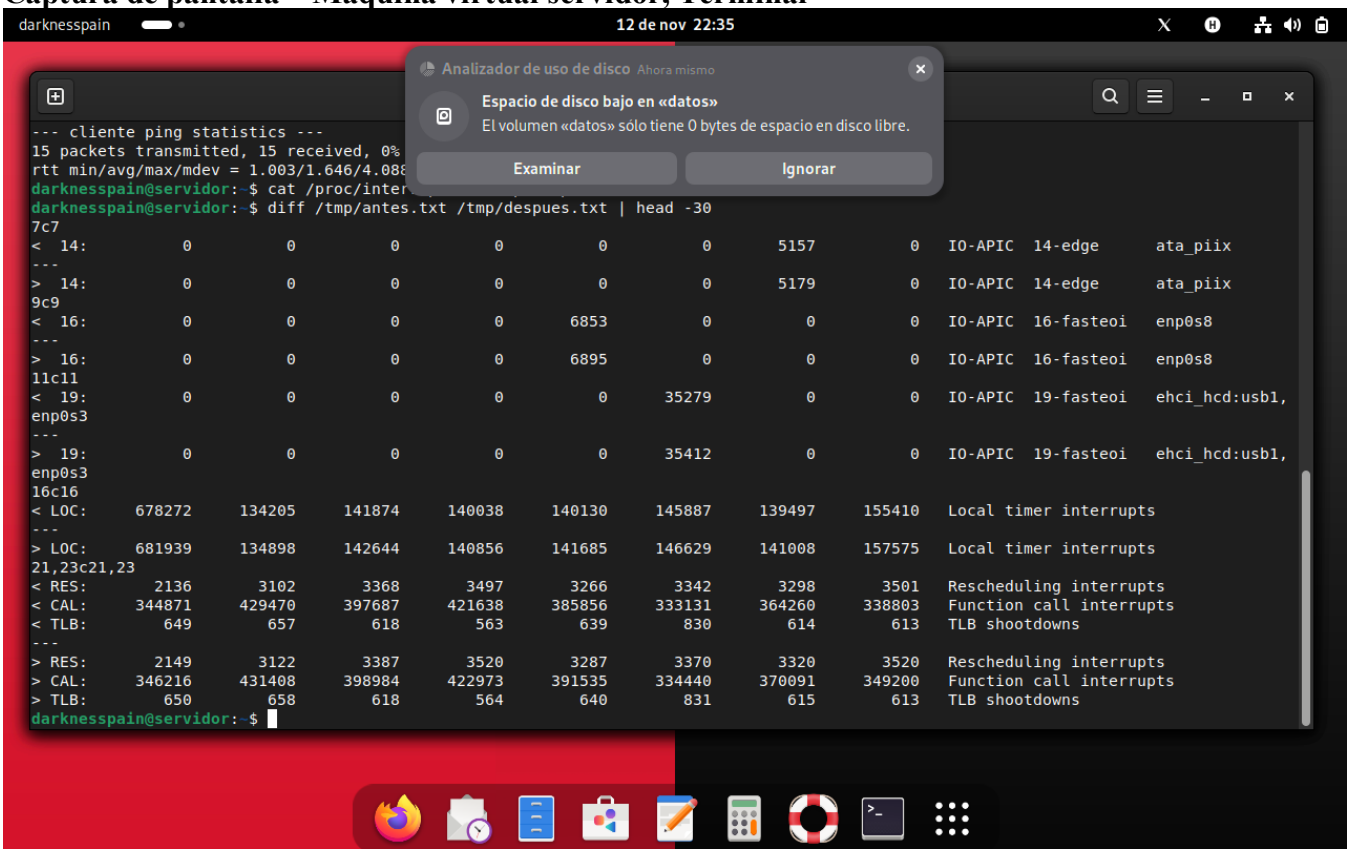
Nuestra tarjeta es una Intel 82540EM emulada por VirtualBox y su driver es el módulo e1000, que aparece en lsmod y que modinfo describe con su versión y sus parámetros, así que es un módulo cargable: se puede quitar con modprobe -r y volver a cargar, algo que sería imposible

si estuviera compilado dentro del núcleo. Su papel es traducir: el programa del usuario escribe en un socket y el kernel arma tramas Ethernet, pero quien mueve los registros físicos de la tarjeta es el driver. Sin él, el sistema vería el dispositivo en lspci pero no tendría forma de enviar ni recibir un solo paquete, y «ip link» mostraría la interfaz sin estado operativo.

C.3 Interrupciones: el hardware «toca el timbre»

Antes de medir escribimos la predicción en el archivo C3_interrupciones.md: esperábamos que el ping moviera la fila de la tarjeta de red, que dd moviera la del controlador de disco y que el teclado moviera la del controlador PS/2, con el mayor aumento absoluto en el caso de dd.

Figura 128
Captura de pantalla – Máquina virtual servidor, Terminal

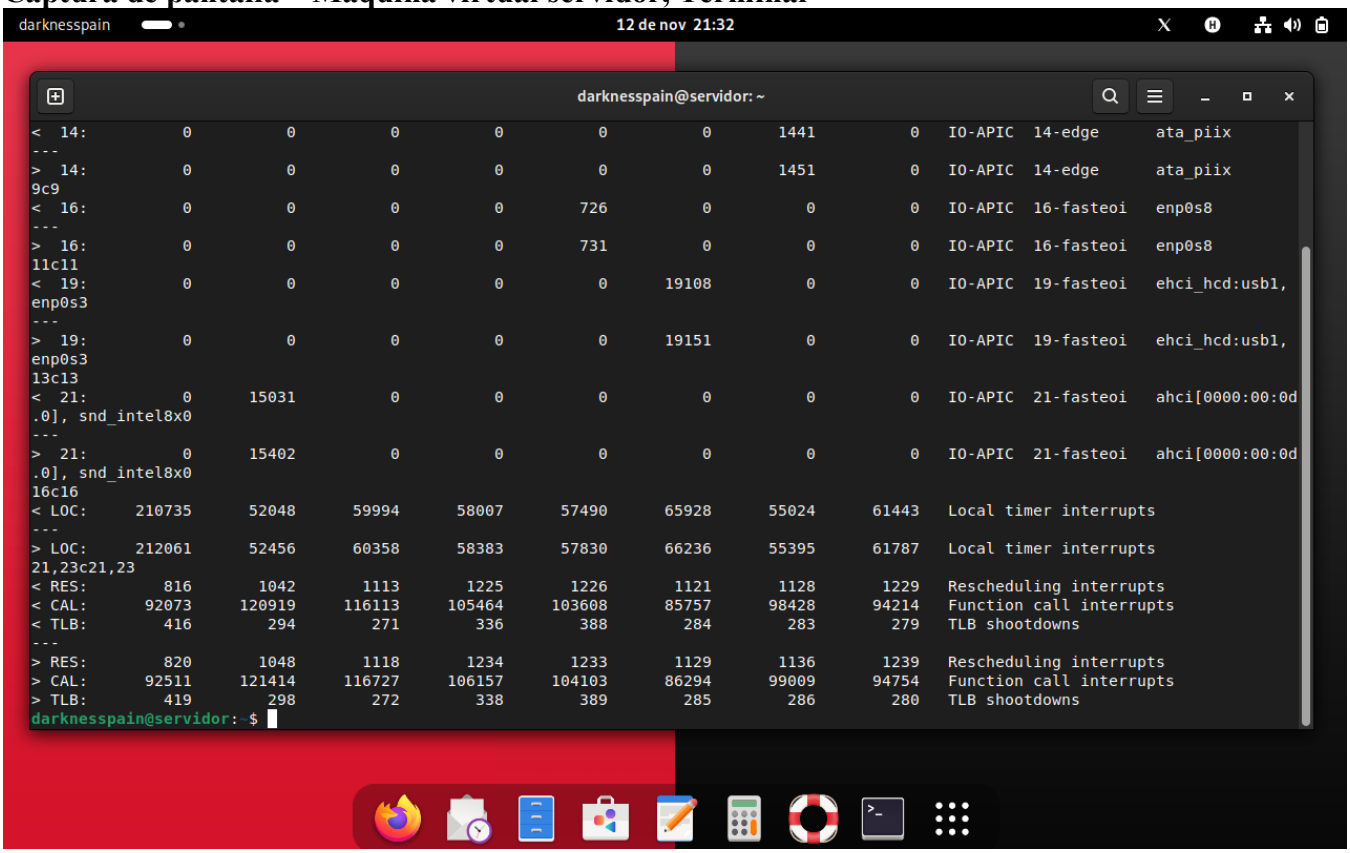


Nota. Tras «ping -c 15 cliente» (15 paquetes transmitidos y 15 recibidos, 0 % de pérdida) el «diff» de /proc/interrupts muestra aumentos en las filas de enp0s3 y enp0s8.

Cada paquete que entra o sale genera una interrupción del controlador, y el contador de /proc/interrupts las acumula por línea. El ping movió las filas de las dos interfaces porque el

tráfico salió por enp0s8 y la respuesta también entró por ella, mientras enp0s3 registró algo de ruido de la red NAT. Quince paquetes producen quince eventos: pocos, pero suficientes para ver el cambio. Comparado con el disco, es un dispositivo de baja frecuencia de interrupciones.

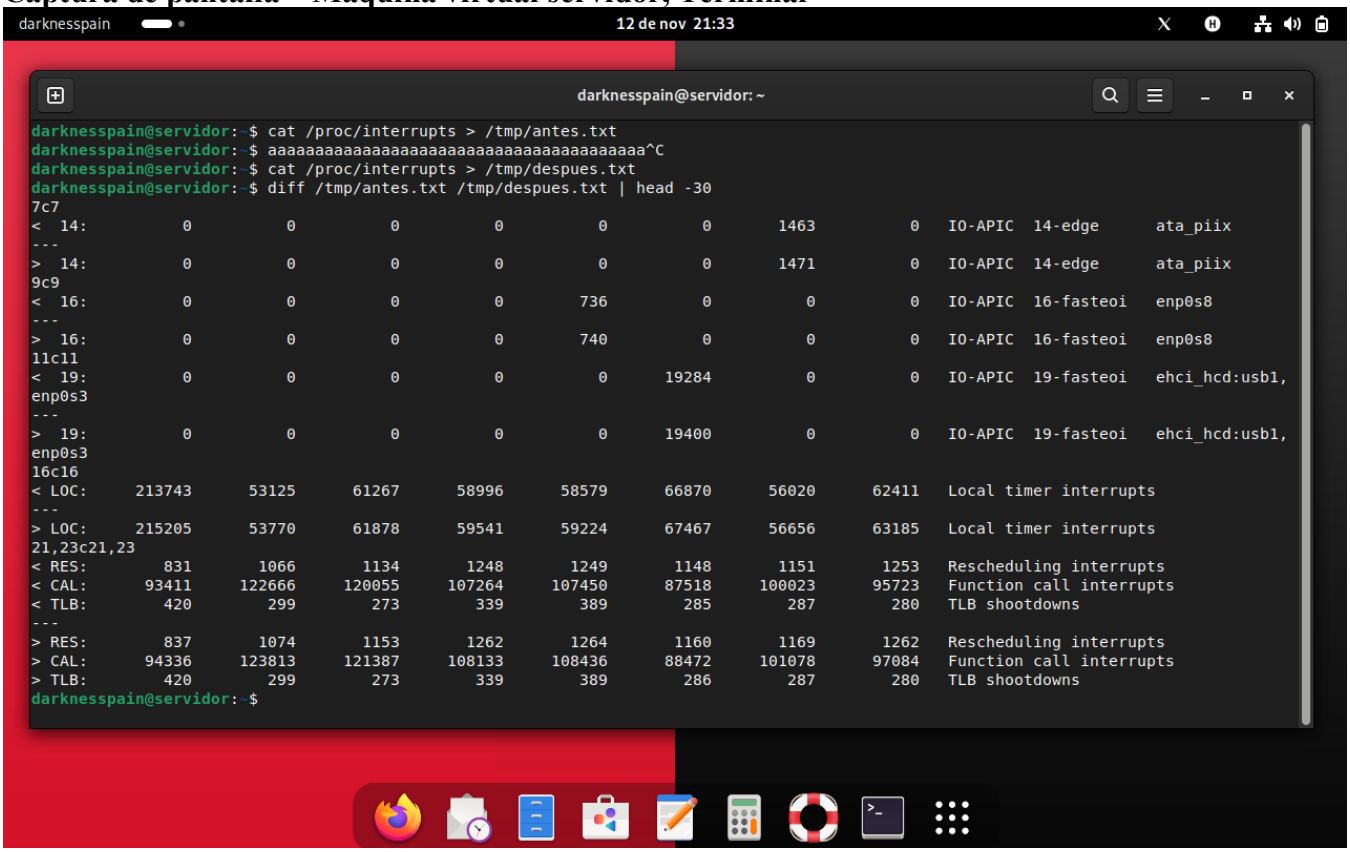
Figura 129
Captura de pantalla – Máquina virtual servidor, Terminal



Nota. «dd if=/dev/sda of=/dev/null bs=1M count=300» copia 300 MB en 0,3 s y el diff de /proc/interrupts muestra las filas de «ata-piix» con incrementos de más de mil interrupciones.

Aquí se ve la diferencia de escala: mover 300 MB obliga a miles de transferencias y cada una termina con una interrupción del controlador SATA. Por eso la fila de «ata-piix» creció en más de mil, mientras el ping había movido decenas. Ese es el motivo por el que los discos y las tarjetas de red de altas prestaciones usan interrupciones agrupadas (una interrupción por lote de paquetes o de bloques) en lugar de una por evento.

Figura 130
Captura de pantalla – Máquina virtual servidor, Terminal



Nota. Se pulsaron 40 teclas con «a» y después se comparó /proc/interrupts: el diff vuelve a mostrar las filas de ata y de enp0s8 con los mismos contadores que en la medición anterior, y no aparece un aumento apreciable en la fila del teclado.

El resultado no coincidió del todo con la predicción: esperábamos ver crecer la fila del controlador PS/2 (i8042) y el diff no la destacó. La explicación es que el teclado de la máquina virtual no pasa por ese controlador: VirtualBox inyecta las pulsaciones como eventos del dispositivo USB/PS2 emulado y el terminal las recibe por la sesión gráfica, de modo que el contador que cambia es otro. Lo dejamos escrito tal como salió, porque ese es justamente el aprendizaje: el mapa de interrupciones depende de cómo el hipervisor presenta el hardware, no solo del sistema operativo.

Tabla 7
Interrupciones observadas por actividad

Actividad	Fila que más aumentó	¿Coincide con la predicción?
ping -c 15 cliente	enp0s3 y enp0s8 (red)	Sí: el tráfico lo mueve la tarjeta
dd if=/dev/sda (300 MB)	ata-piix (controlador SATA), más de mil	Sí: es el mayor aumento absoluto
Escribir 40 teclas	No se apreció cambio en i8042; el diff repite ata y enp0s8	No: el teclado emulado no usa ese contador

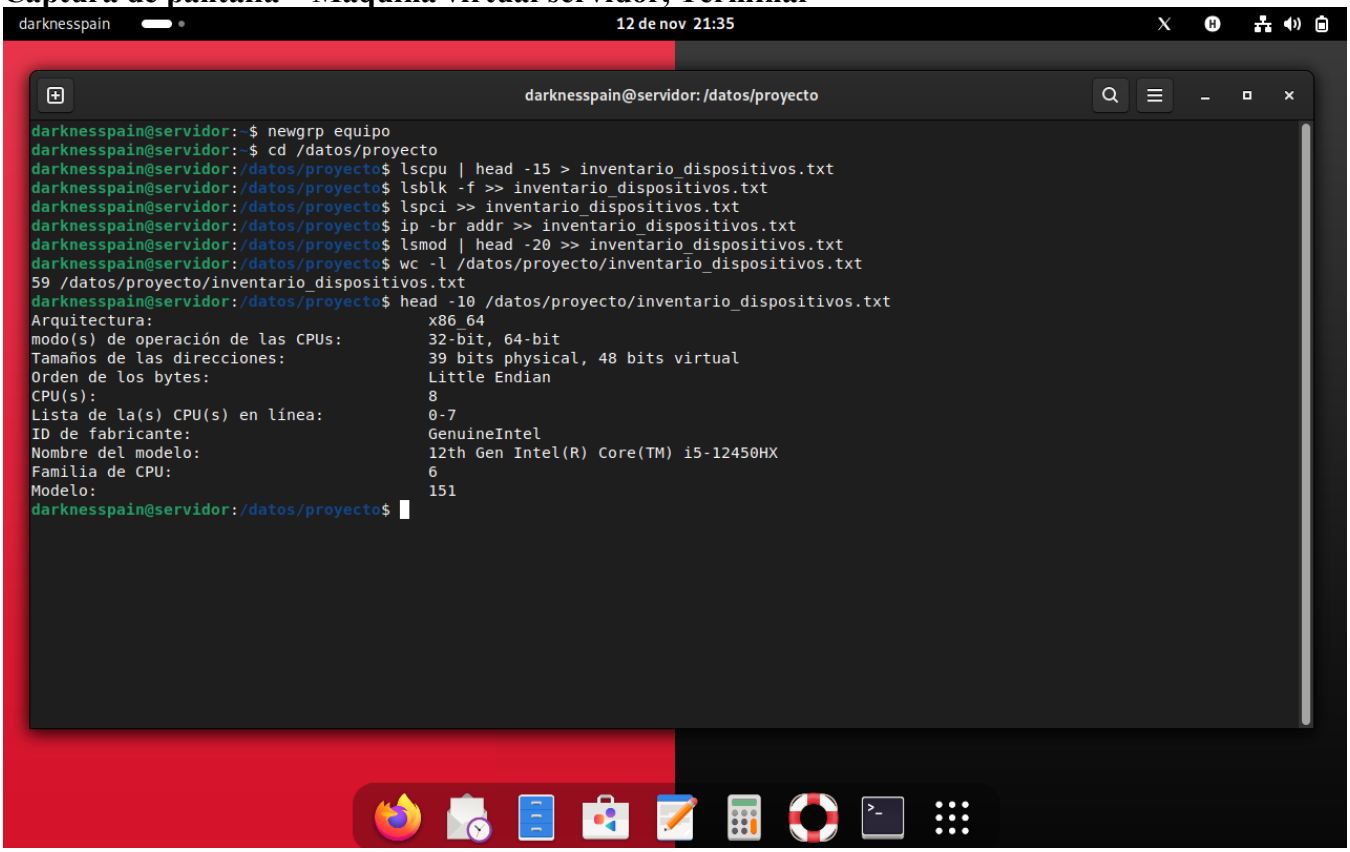
Nota. Los valores salen de los diff de las Figuras 129, 130 y 131 sobre /proc/interrupts.

- **Pregunta 9. Expliquen con un ejemplo cotidiano la diferencia entre encuesta (polling) e interrupciones. ¿Cuándo preferirían encuesta?**

La encuesta es preguntar una y otra vez «¿hay algo?» aunque no haya nada; la interrupción es que el dispositivo avise cuando ocurre algo. En la vida diaria, encuesta es mirar la puerta cada cinco segundos para ver si llegó el mensajero, e interrupción es que suene el timbre. La encuesta desperdicia tiempo de CPU en preguntas que casi siempre tienen respuesta negativa, y las interrupciones gastan tiempo en cambiar de contexto por cada evento. Por eso se prefiere encuesta cuando los eventos son muchísimo más frecuentes que el coste de preguntar: una tarjeta de 10 Gb/s que recibe miles de paquetes por segundo produce más interrupciones que trabajo útil, y entonces el driver pasa a leer en lotes (NAPI). Es exactamente el mismo equilibrio que vimos entre el ping y el dd.

C.4 Inventario del servidor

Figura 131
Captura de pantalla – Máquina virtual servidor, Terminal



Nota. Tras entrar a /datos/proyecto con el grupo equipo activo, se volcaron lscpu, lsblk -f, lspci, ip -br addr y lsmod en inventario_dispositivos.txt. «wc -l» informa 59 líneas y «head» muestra el comienzo del archivo.

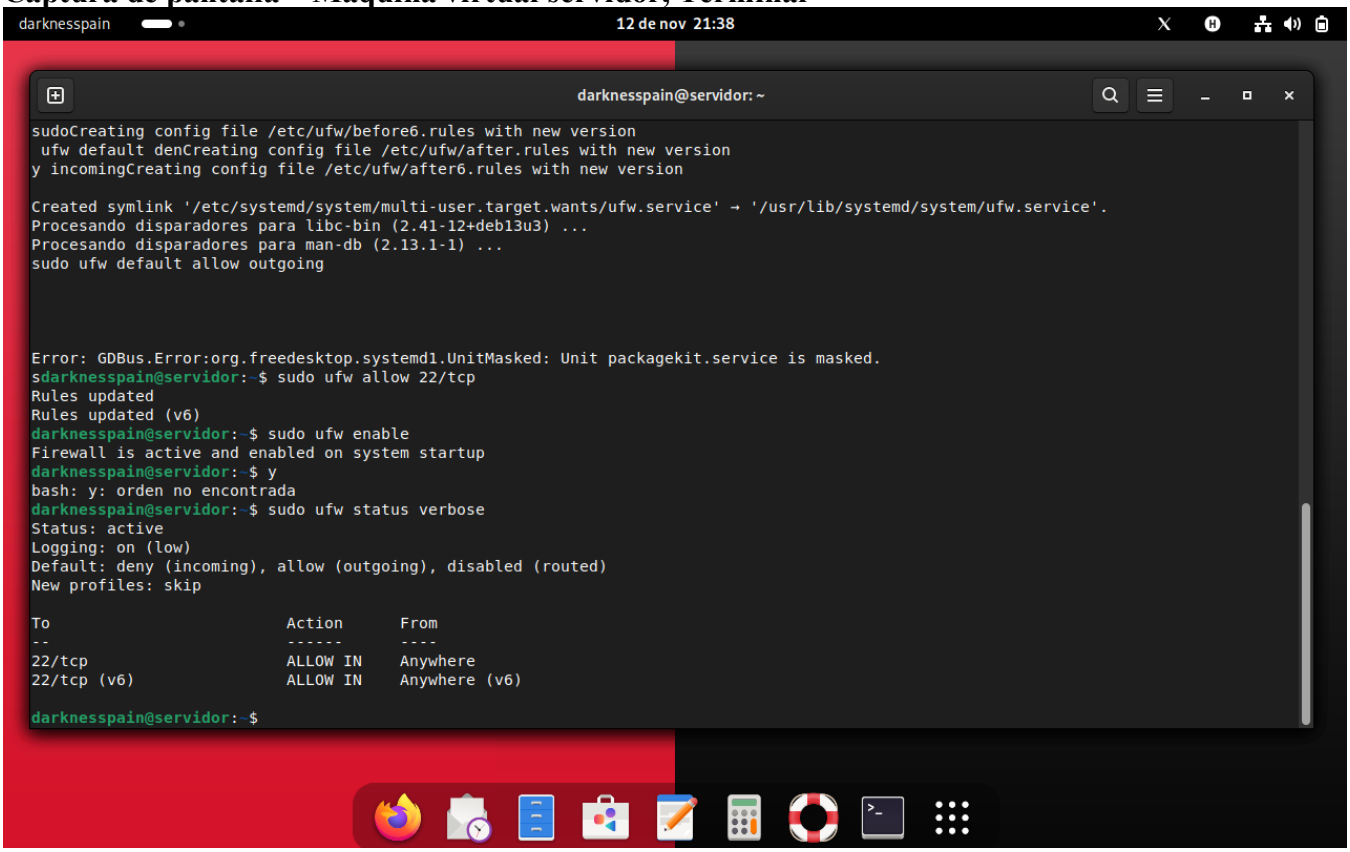
El inventario es la fotografía del hardware del servidor y el verificador lo exige como parte de la documentación. Sirve para dos cosas: responder rápido qué CPU, qué discos y qué tarjetas tiene la máquina cuando algo falla, y detectar cambios (un disco que desaparece, una tarjeta que ya no aparece en lspci). Guardarlo dentro de /datos tiene además una ventaja práctica: entra en el respaldo automático sin hacer nada más.

Parte D — Seguridad y administración

D.1 Cortafuegos con ufw

Instalamos ufw y aplicamos la política por defecto: negar lo que entra, permitir lo que sale. El orden importa: primero se permite el puerto 22 y solo después se activa el cortafuegos, porque activarlo sin esa regla deja al administrador fuera de su propio servidor.

Figura 132
Captura de pantalla – Máquina virtual servidor, Terminal

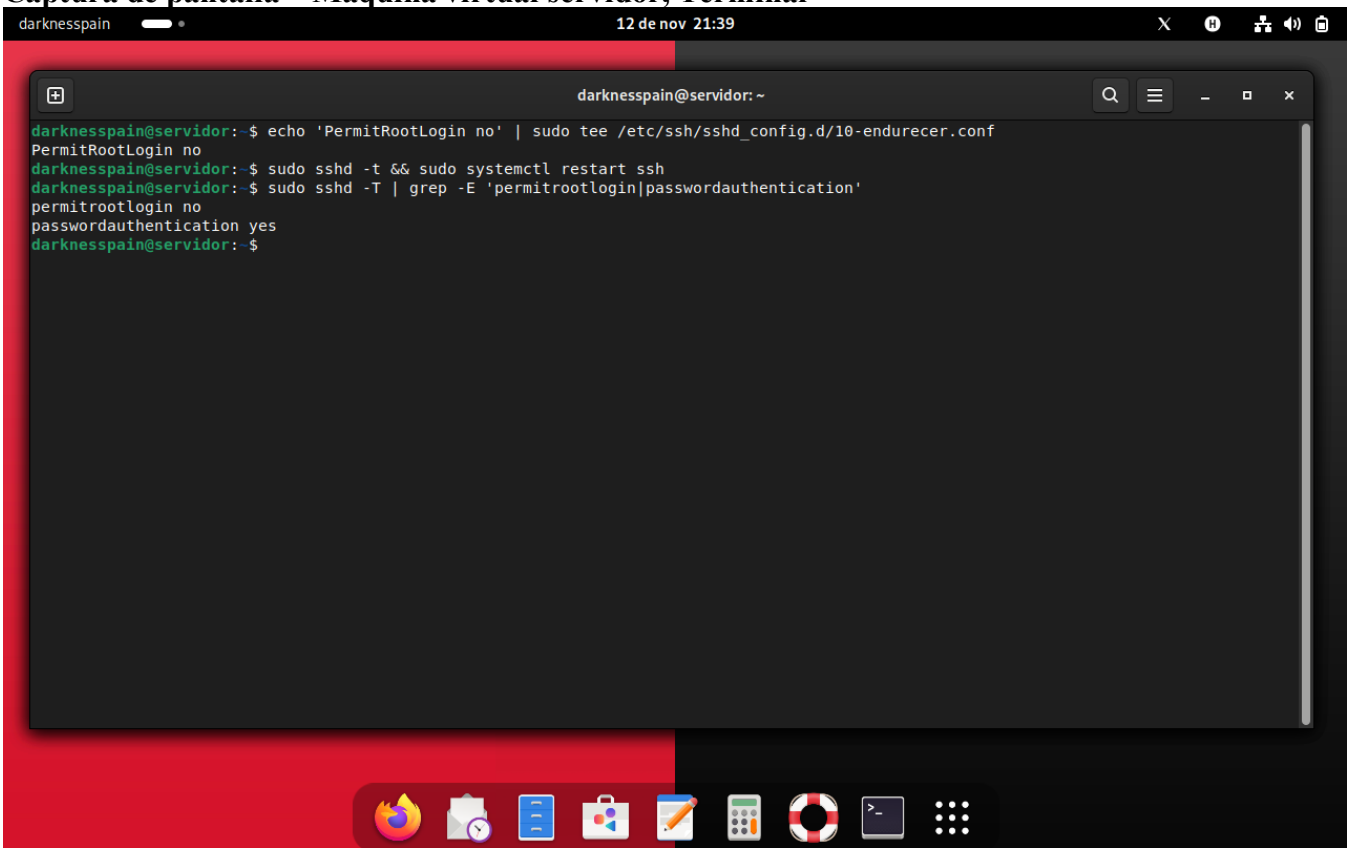


Nota. «ufw status verbose» responde «Status: active», «Default: deny (incoming), allow (outgoing)» y muestra la regla 22/tcp ALLOW IN para IPv4 y para IPv6.

La salida confirma las tres decisiones: el cortafuegos está activo, la política es denegar por defecto lo entrante y la única excepción es SSH. Ese «deny incoming» es lo que hace que el servidor no tenga superficie expuesta más allá de lo que decidimos abrir. En la Parte F veremos el otro lado de la moneda: si esa regla desaparece, el puerto se cierra sin que el servicio SSH deje de estar activo, y distinguir una cosa de la otra es la clave del diagnóstico.

D.2 SSH más seguro

Figura 133
Captura de pantalla – Máquina virtual servidor, Terminal

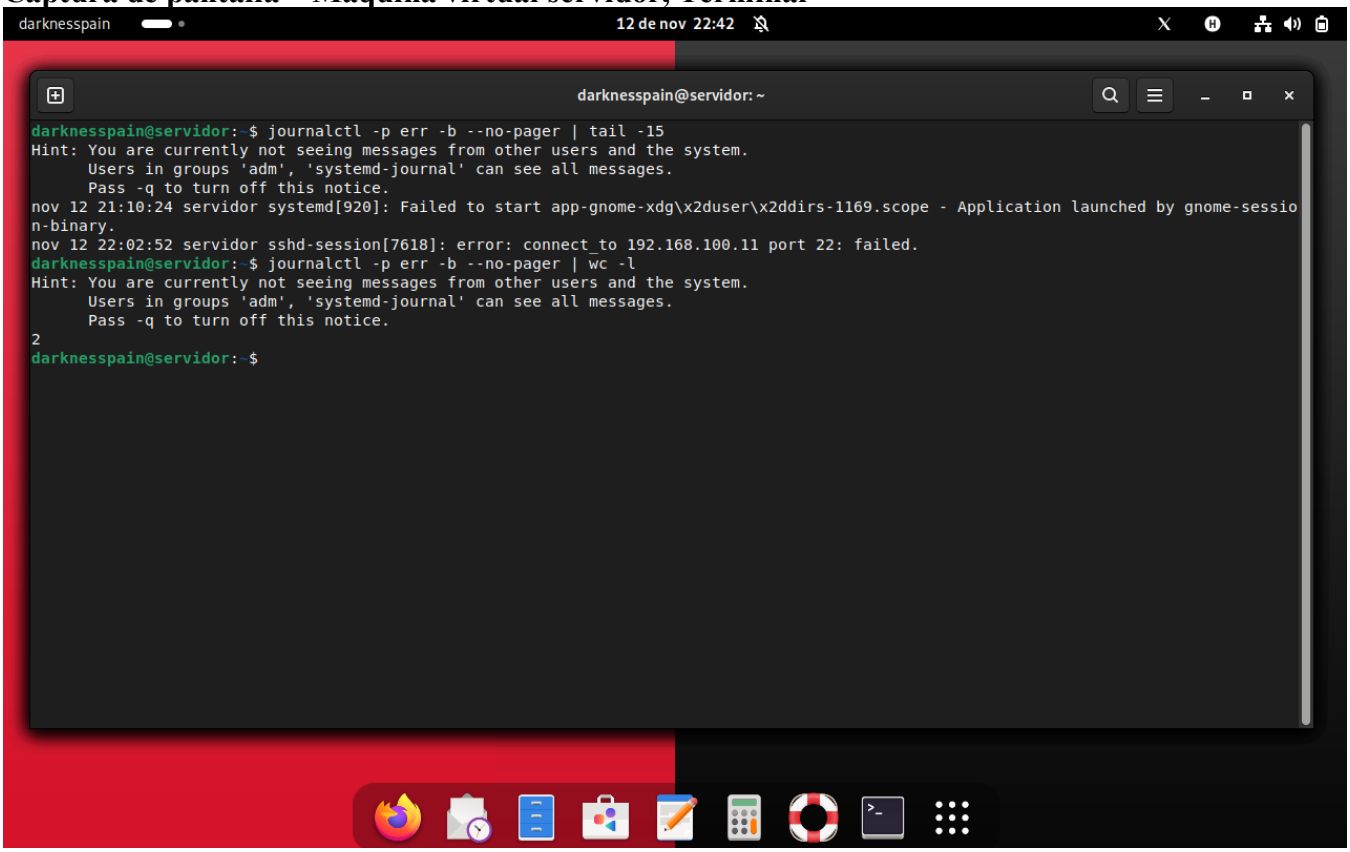


Nota. Se creó /etc/ssh/sshd_config.d/10-endurecer.conf con «PermitRootLogin no»; tras «sshd -t» y reiniciar el servicio, «sshd -T» confirma «permitrootlogin no» y todavía «passwordauthentication yes».

sshd -t valida el archivo antes de reiniciar: si la sintaxis fuera incorrecta, el servicio no arrancaría y perderíamos el acceso. Por eso se prueba primero y se reinicia después. La directiva se escribe en un archivo aparte dentro de sshd_config.d en lugar de editar sshd_config: Debian lee ese directorio al final, así que la nuestra gana sin romper el archivo original y se puede revertir borrando un solo fichero. En ese momento la contraseña seguía permitida porque todavía no había llaves; en la Parte E se cierra esa puerta.

D.3 Leer los registros (logs) del sistema

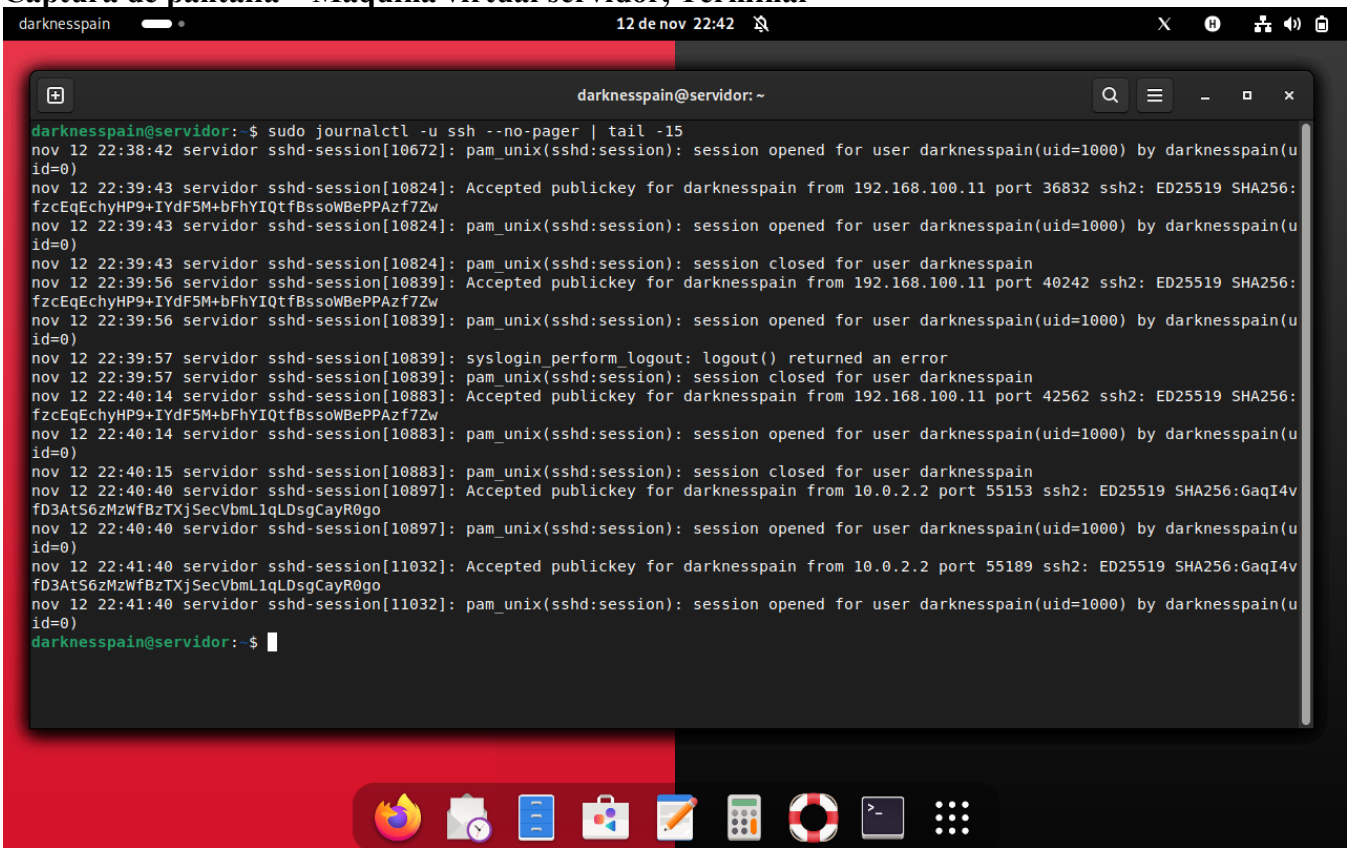
Figura 134
Captura de pantalla – Máquina virtual servidor, Terminal



Nota. «journalctl -p err -b --no-pager | tail -15» muestra el aviso de que el usuario no ve todos los mensajes y una línea de error sobre app-gnome-xdg-user-dirs; «wc -l» cuenta las líneas de error del arranque actual.

El diario del sistema es la primera fuente cuando algo falla. La opción -p err filtra solo errores y -b limita al arranque actual, que es lo que interesa en caliente. El aviso de permisos explica que un usuario normal no ve los mensajes del sistema: para eso está el grupo adm o sudo. La línea de xdg-user-dirs es un fallo conocido del escritorio GNOME en máquinas virtuales y no afecta al servidor, pero aparece porque el criterio es «todo lo que el sistema marcó como error», sin juzgar su importancia.

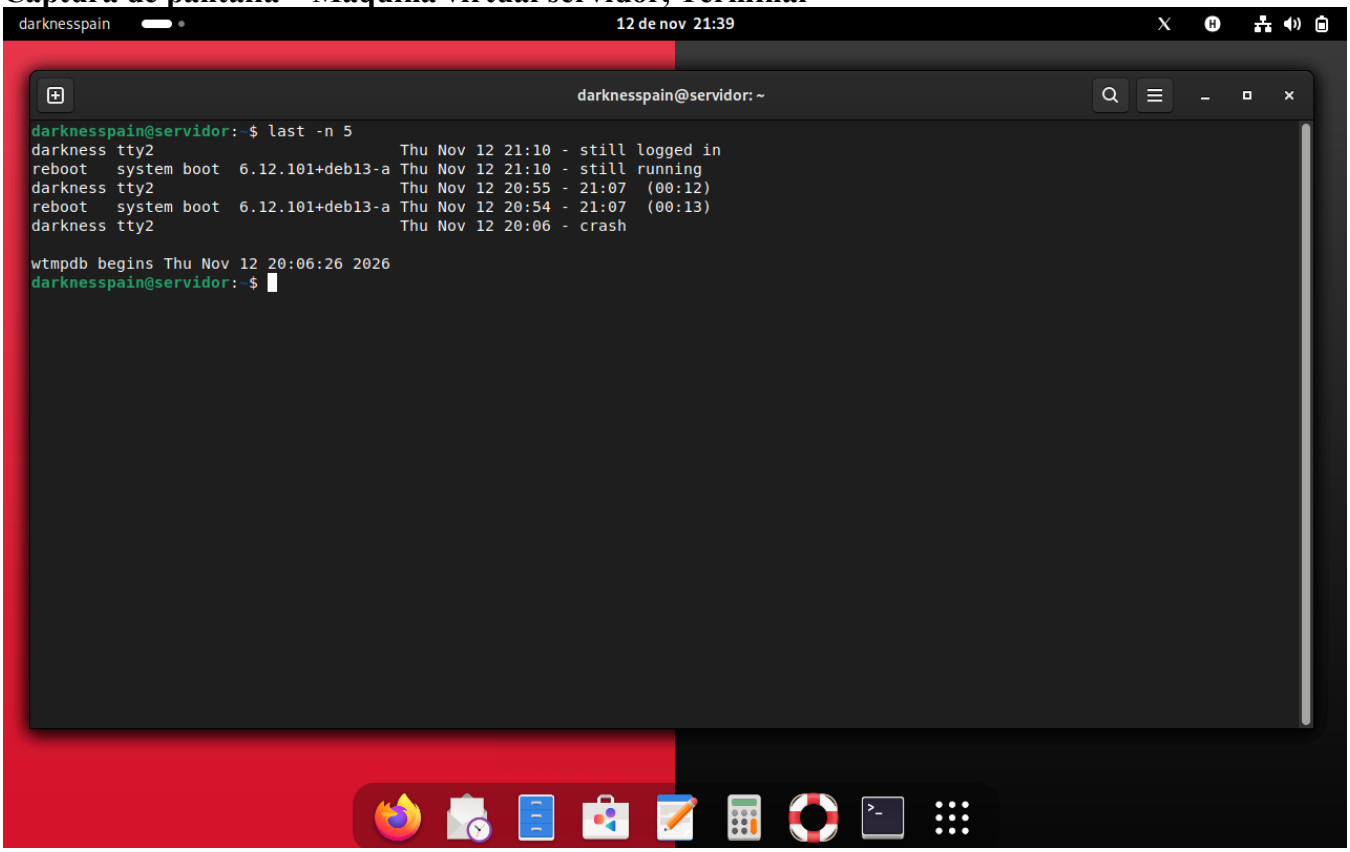
Figura 135
Captura de pantalla – Máquina virtual servidor, Terminal



Nota. «sudo journalctl -u ssh --no-pager | tail -15» muestra sesiones abiertas y la línea «Accepted publickey for darknesspain from 192.168.100.11 ... ED25519 SHA256:…».

Esta captura es la prueba de que el acceso por llave está funcionando: la línea dice explícitamente «publickey», la IP de origen es la del cliente y aparece la huella de la llave. Si en su lugar apareciera «Failed password», sabríamos que alguien está probando contraseñas y podríamos reaccionar. Leer el registro del servicio es la forma de auditar quién entra y cómo, algo que un administrador hace de rutina y que además permite detectar intentos de fuerza bruta.

Figura 136
Captura de pantalla – Máquina virtual servidor, Terminal



Nota. «last -n 5» lista los últimos ingresos: sesiones en tty2 y varios «reboot system boot» con su fecha y su duración.

last lee el registro de sesiones y las ordena de la más reciente a la más antigua. Muestra dos tipos de entrada: las sesiones de usuario (tty2, la consola gráfica) y los arranques del sistema. Sirve para reconstruir qué pasó: si la máquina se reinició sola a las 21:07 y volvió a las 21:19, hay que buscar la causa en el diario de ese arranque. Junto con journalctl forma el par básico de auditoría: uno dice qué sesiones hubo y el otro qué ocurrió dentro de ellas.

- **Pregunta 10. Escojan UNA línea de los registros y expliquen quién la escribió, qué evento reporta y para qué le serviría a un administrador.**

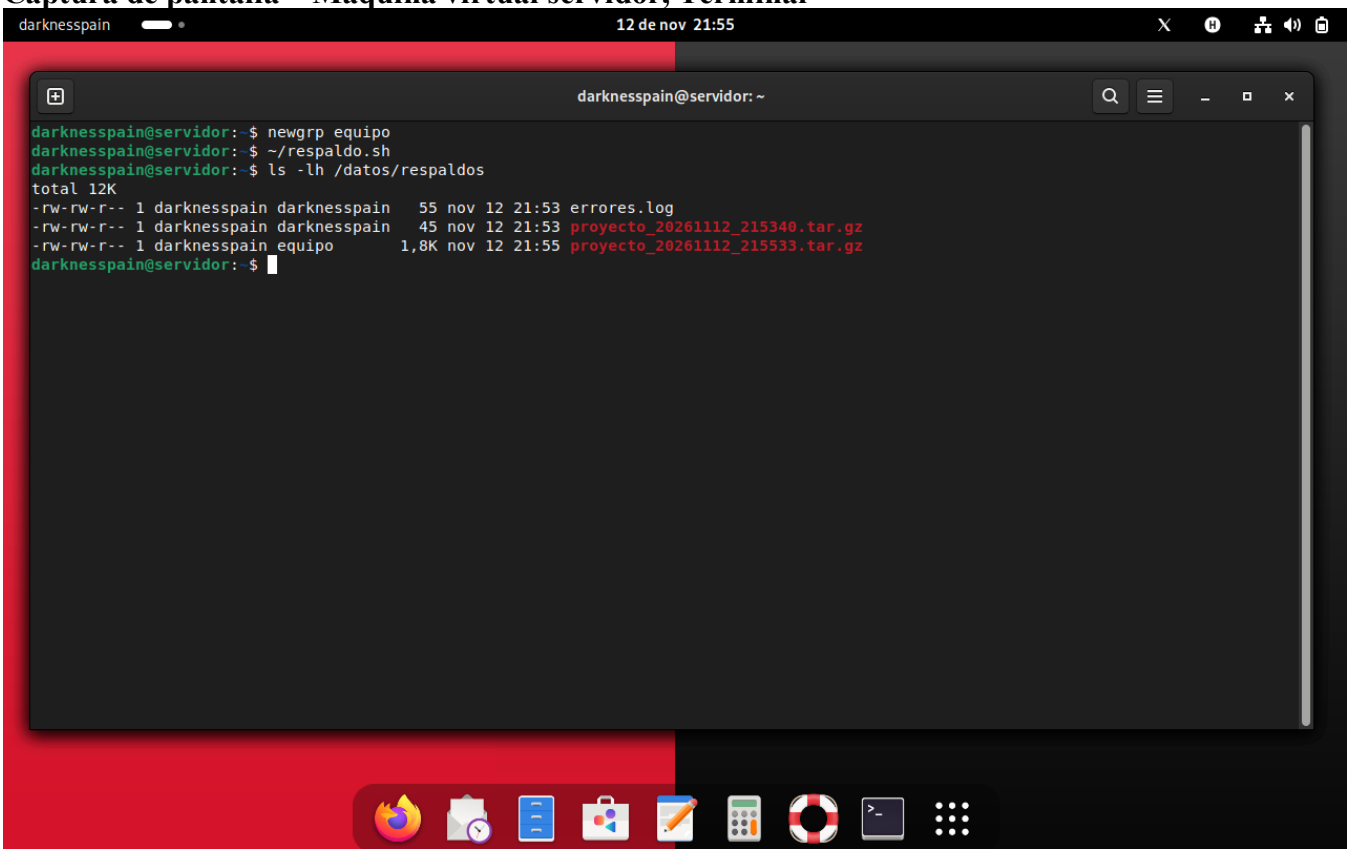
Elegimos la línea «Accepted publickey for darknesspain from 192.168.100.11 port 36832 ssh2: ED25519 SHA256:fzcEqEchYHP9+1YdF5M+bFhYIQtfBssowBePPAzf7zw». La escribió sshd, más exactamente el proceso sshd-session que atiende esa conexión, en el momento en que aceptó la autenticación. Reporta tres datos: el usuario autenticado, la dirección desde la que llegó la conexión y el tipo y huella de la llave que se usó. Para un administrador es una línea de

auditoría: le dice quién entró, desde qué máquina y con qué método, sin depender de lo que el usuario recuerde. Si en lugar de «Accepted publickey» apareciera «Failed password» repetido, sabría que alguien está probando contraseñas y podría bloquear la IP o endurecer el acceso, que es justo lo que hicimos en D.2 y E.1.

D.4 Respaldo automático con cron

Preparamos la carpeta de respaldos, copiamos el script de la docente a nuestro directorio personal y lo ejecutamos a mano antes de programarlo, para saber que funciona.

Figura 137
Captura de pantalla – Máquina virtual servidor, Terminal

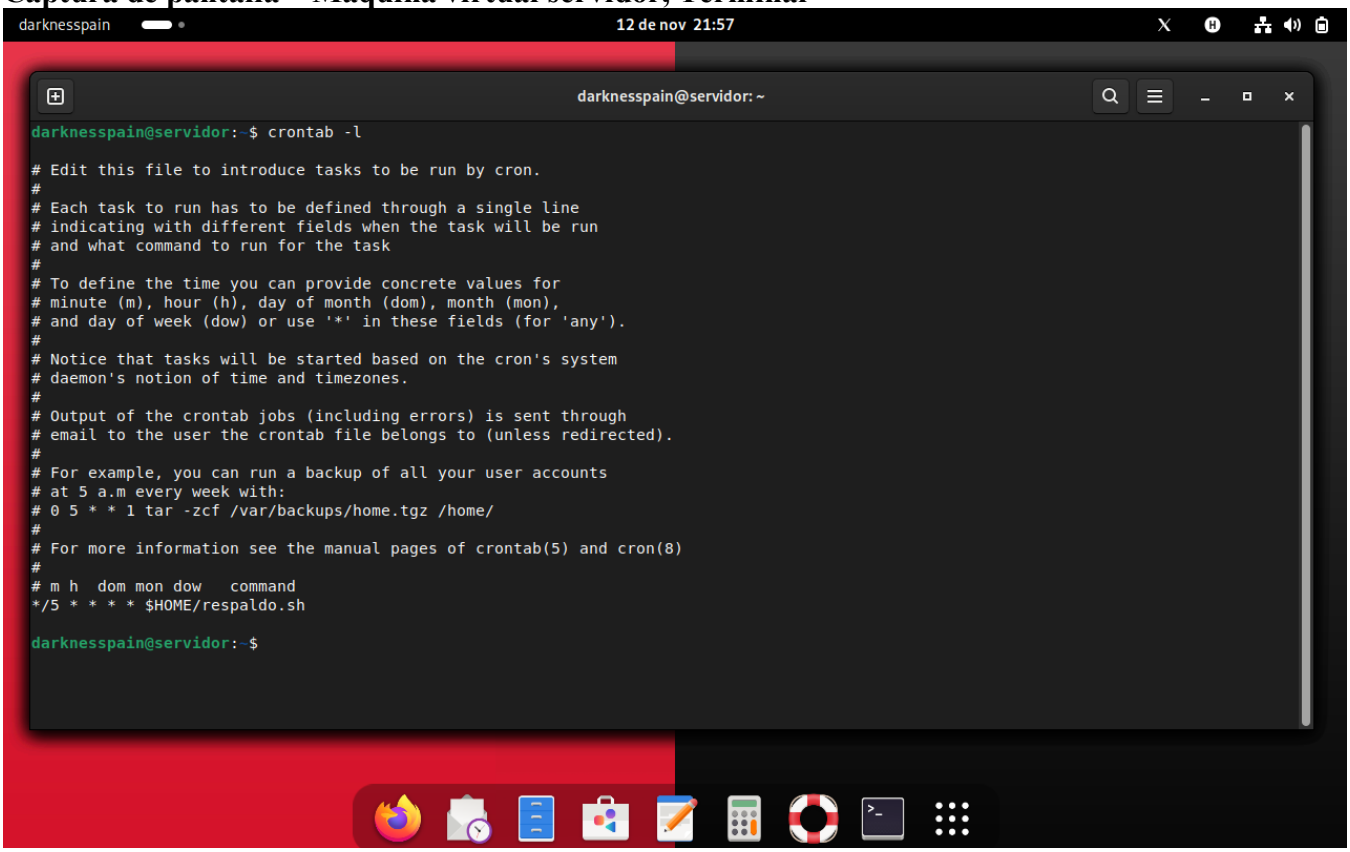


Nota. Con el grupo equipo activo, «~/respaldo.sh» crea proyecto_20261112_215533.tar.gz de 1,8 K y «ls -lh /datos/respaldos» muestra el archivo junto a errores.log.

La ejecución manual es el paso que no se debe saltar: si el script se programara en cron sin probarlo, un error de permisos pasaría inadvertido hasta que hiciera falta el respaldo. Aquí se ve además por qué hay que activar el grupo equipo: la primera vez que lo intentamos sin él, tar respondió «Permiso denegado» sobre /datos/proyecto y generó un archivo vacío de 45 bytes. Un

respaldo de 45 bytes es peor que ningún respaldo, porque parece que existe.

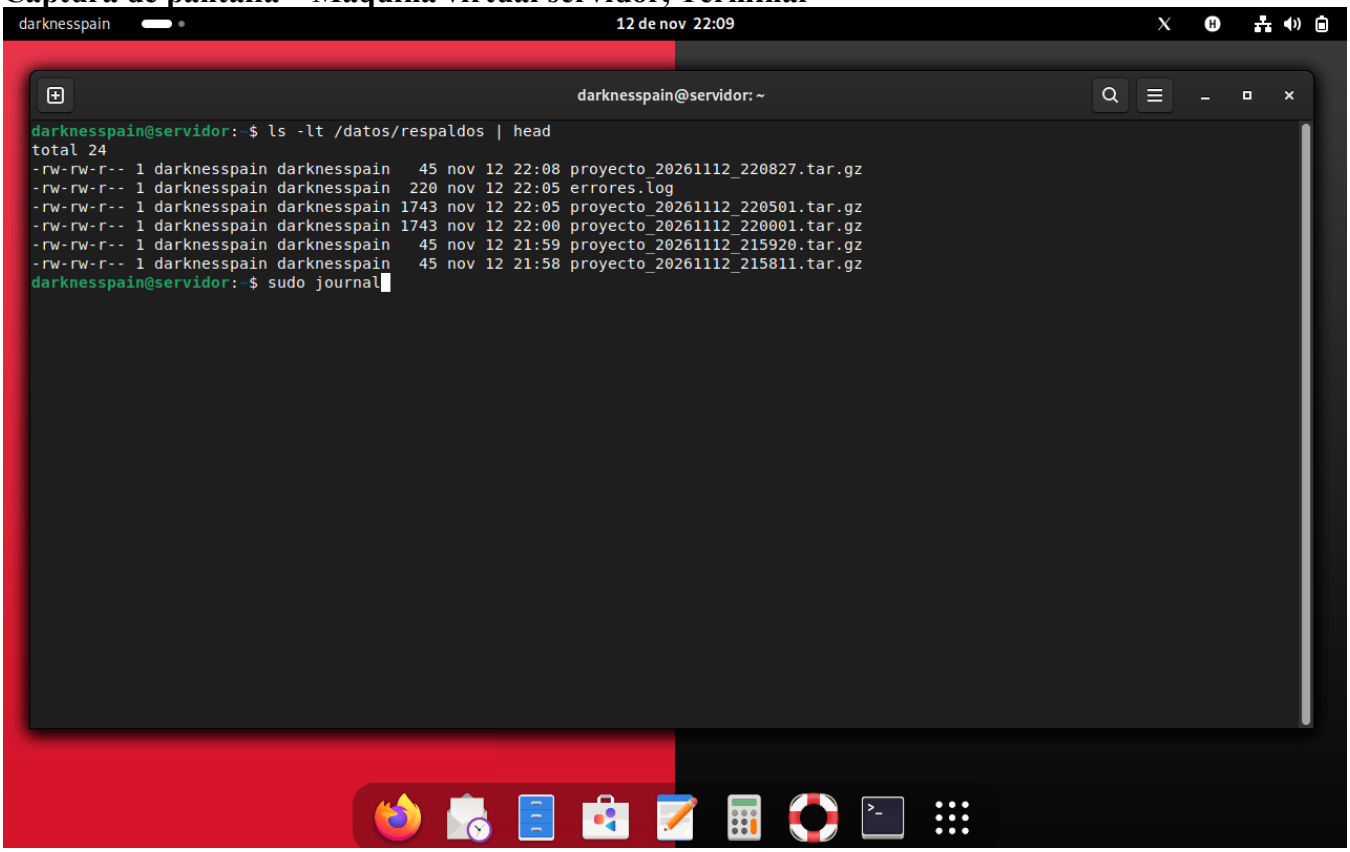
Figura 138
Captura de pantalla – Máquina virtual servidor, Terminal



Nota. «crontab -l» muestra el archivo con la línea «*/5 * * * * $HOME/respaldo.sh» al final, debajo de los comentarios que Debian agrega.

La línea tiene cinco campos de tiempo (minuto, hora, día, mes, día de la semana) y la orden a ejecutar. «*/5» significa cada cinco minutos, un intervalo elegido para poder ver el efecto dentro de una práctica; en producción sería diario. La variable $HOME se expande con el entorno de cron, que es mínimo: por eso conviene usar rutas absolutas o variables como esta, porque cron no hereda el PATH ni el directorio de trabajo de la sesión.

Figura 139
Captura de pantalla – Máquina virtual servidor, Terminal



Nota. «ls -lt /datos/respaldos | head» lista cinco respaldos con horas 22:08, 22:05, 22:00, 21:59 y 21:58; «journalctl -u cron --since '20 min ago'» muestra las ejecuciones del servicio cron.

Aquí se ven las dos cosas que pedía la guía: que aparecieron respaldos nuevos sin que nadie los lanzara a mano y que nunca hay más de cinco, porque el script borra los antiguos con «tail -n +6». El journal de cron confirma que el servicio ejecutó la tarea. Los respaldos de 21:59 y 22:00 coinciden con la prueba en que el cliente estaba apagado: el archivo local se creó igual y el error quedó anotado en errores.log, que es exactamente el comportamiento que buscábamos.

- **Pregunta 11. ¿Por qué el script conserva solo 5 respaldos? ¿Es suficiente tener el respaldo en la misma VM?**

Porque sin un límite la carpeta crecería hasta llenar el disco, que es justamente el incidente 1 de esta entrega: cada respaldo ocupa espacio y los bloques se agotan. Conservar cinco es un equilibrio entre tener historia suficiente y no comerse el disco; el mismo razonamiento aplica a los inodos, porque muchos archivos pequeños agotan la tabla de inodos aunque sobren bloques. Y no, no es suficiente tener el respaldo en la misma VM: si el disco se

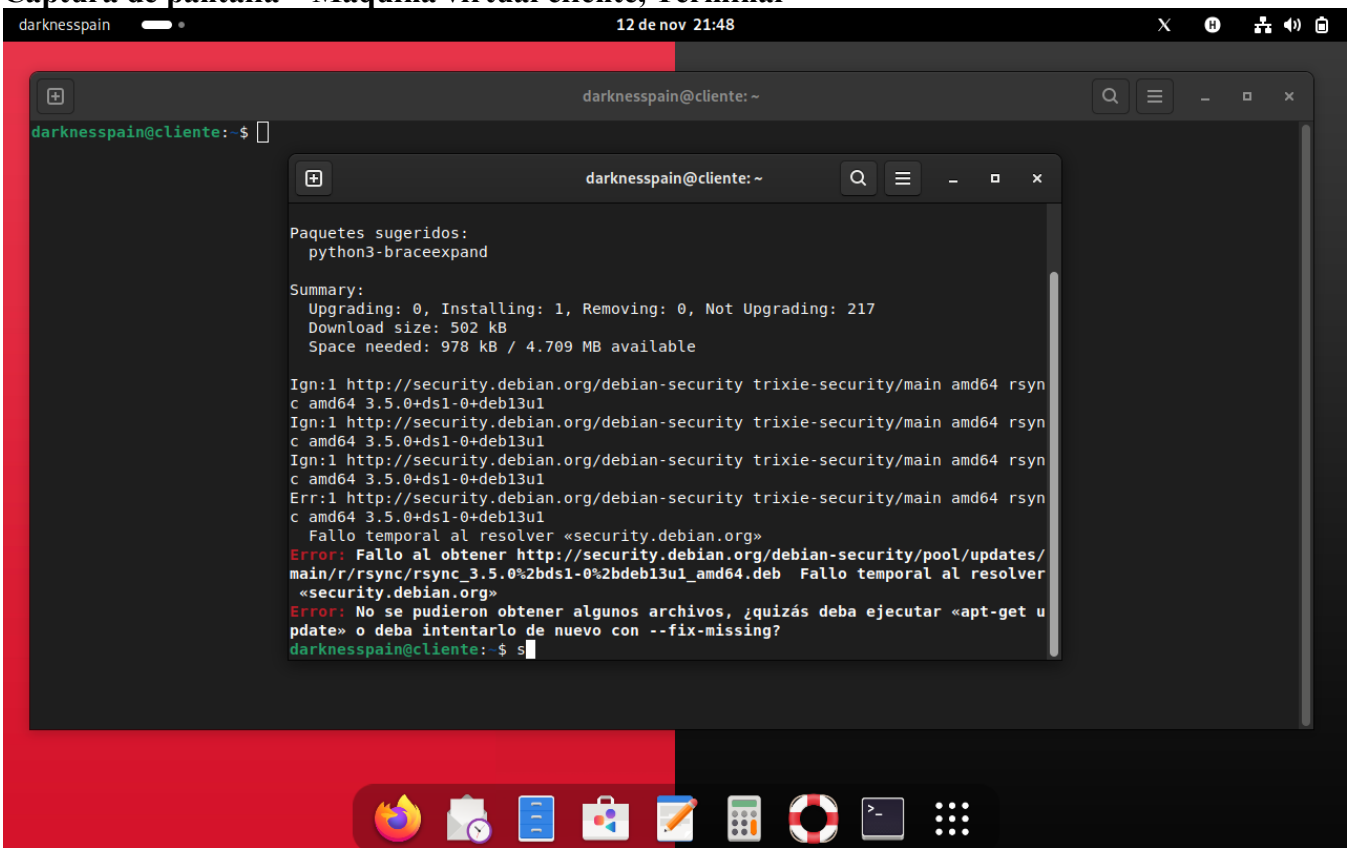
llena, se corrompe o la máquina no arranca, los respaldos se pierden con ella. Por eso el script copia a la otra máquina con rsync, y por eso en E.3 probamos qué pasa cuando ese segundo nodo no está: el respaldo local sobrevive, pero queda sin copia externa y con el error registrado.

Parte E — Un sistema distribuido de dos máquinas

E.1 Llaves SSH entre las dos máquinas

Hasta aquí las dos máquinas se conocían solo por ping. Para que el cliente administre al servidor y el servidor guarde copias en el cliente, cada una necesita una llave sin contraseña que la identifique ante la otra.

Figura 140
Captura de pantalla – Máquina virtual cliente, Terminal

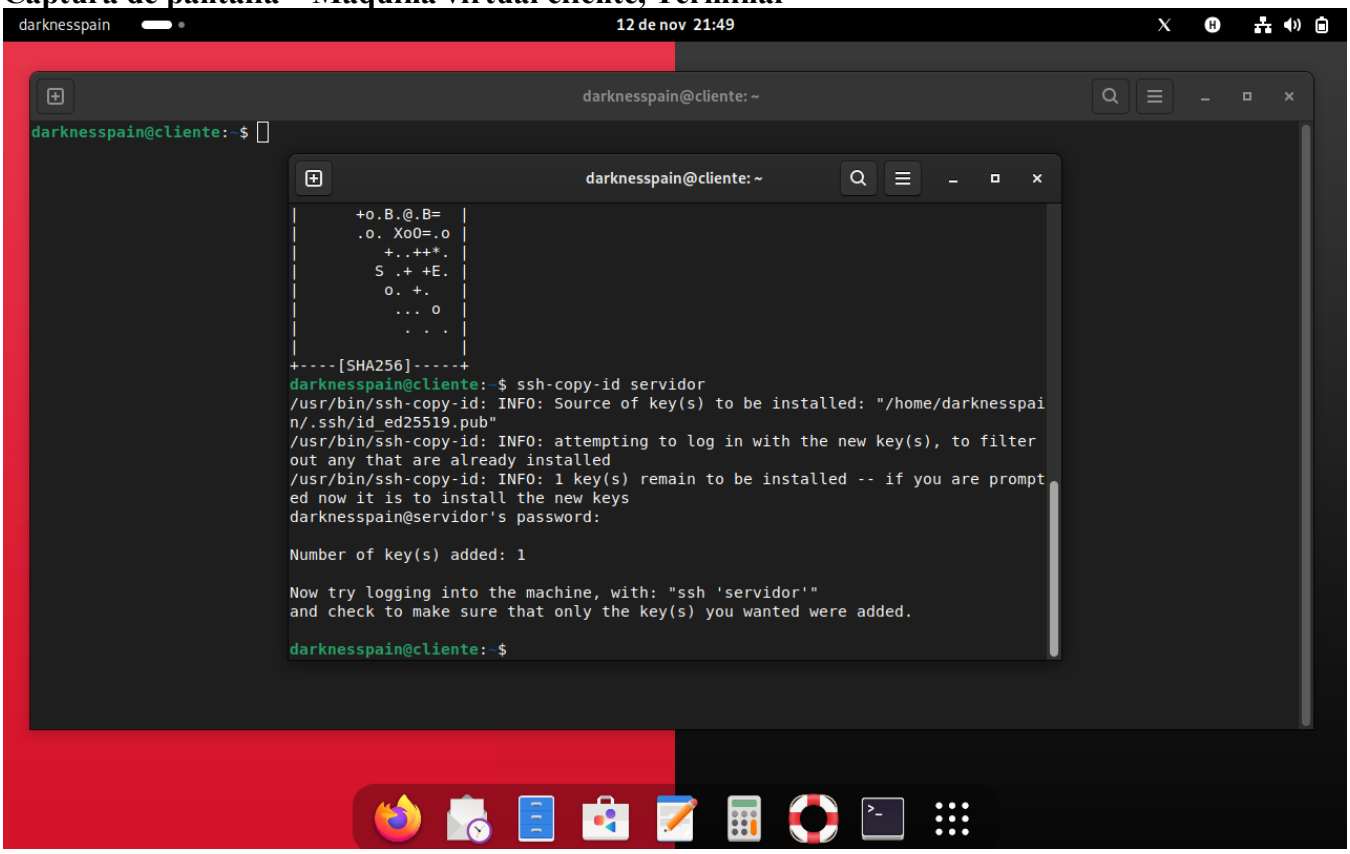


Nota. En el cliente, «ssh-keygen -t ed25519 -N '' -f ~/.ssh/id_ed25519» genera el par de llaves e imprime su huella SHA256 y el dibujo aleatorio de verificación.

La llave es de tipo ed25519, más corta y rápida que RSA con seguridad equivalente. La opción -N '' deja la llave privada sin frase de paso, que es lo que permite usarla desde un script o desde cron sin que nadie escriba nada: es el precio de la automatización, y por eso el archivo privado debe tener permisos 600. La huella impresa es la forma de comparar que la llave que se

copió es la misma que se generó, sin transmitir nunca la parte privada: la pública viaja, la privada se queda.

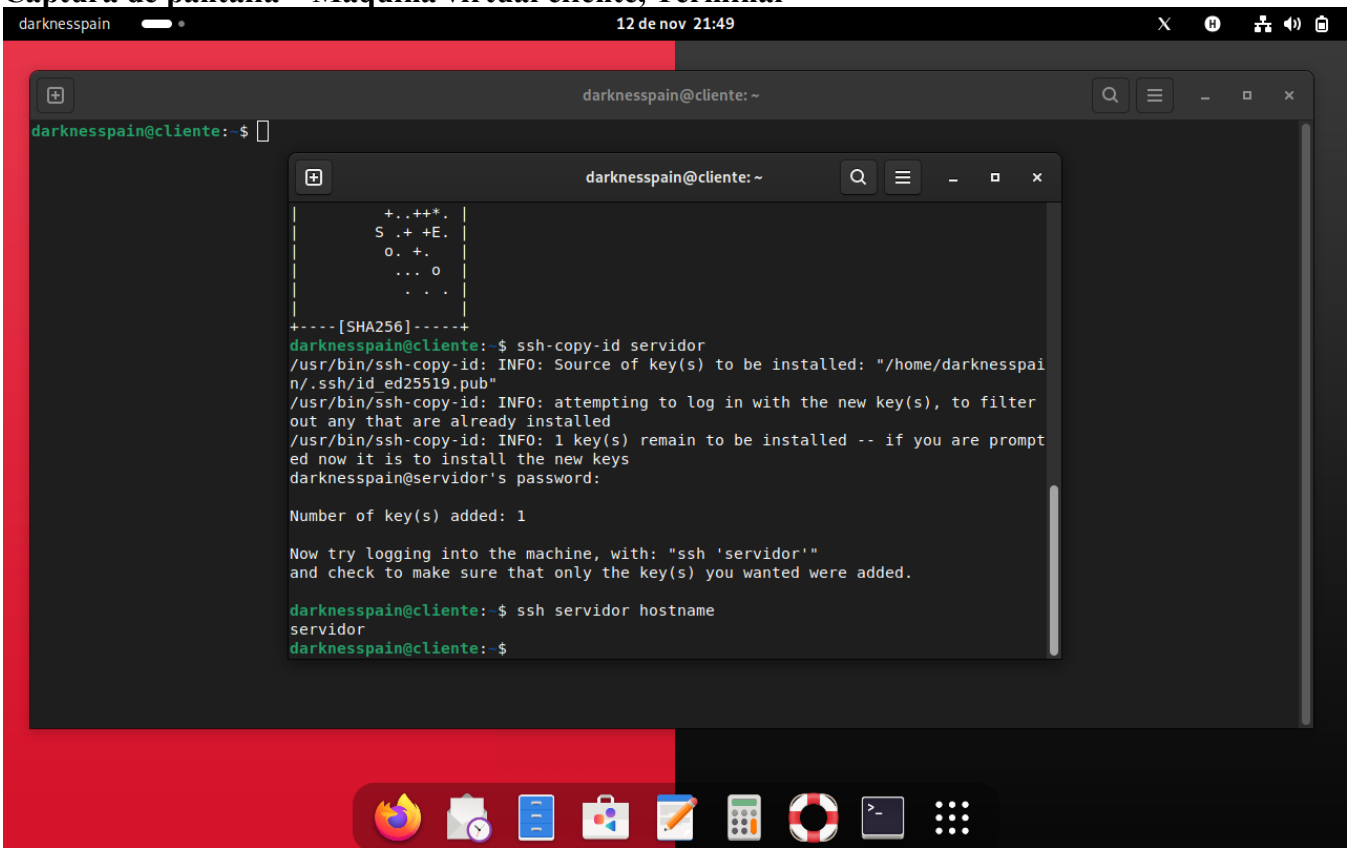
Figura 141
Captura de pantalla – Máquina virtual cliente, Terminal



Nota. «ssh-copy-id servidor» pide la contraseña una sola vez y responde con el número de llaves agregadas y un recordatorio para probar la conexión.

ssh-copy-id hace tres cosas invisibles: crea ~/.ssh en el destino si no existe, añade nuestra llave pública al final de authorized_keys y ajusta los permisos. La contraseña se escribe una única vez porque a partir de aquí la autenticación es por llave. Ese es el cambio de modelo: la contraseña viaja (aunque sea cifrada) y puede adivinarse; la llave privada no sale nunca de la máquina y la pública puede mostrarse sin riesgo.

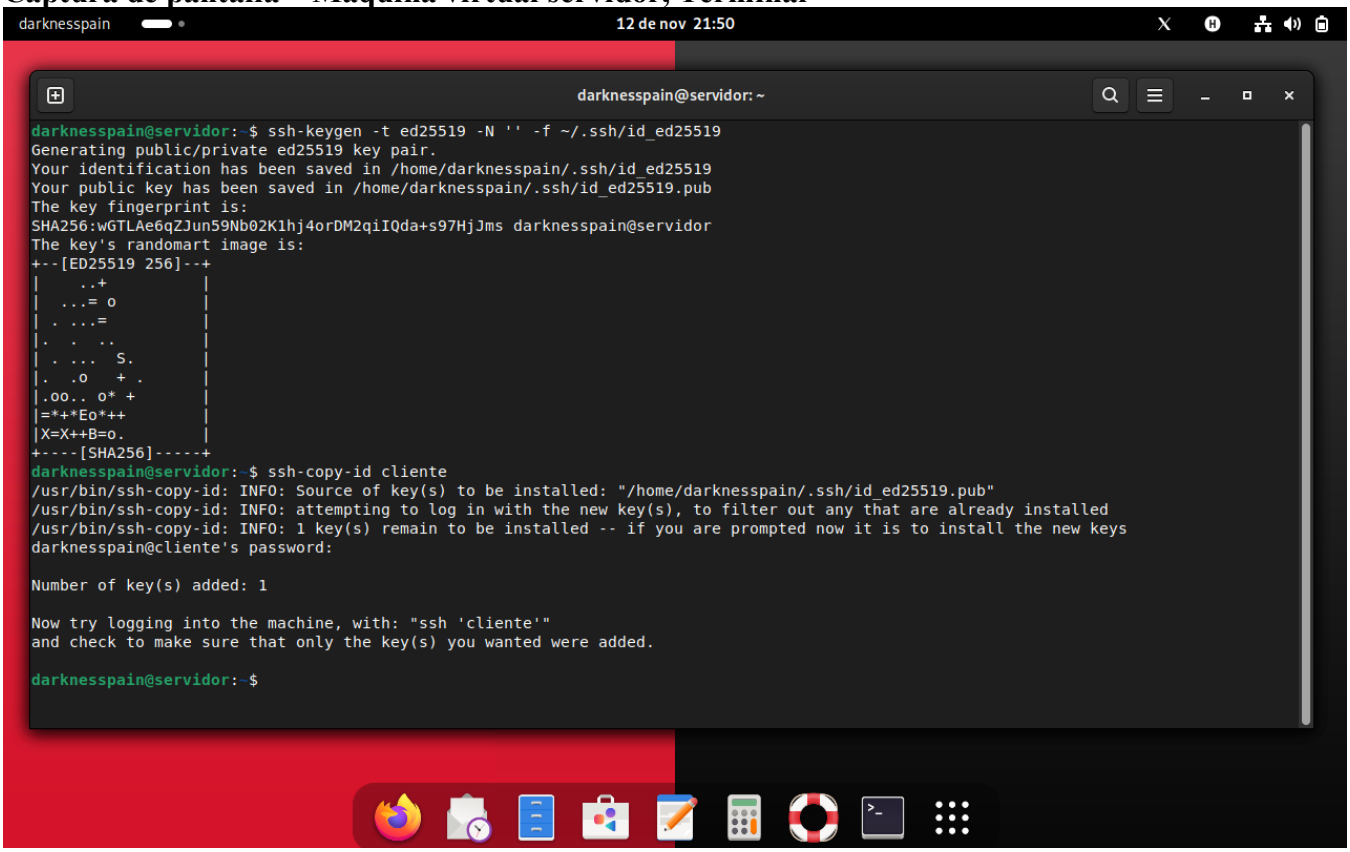
Figura 142
Captura de pantalla – Máquina virtual cliente, Terminal



Nota. «ssh servidor hostname» devuelve «servidor» sin pedir contraseña.

El comando remoto se ejecuta en el servidor y devuelve solo su salida, que es la prueba de que la autenticación por llave funciona. Este patrón —ssh máquina 'orden'— es el que usa cualquier herramienta de administración: no abre una sesión interactiva, ejecuta y cierra. Si aquí hubiera pedido contraseña, el problema estaría en los permisos de ~/.ssh (700) o de authorized_keys (600), que ssh rechaza si son demasiado abiertos.

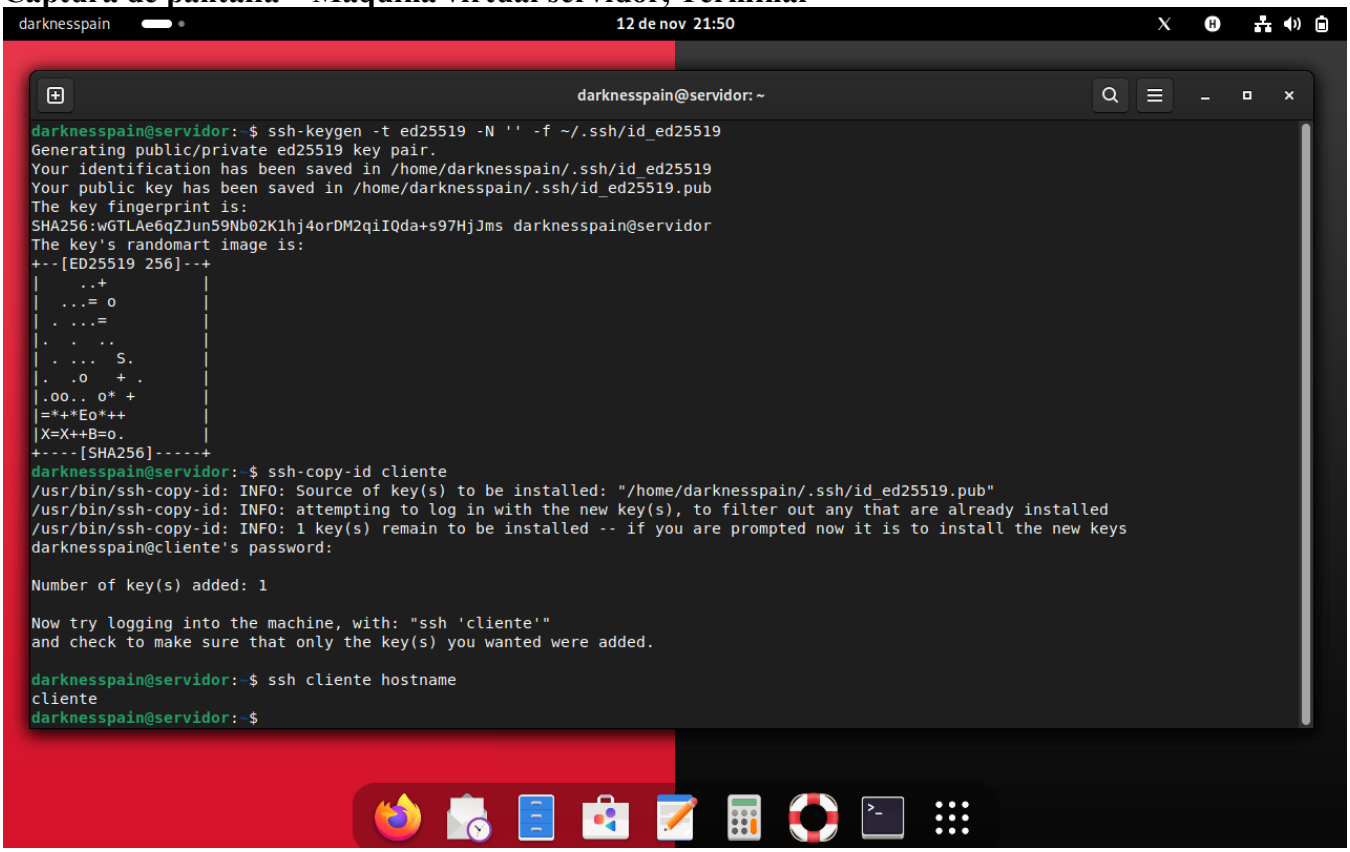
Figura 143
Captura de pantalla – Máquina virtual servidor, Terminal



Nota. En el servidor, ssh-keygen genera su par de llaves ed25519 y «ssh-copy-id cliente» instala la pública en el cliente tras pedir la contraseña una vez.

La dirección importa: el servidor necesita su propia llave porque es él quien empuja los respaldos al cliente con rsync. Sin esta llave, el script funcionaría a mano (con contraseña) pero fallaría desde cron, donde no hay nadie para escribirla. Con las dos llaves instaladas, la relación es mutua y ninguna de las dos máquinas depende de una contraseña para hablar con la otra.

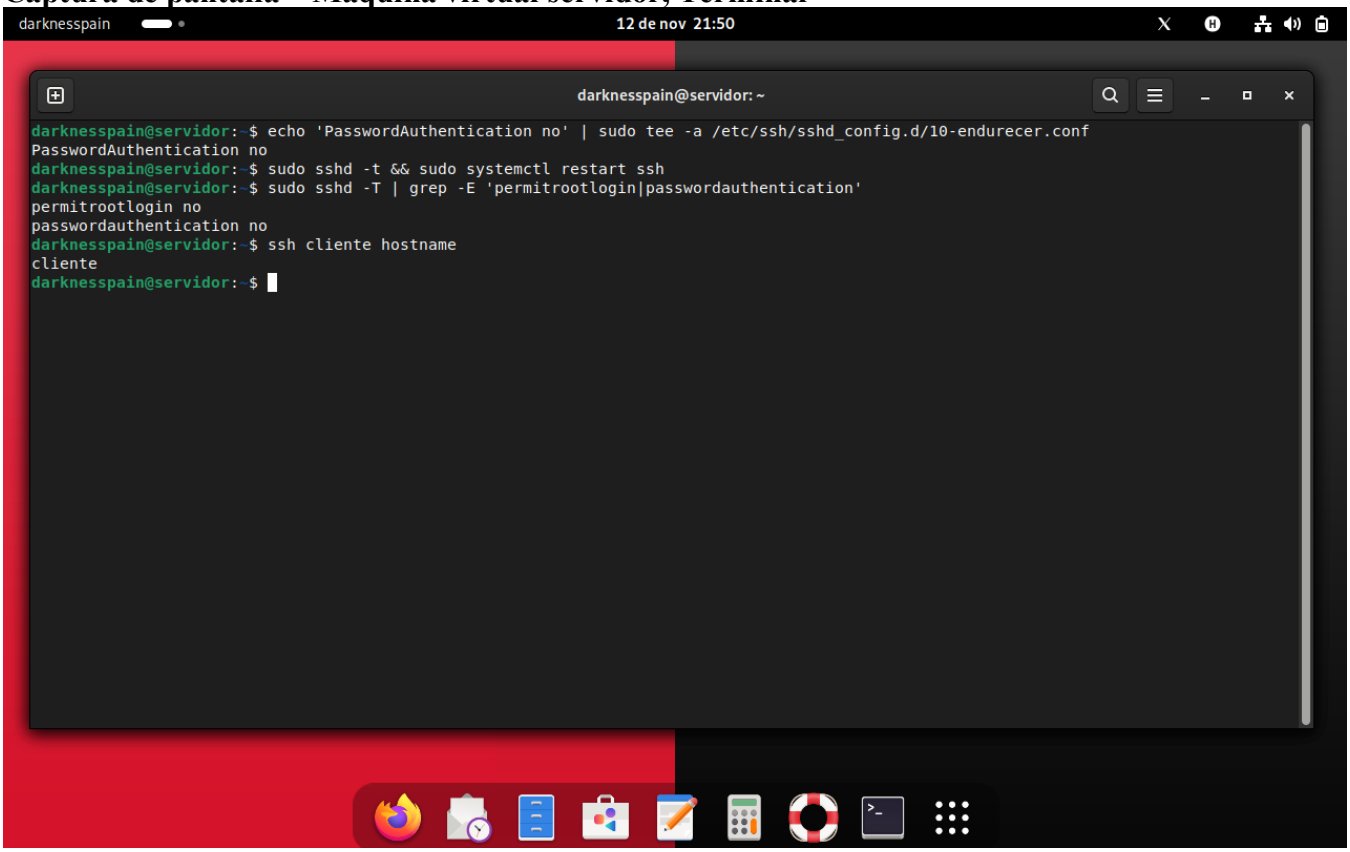
Figura 144
Captura de pantalla – Máquina virtual servidor, Terminal



Nota. «ssh cliente hostname» responde «cliente» sin pedir contraseña.

Es la verificación simétrica de la Figura anterior: ahora el servidor entra al cliente. Con las dos direcciones comprobadas se puede cerrar la puerta de las contraseñas sin miedo a quedarse fuera, que es el paso siguiente. Este orden —probar primero en una segunda terminal y solo después cambiar la configuración— es la regla de oro de la administración remota: nunca se desactiva el método de acceso que se está usando sin tener otro ya funcionando.

Figura 145
Captura de pantalla – Máquina virtual servidor, Terminal

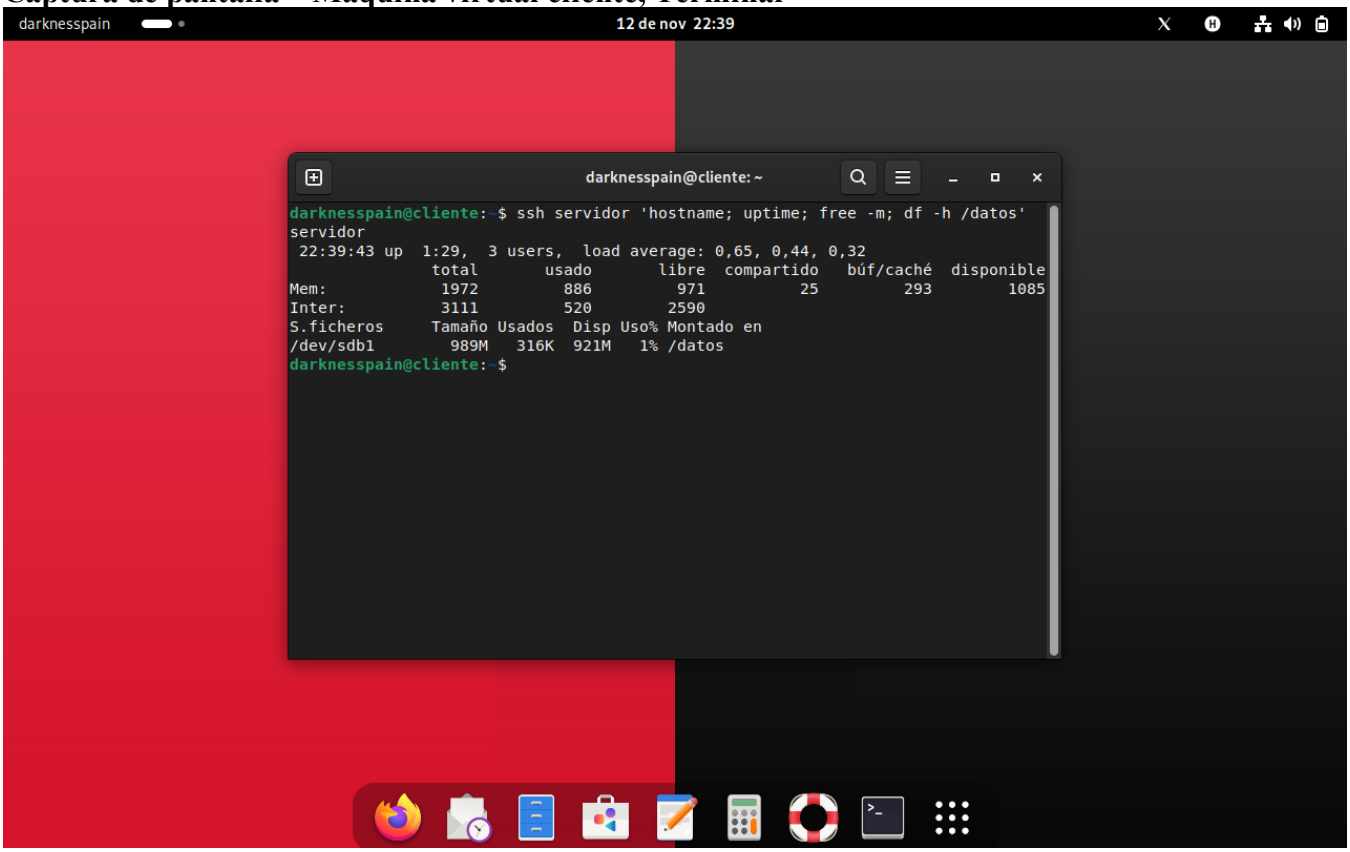


Nota. Se añade «PasswordAuthentication no» al archivo 10-endurecer.conf; tras sshd -t y reiniciar, «sshd -T» muestra «permitrootlogin no» y «passwordauthentication no», y «ssh cliente hostname» sigue respondiendo «cliente».

Con esta línea el servidor deja de aceptar contraseñas: solo entra quien tenga una llave autorizada. Eso elimina de golpe los ataques de fuerza bruta por contraseña, que son la mayoría del tráfico malicioso que recibe cualquier servidor con el puerto 22 abierto. El riesgo es encerrarse fuera, y por eso la comprobación se hace en la misma captura: la conexión por llave sigue funcionando después del cambio. Si se perdiera la llave, quedaría el acceso por la consola de VirtualBox.

E.2 Administrar a distancia

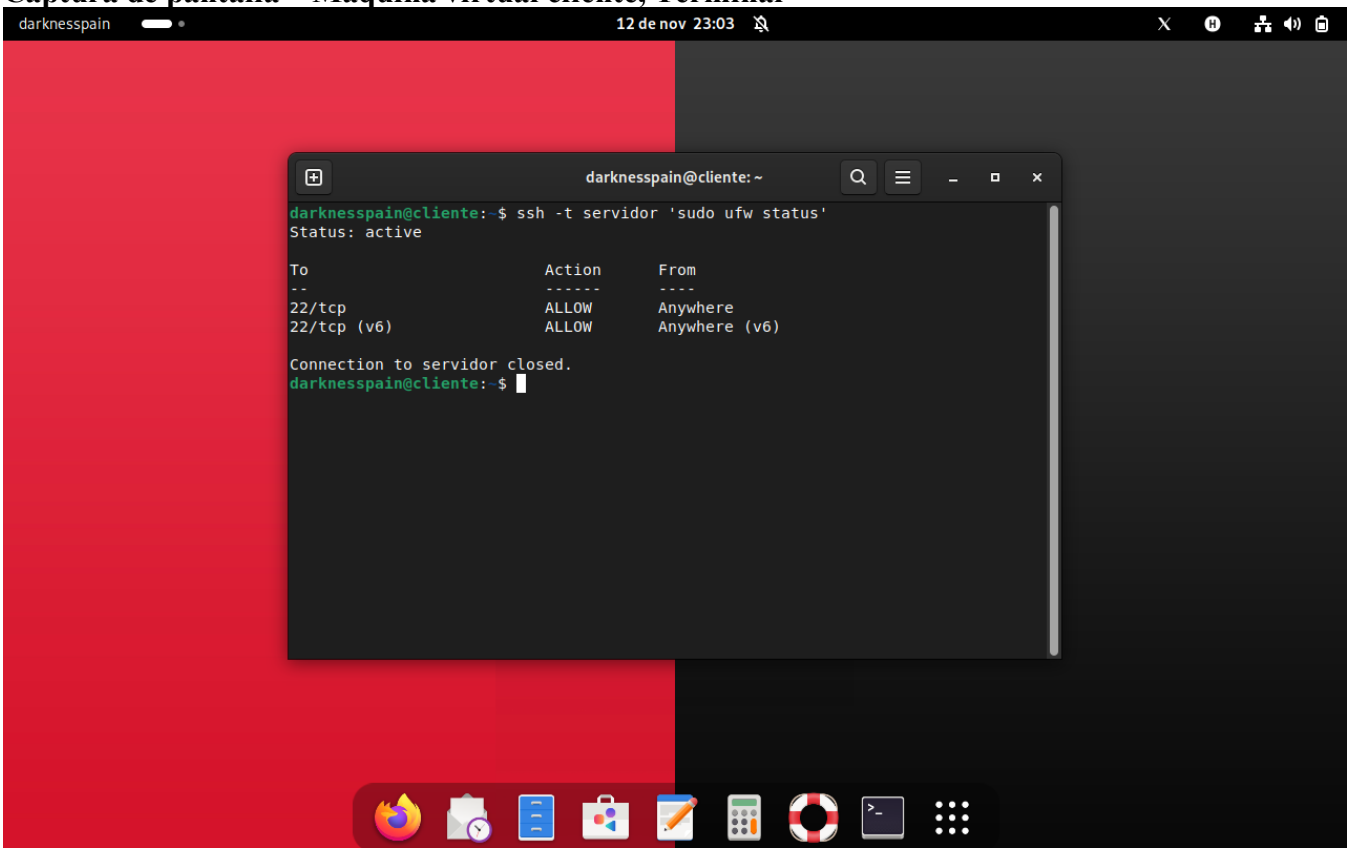
Figura 146
Captura de pantalla – Máquina virtual cliente, Terminal



Nota. «ssh servidor 'hostname; uptime; free -m; df -h /datos'» devuelve servidor, el tiempo encendido con la carga media, la tabla de memoria y el uso de /datos.

Una sola orden desde el cliente trae cuatro vistas del servidor. Esto es la administración remota en su forma más simple y también la base de cualquier sistema de monitorización: ejecutar comandos y leer su salida. Nótese que no hace falta abrir sesión: ssh ejecuta y devuelve el control, lo que permite encadenar comprobaciones desde un script sin intervención humana.

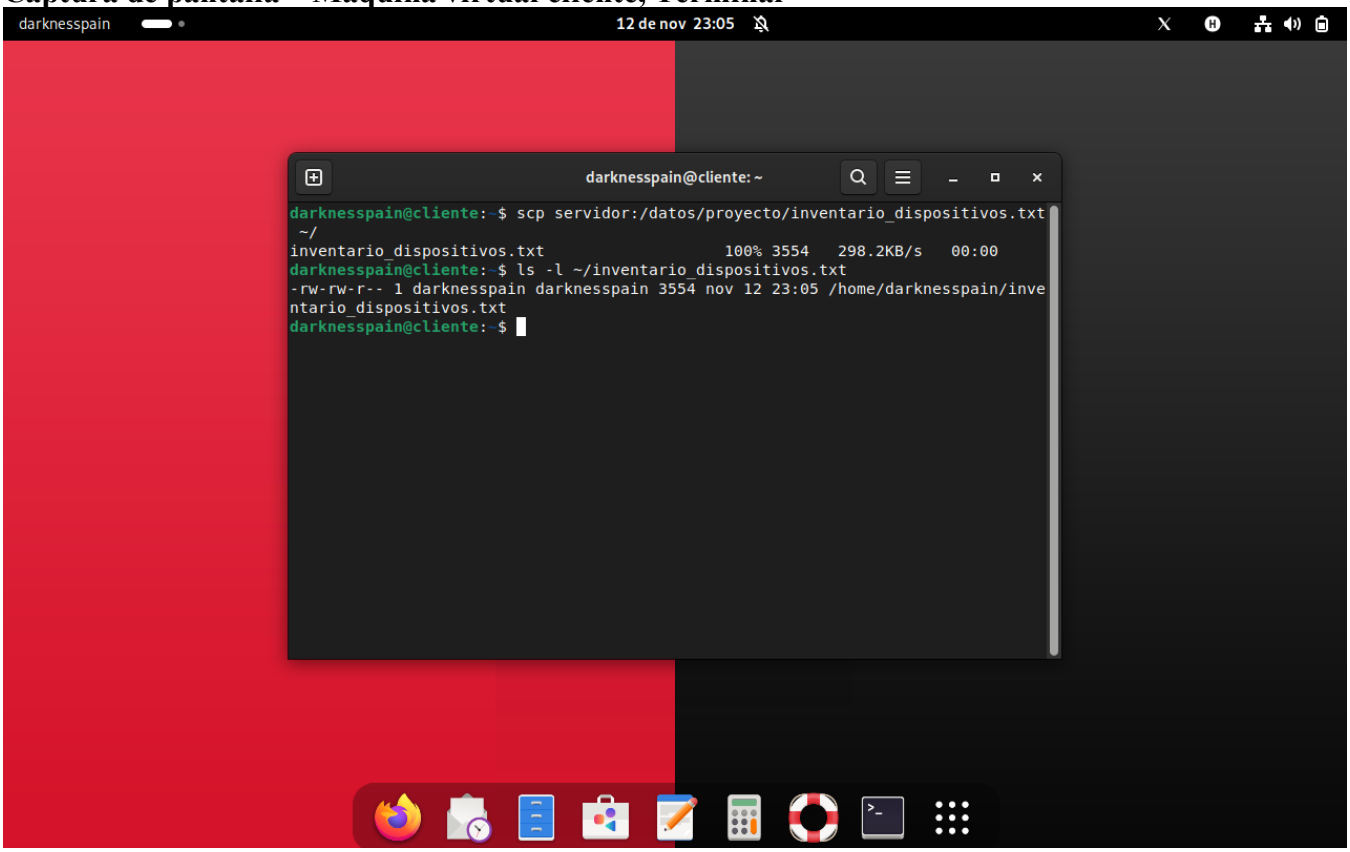
Figura 147
Captura de pantalla – Máquina virtual cliente, Terminal



Nota. «ssh -t servidor 'sudo ufw status'» pide la contraseña de sudo en el servidor y muestra el estado del cortafuegos con la regla 22/tcp permitida.

La opción -t fuerza la asignación de una terminal remota, y sin ella sudo se negaría a pedir la contraseña («no tty present»). Es un detalle que explica por qué algunas órdenes con sudo funcionan en una sesión interactiva y fallan dentro de un script: el programa necesita un terminal para leer la clave sin mostrarla. La alternativa en automatización es sudoers con NOPASSWD para comandos concretos, nunca la contraseña escrita en el script.

Figura 148
Captura de pantalla – Máquina virtual cliente, Terminal



Nota. «scp servidor:/datos/proyecto/inventario_dispositivos.txt ~/» copia el archivo y «ls -l» confirma que llegó al directorio personal del cliente.

scp usa SSH por debajo, así que hereda la autenticación por llave que acabamos de configurar. Sirve para traer evidencia del servidor sin abrir sesión interactiva: es la manera de recoger el inventario, un registro o un archivo de configuración para analizarlo en otra máquina. En la Parte G se usa el mismo criterio para llevarse el reporte del verificador.

Tabla 8
Comparación de las dos máquinas (E.2)

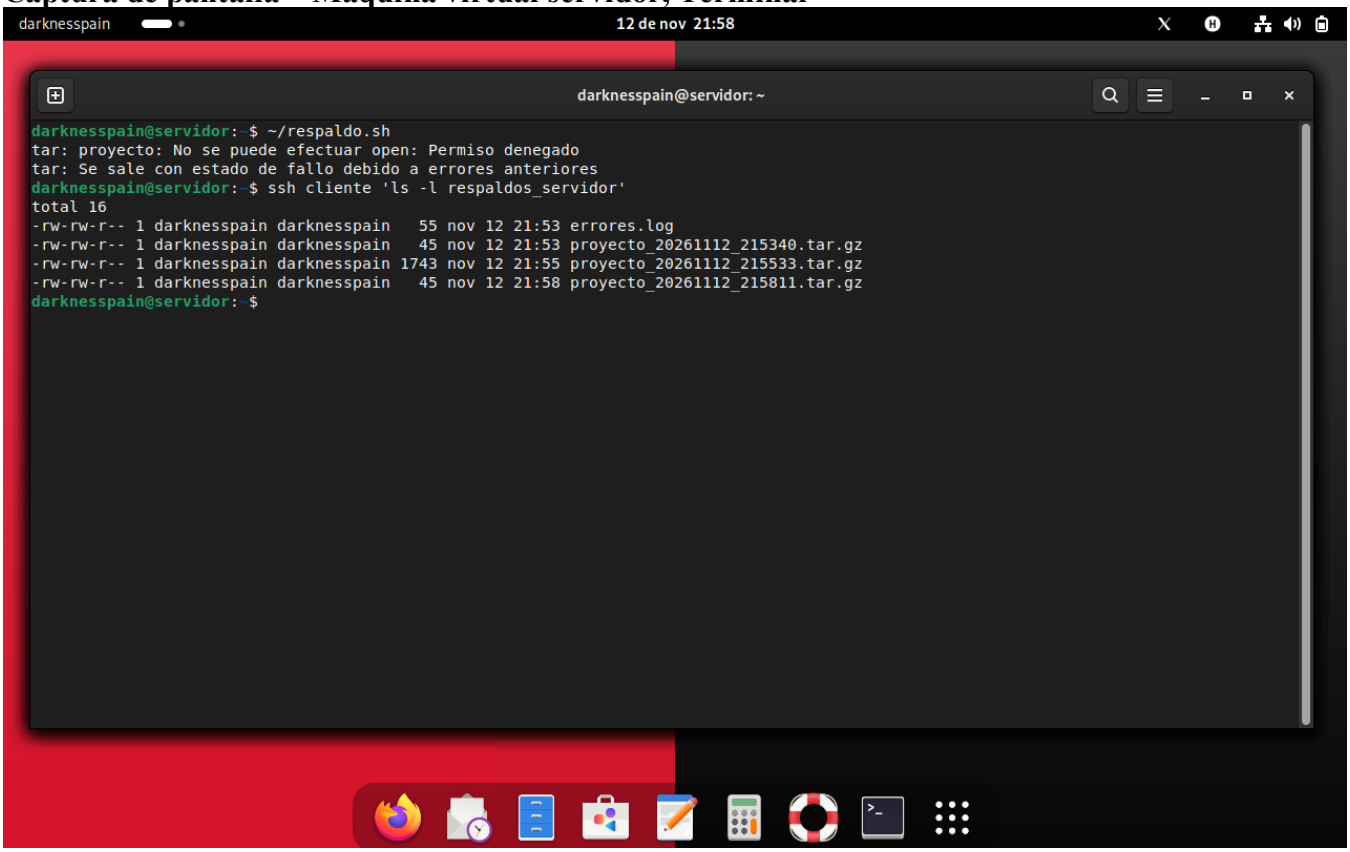
Pregunta	En servidor	En cliente
Memoria disponible	1064 MB de 1972 MB (923 MB libres)	175 MB de 965 MB (80 MB libres)
Tiempo encendido	1 hora 36 minutos	40 minutos
Últimos usuarios de /etc/passwd	darknesspain, sshd, lector	darknesspain, sshd

Nota. Datos tomados con free -m, uptime -p y cut -d: -f1 /etc/passwd al cierre de la práctica. El cliente tiene la mitad de RAM porque así se configuró el clon, y por eso le queda menos memoria disponible.

E.3 Respaldo a la otra máquina y falla de un nodo

Antes de apagar el cliente escribimos la predicción: el respaldo local se crearía igual, la copia remota fallaría en pocos segundos por BatchMode y el rastro quedaría en errores.log; al volver a encender el cliente, rsync sincronizaría lo pendiente.

Figura 149
Captura de pantalla – Máquina virtual servidor, Terminal

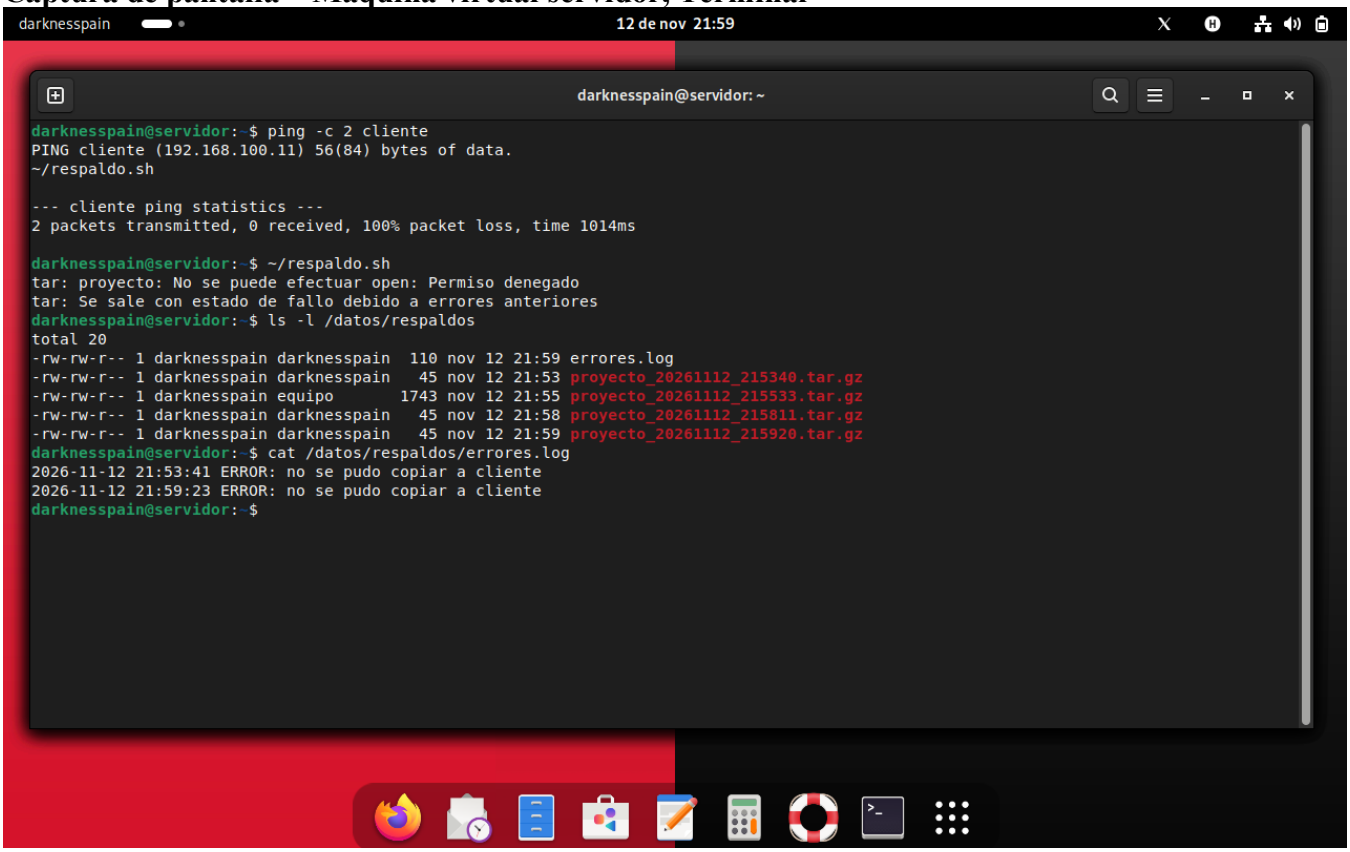


Nota. Con el cliente encendido, «~/respaldo.sh» crea el archivo y «ssh cliente 'ls -l respaldos_servidor'» muestra las copias que ya están en la otra máquina.

Este es el caso normal: el tar se crea en el servidor y rsync lo empuja al cliente. La copia existe en dos máquinas distintas, que es la única forma de sobrevivir a la pérdida de una.

Conviene fijarse en que el script no pide contraseña: eso es posible gracias a la llave de E.1 y a la opción BatchMode, que además hace que falle rápido en lugar de quedarse esperando una respuesta que nunca llegará.

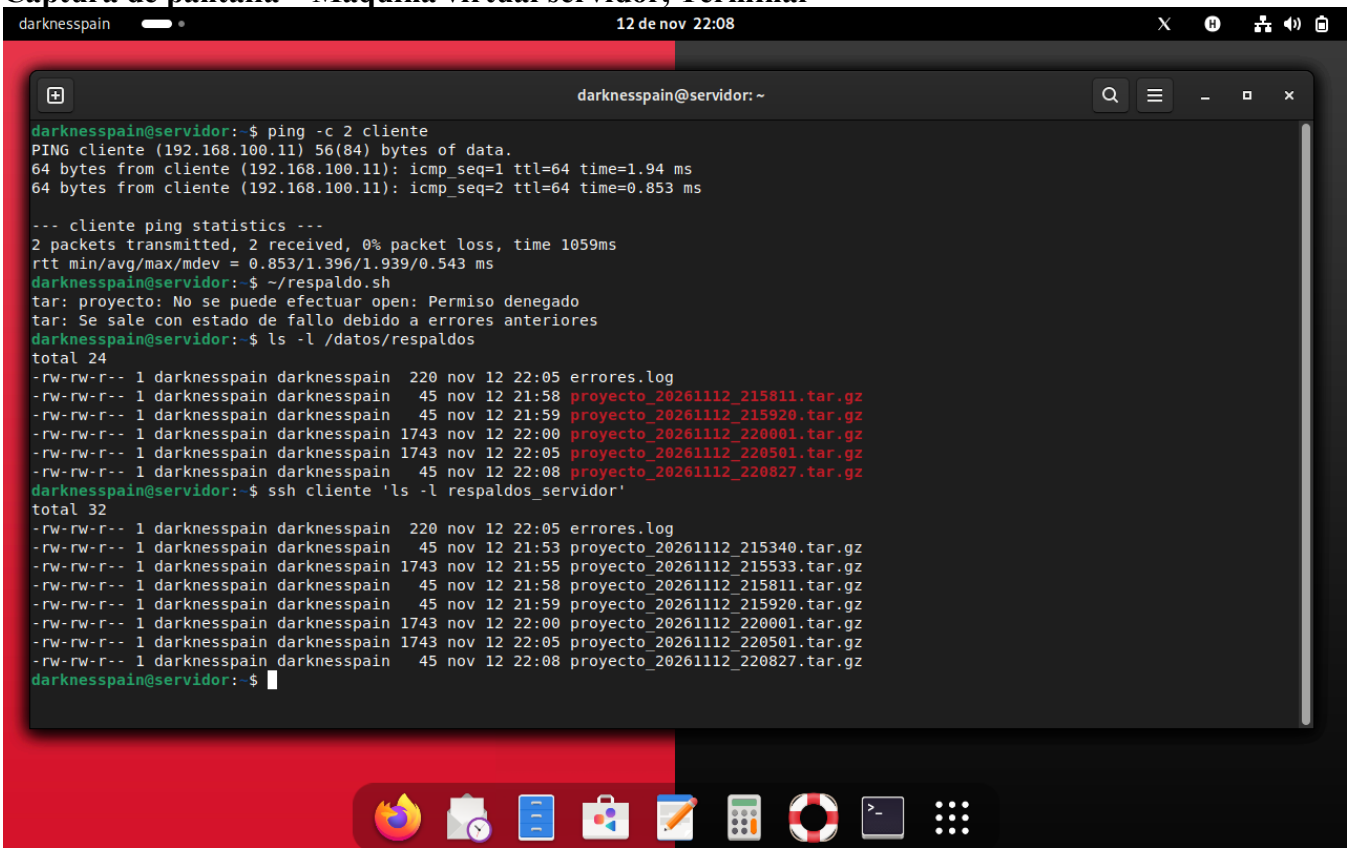
Figura 150
Captura de pantalla – Máquina virtual servidor, Terminal



Nota. Con el cliente apagado, el ping falla, «~/respaldo.sh» sigue creando el respaldo local (proyecto_20261112_215920.tar.gz) y «cat errores.log» muestra la línea «2026-11-12 21:59:23 ERROR: no se pudo copiar a cliente».

El fallo fue parcial, tal como se predijo: el servicio de respaldo del servidor no se detuvo, el archivo local se generó y solo falló la pata remota. Ese es el comportamiento correcto de un sistema tolerante a fallos: degradarse en lugar de caerse. El detalle importante es que el error quedó registrado con fecha y hora, porque un fallo silencioso es peor que un fallo ruidoso: el administrador se enteraría de que no tiene copias el día que las necesita.

Figura 151
Captura de pantalla – Máquina virtual servidor, Terminal



Nota. Tras encender el cliente, el ping responde, «~/respaldo.sh» vuelve a copiar y «ssh cliente 'ls -l respaldos_servidor'» muestra proyecto_20261112_220827.tar.gz en el cliente.

La recuperación no requirió ninguna intervención especial: rsync compara tamaños y fechas y sincroniza lo que falta, incluyendo el archivo que había quedado pendiente. El sistema vuelve a su estado normal solo. Lo que no es automático es enterarse del fallo: para eso está errores.log, que en una instalación real se revisaría con una alerta o se enviaría por correo.

- **Pregunta 12. Con esta prueba vieron un fallo parcial: una máquina cae y la otra sigue. Expliquen qué propiedades de un sistema operativo distribuido se ven aquí y cuáles faltan en su montaje.**

Se ve la tolerancia a fallos en su forma mínima: el servidor sigue prestando su servicio (respaldar) aunque el cliente no esté, y el sistema se degrada en vez de detenerse. También hay compartición de recursos, porque el disco del cliente se usa como segundo destino, y cierta transparencia, ya que rsync y ssh ocultan que se trata de otra máquina: para el script es una ruta más. Lo que falta es todo lo demás: no hay detección automática del fallo ni conmutación por error (el respaldo remoto simplemente no ocurre), no hay consistencia garantizada entre las dos

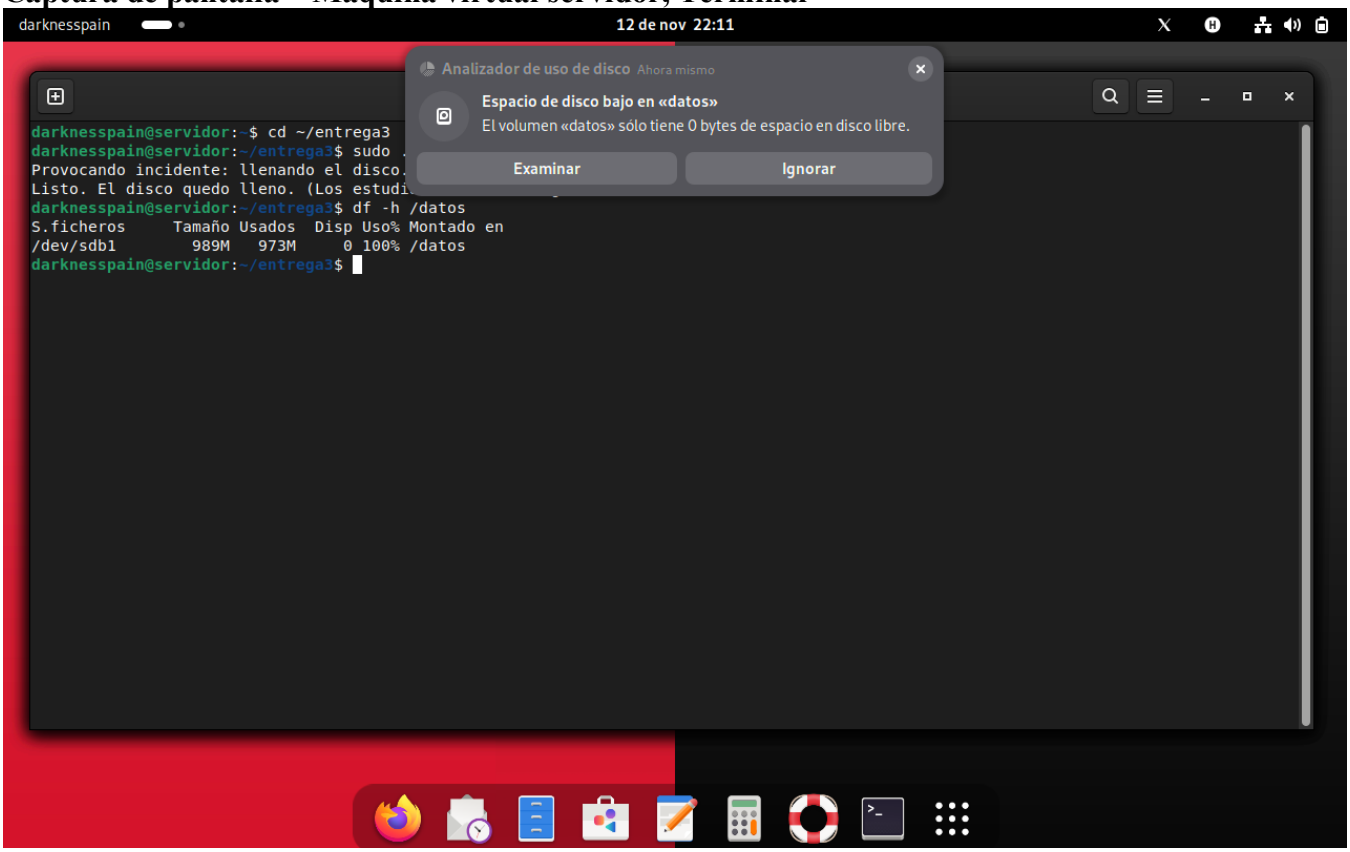
copias (si el cliente cae a mitad de la transferencia, rsync lo resuelve al reintentar, pero nadie lo reintenta solo), no hay reloj ni estado compartido, y el nombre del cliente se resuelve por /etc/hosts, así que no hay servicio de nombres ni descubrimiento. En términos prácticos: tenemos replicación manual con reintento humano, no alta disponibilidad.

Parte F — Incidentes: el servidor falla y ustedes diagnostican

En cada incidente uno de los integrantes provocó la falla con el script incidentes.sh o con una orden concreta, y el otro la diagnosticó sin reiniciar nada. Todas las fallas quedaron confinadas a /datos/incidente, que el script crea y borra, de modo que el resto del servidor nunca estuvo en riesgo.

Incidente 1 — «No space left on device» por bloques agotados

Figura 152
Captura de pantalla – Máquina virtual servidor, Terminal

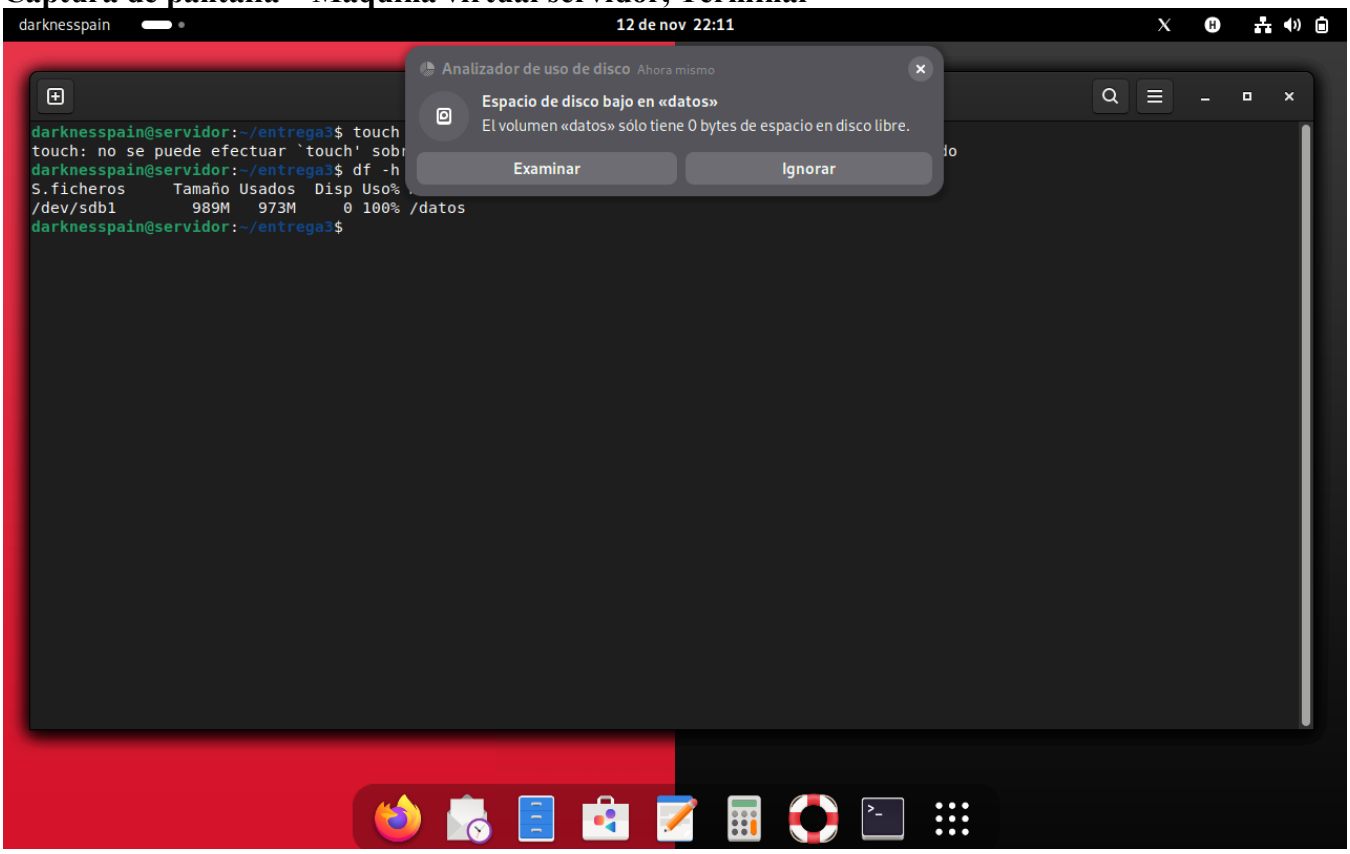


Nota. «sudo ./incidentes.sh disco» escribe un archivo gigante y «df -h /datos» queda en 989M de tamaño, 973M usados, 0 disponibles y 100 % de uso.

El sabotaje llena el disco con un único archivo de crecimiento indefinido hasta que el sistema de archivos no puede asignar más bloques. El síntoma es el mismo que verá cualquier

usuario: una operación de escritura que falla. Lo importante del diagnóstico es no quedarse en el mensaje y mirar el consumo real: df responde la pregunta «¿cuánto espacio por bloques queda?» y la respuesta es cero.

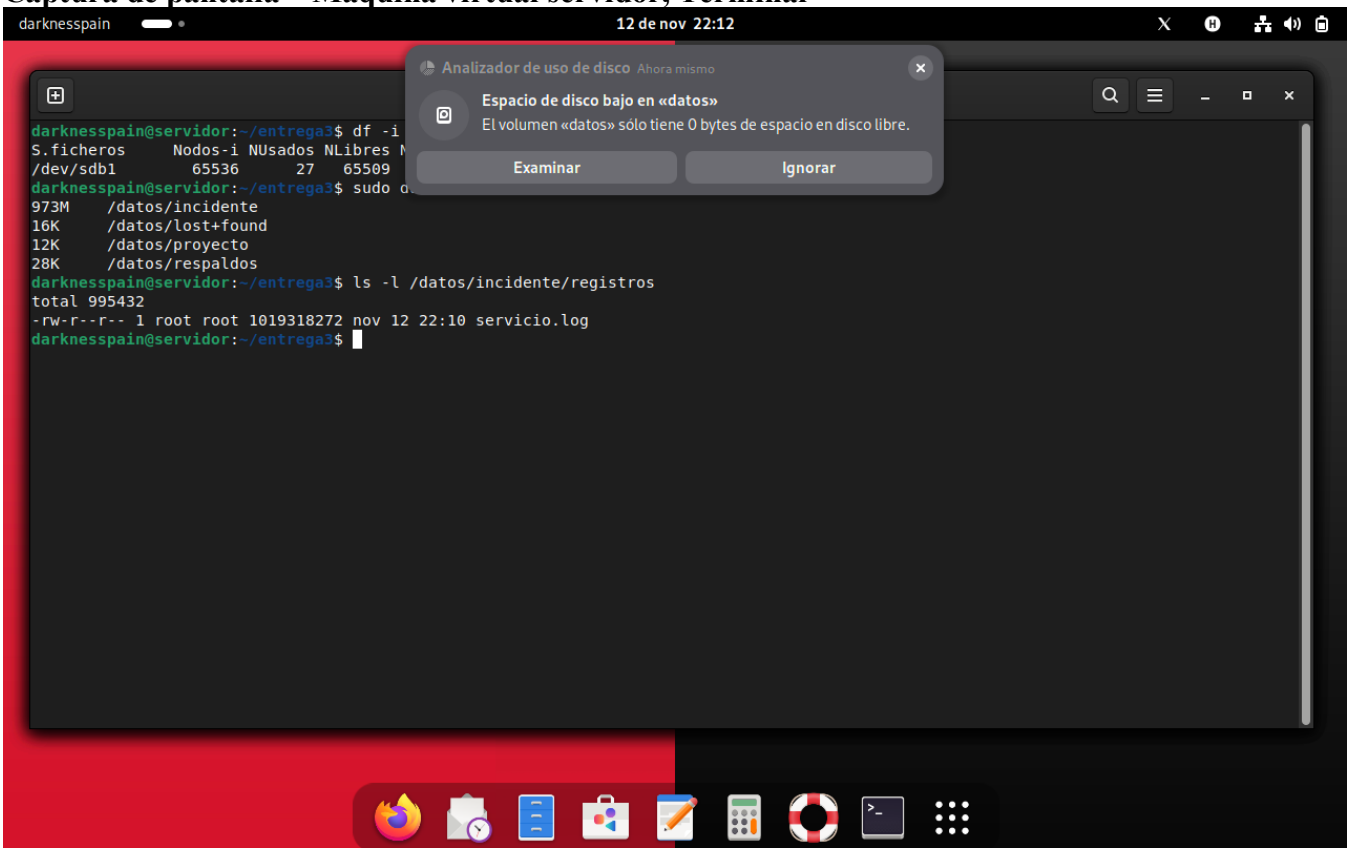
Figura 153
Captura de pantalla – Máquina virtual servidor, Terminal



Nota. «touch /datos/proyecto/prueba_incidente.txt» falla con «No space left on device» y «df -h /datos» confirma 100 % de uso.

El mensaje no dice qué archivo llenó el disco, solo que no hay espacio. Es la razón por la que el diagnóstico no puede terminar aquí: hay que encontrar al culpable. En un servidor real esta falla también afecta a los servicios que escriben registros, y puede impedir que arranque el sistema si el disco lleno es la raíz, porque systemd necesita escribir en /var.

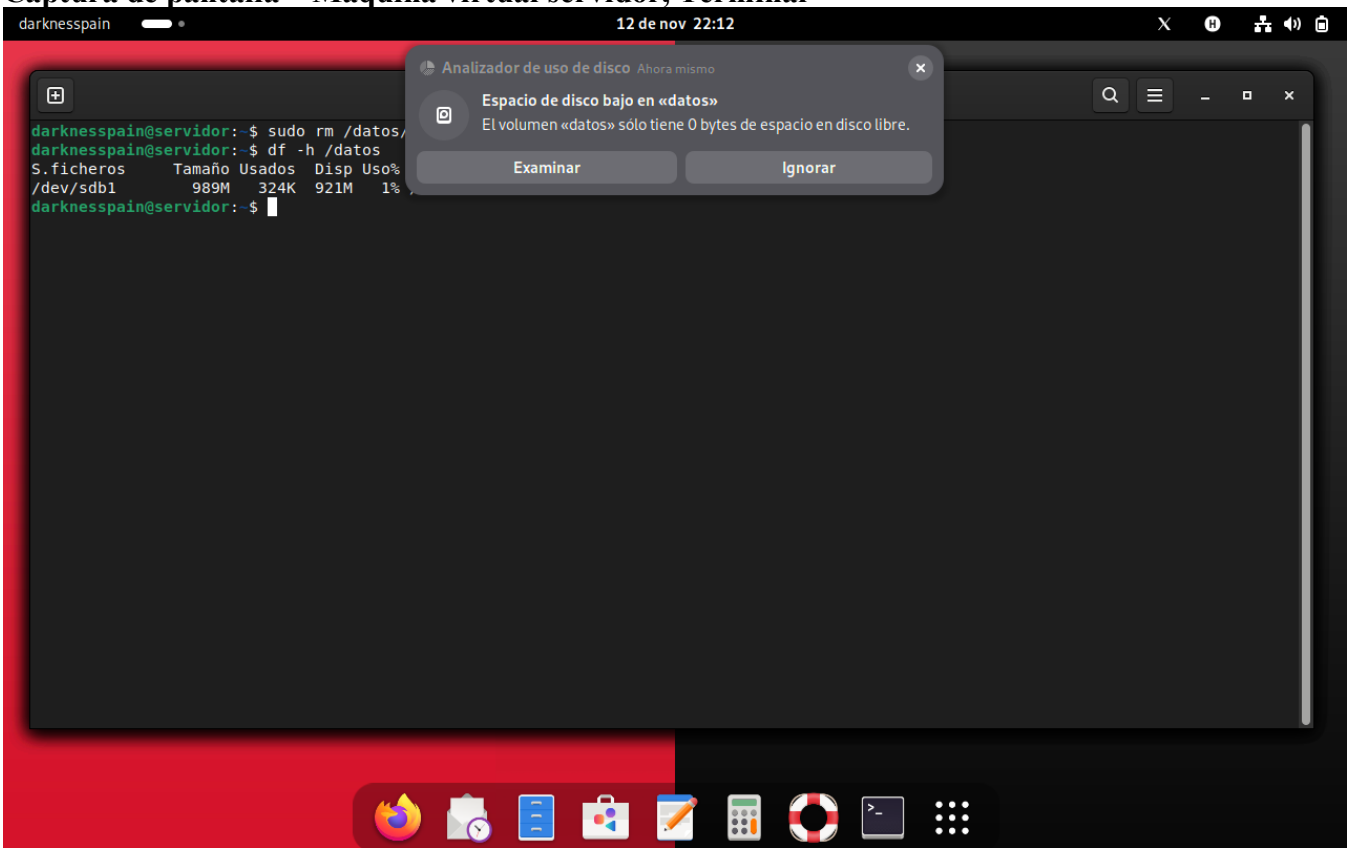
Figura 154
Captura de pantalla – Máquina virtual servidor, Terminal



Nota. «df -i /datos» muestra inodos casi libres, «sudo du -sh /datos/*» señala a /datos/incidente como la carpeta que pesa, y «ls -l /datos/incidente/registros» muestra un único servicio.log de 1019318272 bytes.

El orden del diagnóstico es lo que hay que aprender: primero se descarta el otro recurso (los inodos están libres, así que no es el incidente 2), después se busca la carpeta que más pesa y por último el archivo concreto con su tamaño. du -sh resume por carpeta y sort -h ordena por tamaño humano; ls -l del directorio final muestra el archivo de casi 1 GB. La causa raíz es un registro que creció sin límite, que es exactamente lo que ocurre cuando no hay rotación de logs.

Figura 155
Captura de pantalla – Máquina virtual servidor, Terminal

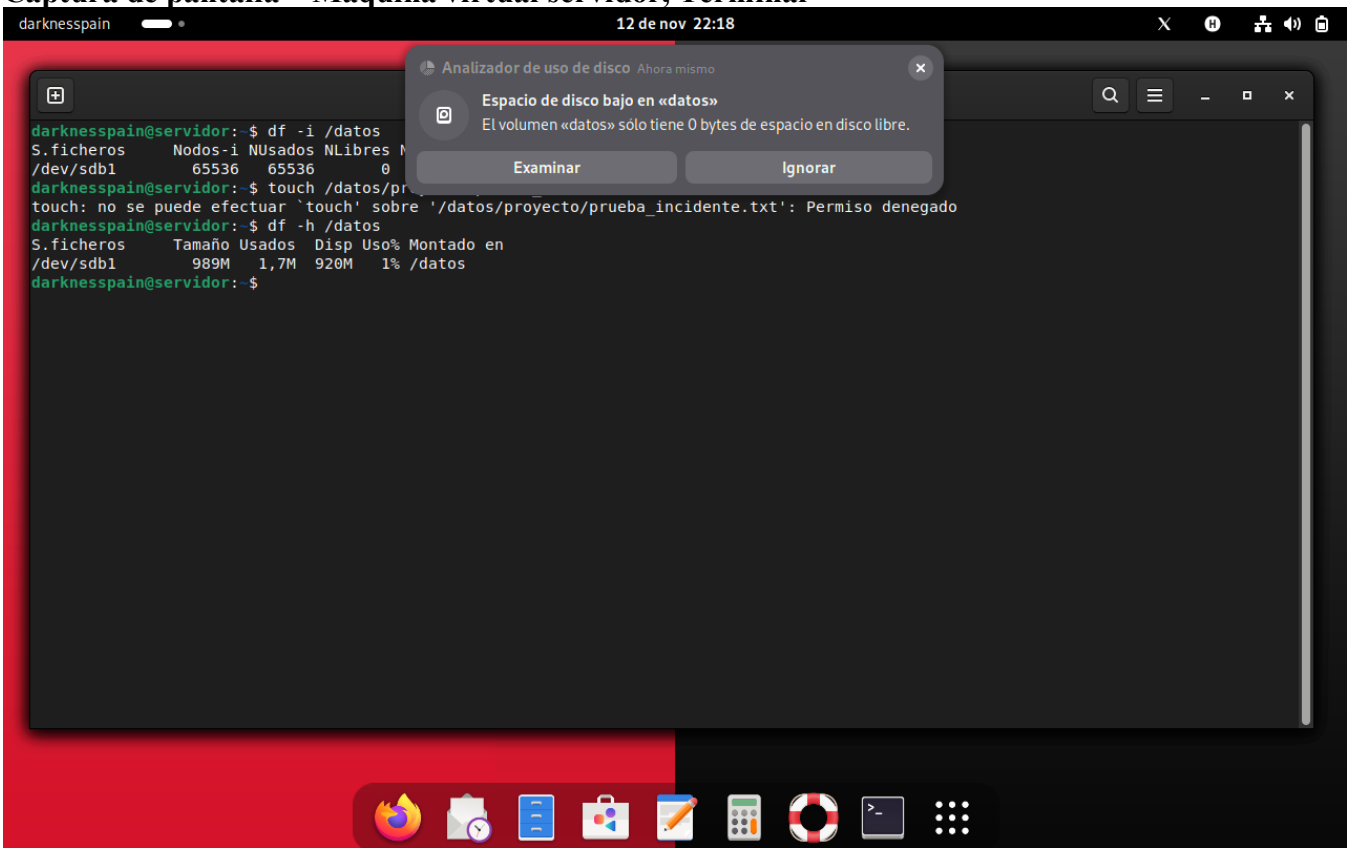


Nota. «sudo rm /datos/incidente/registros/servicio.log» y «df -h /datos» vuelve a 1 % de uso con 921 MB disponibles.

Borrar el archivo devuelve el espacio de inmediato porque el sistema de archivos libera los bloques del inodo. La solución ataca la causa (el archivo que creció) y no el síntoma. Para que no se repita, en un servidor real se configura logrotate con límites de tamaño y rotación, y se alerta cuando el uso pasa de un umbral (por ejemplo 85 %). También se puede reservar espacio para root, que es lo que permite recuperar el control cuando un usuario llena el disco.

Incidente 2 — «No space left on device» con espacio libre

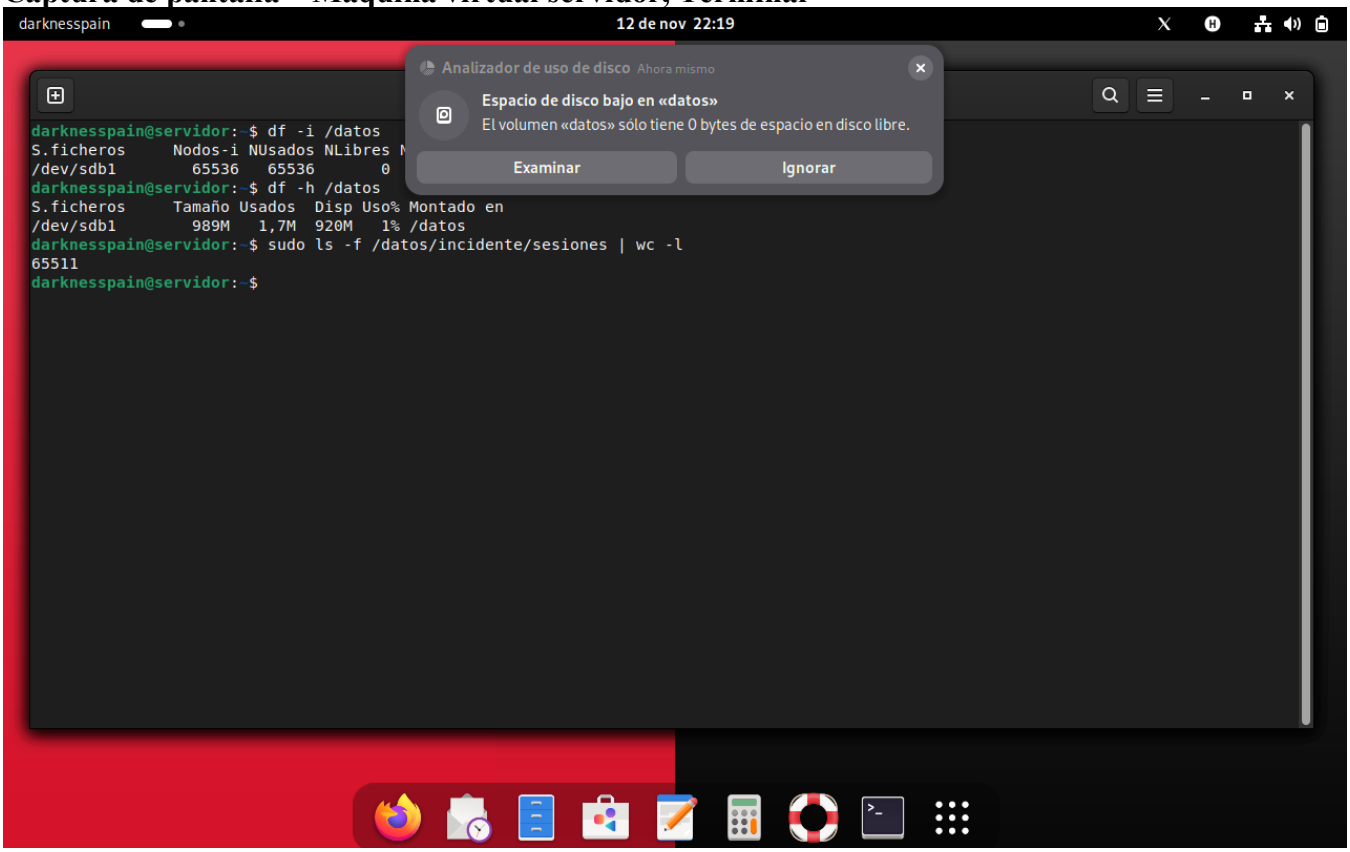
Figura 156
Captura de pantalla – Máquina virtual servidor, Terminal



Nota. «df -i /datos» muestra 65536 inodos, 65536 usados, 0 libres y 100 %; «touch» falla y «df -h /datos» sigue mostrando 921 MB disponibles.

Aquí está el aprendizaje central de la práctica: el mismo mensaje de error, dos causas distintas. El disco tiene espacio de sobra y aun así no se puede crear un archivo, porque cada archivo necesita un inodo y la tabla está llena. El script creó cientos de miles de archivos vacíos, que ocupan casi cero bloques pero un inodo cada uno. Quien solo mira df -h concluye que el disco está bien y no encuentra la falla.

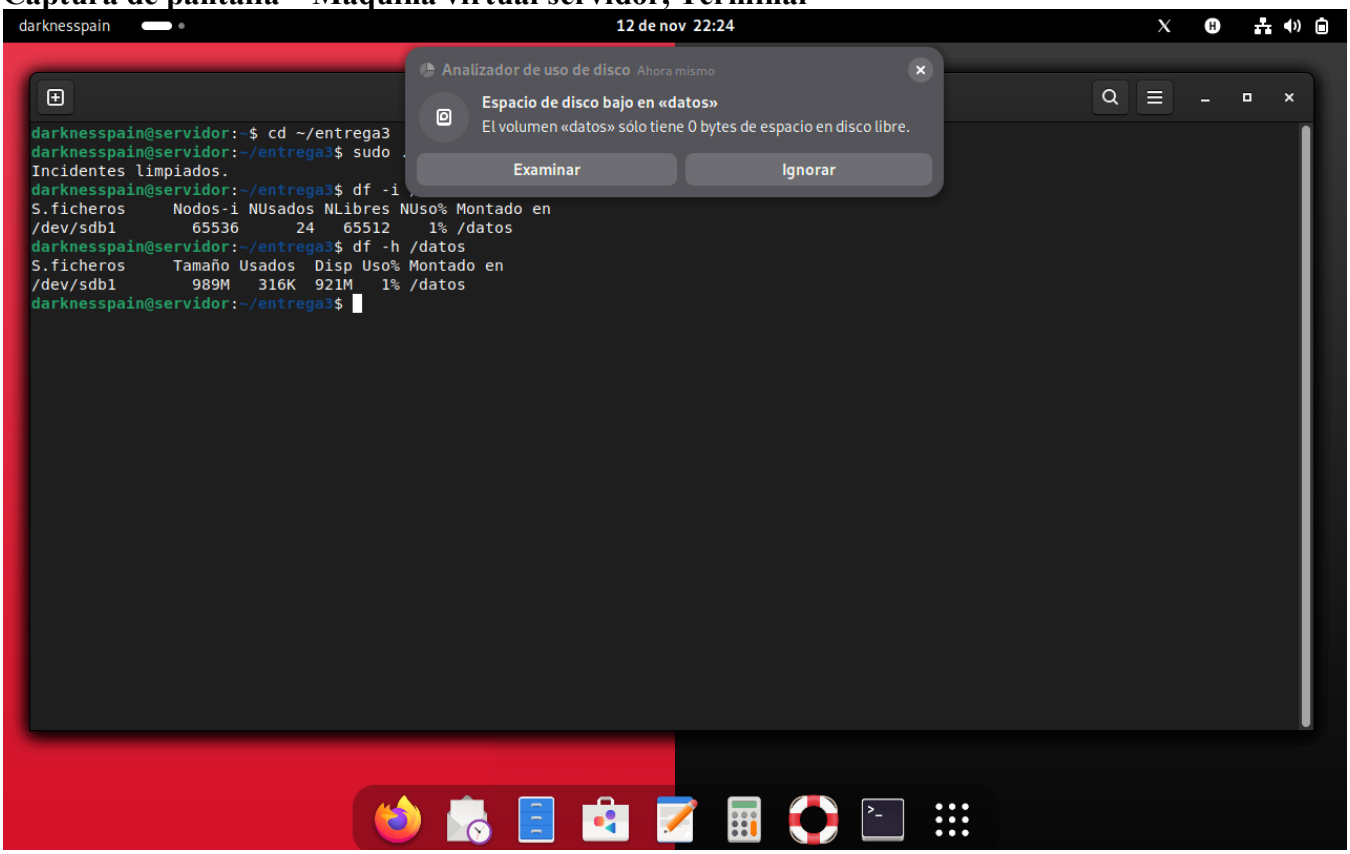
Figura 157
Captura de pantalla – Máquina virtual servidor, Terminal



Nota. «df -i /datos» confirma 65536 de 65536 inodos usados, «df -h» muestra el espacio libre y «sudo ls -f /datos/incidente/sesiones | wc -l» cuenta 65511 entradas en una sola carpeta.

El comando que revela la causa es el conteo de entradas de una carpeta: 65511 nombres en un solo directorio no son un uso normal. ls -f evita ordenar y por eso es rápido con decenas de miles de archivos; con ls normal el listado tardaría y se ordenaría en memoria. La cifra coincide con los inodos consumidos, lo que cierra el diagnóstico: no es espacio, es cantidad de archivos.

Figura 158
Captura de pantalla – Máquina virtual servidor, Terminal

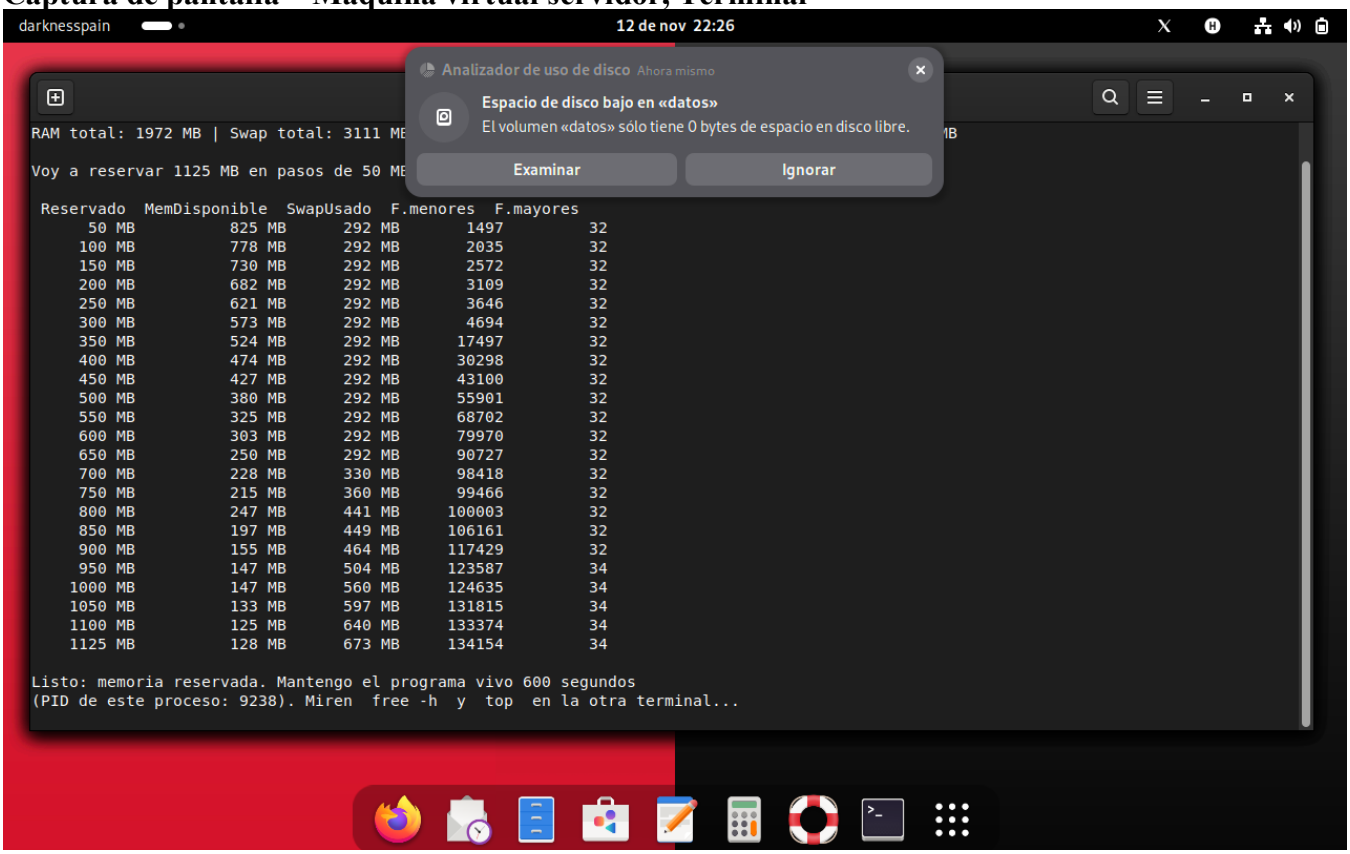


Nota. «sudo ./incidentes.sh limpiar» borra /datos/incidente y «df -i /datos» baja a 24 inodos usados de 65536, con «df -h» en 1 %.

La limpieza devuelve los inodos al liberar cada entrada del directorio. La prevención realista es distinta de la del incidente 1: aquí no sirve limitar el tamaño, hay que limitar la cantidad. Se consigue con cuotas de inodos por usuario, con sistemas de archivos que asignan inodos dinámicamente (xfs, btrfs) o con monitorización de df -i y alertas cuando el porcentaje sube. Un servidor de correo o de sesiones es el caso típico donde este incidente aparece en producción.

Incidente 3 — La memoria se agota

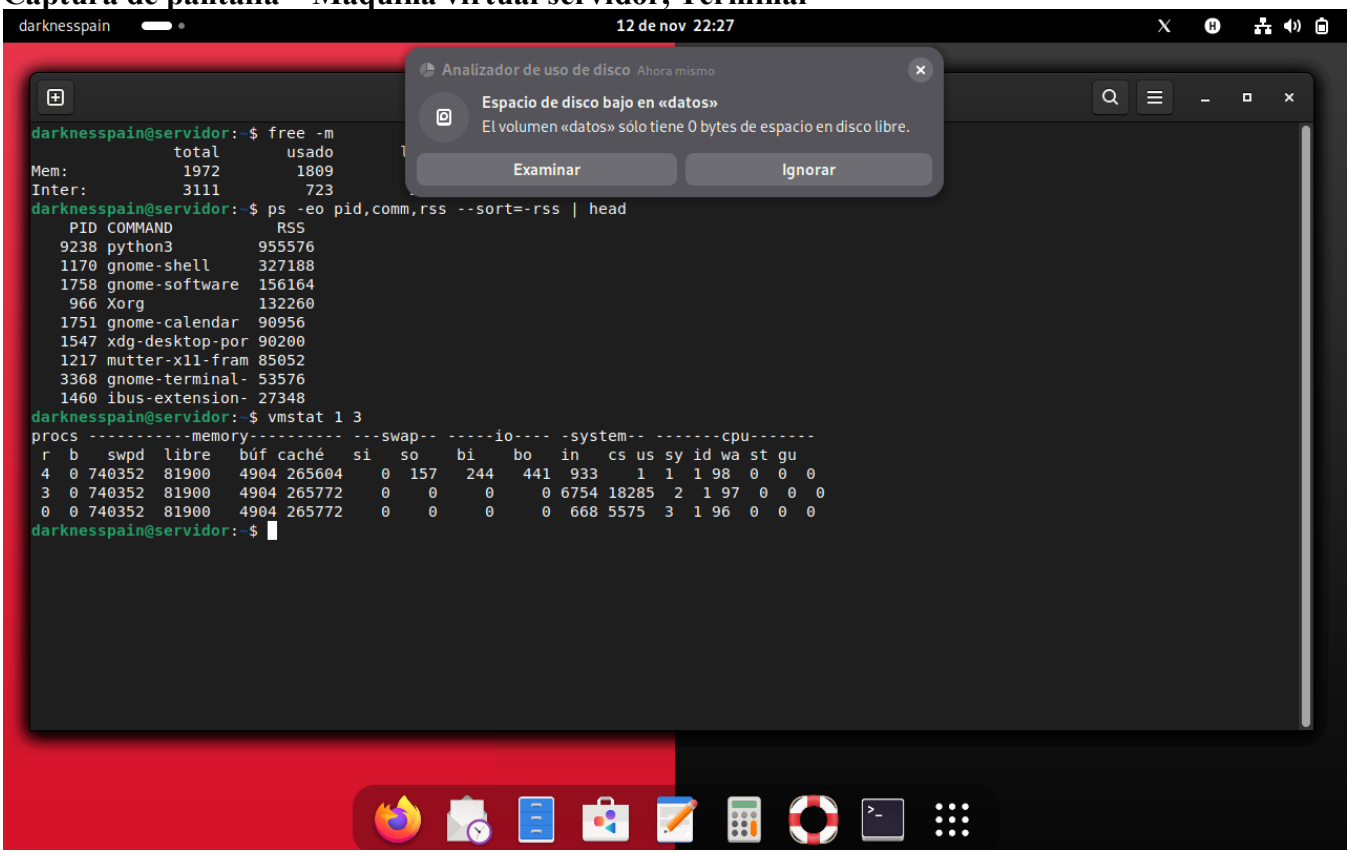
Figura 159
Captura de pantalla – Máquina virtual servidor, Terminal



Nota. El programa se lanzó con MEGAS_A_RESERVAR = 1125 y MANTENER_SEGUNDOS = 600; la tabla muestra cómo la memoria disponible baja hasta el mínimo y el swap empieza a usarse, y el programa queda «manteniendo» la memoria reservada.

El sabotaje no rompe nada: solo pide más memoria de la que hay y la retiene. El sistema no se cae, empieza a intercambiar con el disco, y esa es la razón por la que la máquina se vuelve lenta sin mostrar un error. Para el usuario es un cuelgue; para el administrador es una cifra: la memoria disponible y el swap usado. Ese contraste es el que hay que saber leer en la sustentación.

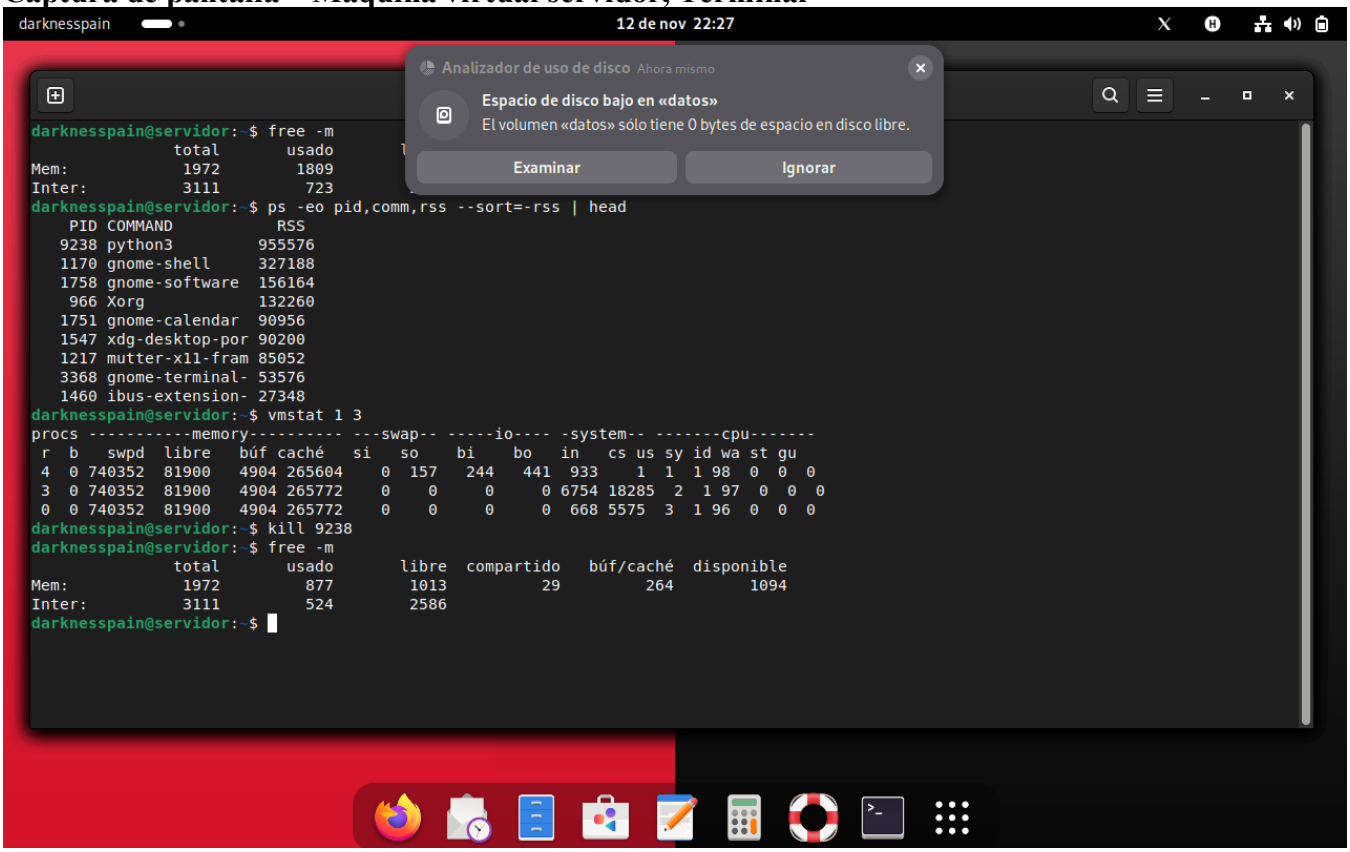
Figura 160
Captura de pantalla – Máquina virtual servidor, Terminal



Nota. En una segunda terminal, «free -m» muestra la memoria casi agotada y el swap en uso, «ps -eo pid,comm,rss --sort=-rss | head» pone a python3 en cabeza con un RSS muy alto y «vmstat 1 3» muestra las columnas si y so distintas de cero.

El diagnóstico se hace desde otra terminal precisamente porque la primera está ocupada por el culpable. ps ordenado por RSS identifica al proceso y su PID; free -m y vmstat confirman que el sistema está paginando. La columna si (swap in) distinta de cero significa que se están leyendo páginas desde el swap, es decir, hiperpaginación: el sistema pasa más tiempo moviendo páginas que ejecutando. Ese es el dato que distingue «mucha memoria usada» de «sistema en problemas».

Figura 161
Captura de pantalla – Máquina virtual servidor, Terminal

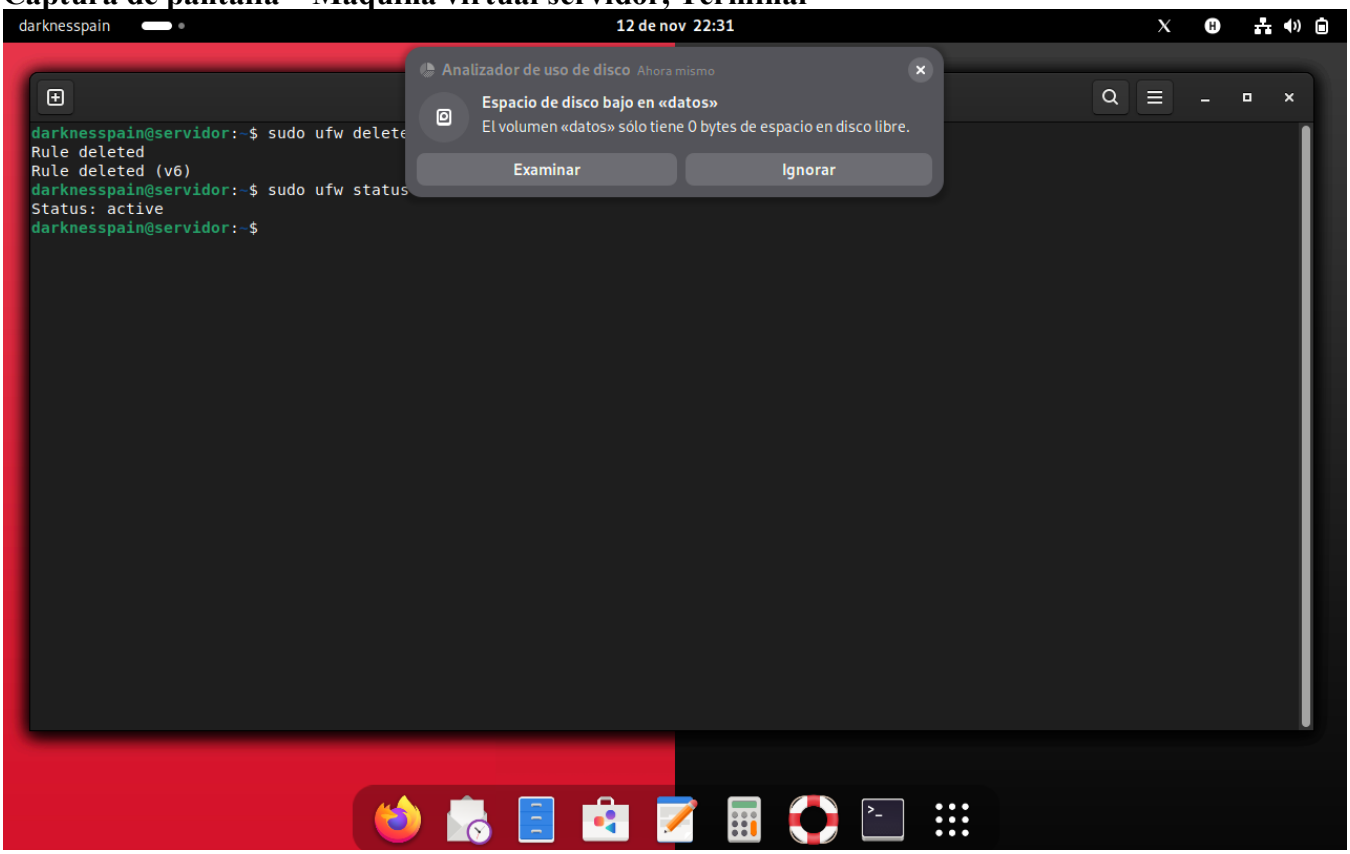


Nota. «kill 9238» termina el proceso y «free -m» muestra la memoria disponible recuperada y el swap bajando.

Matar el proceso devuelve la memoria porque el kernel libera sus páginas cuando el proceso termina. El orden correcto es identificar antes de matar: un kill a ciegas puede tumbar un servicio crítico. La prevención realista es triple: límites por servicio (systemd con MemoryMax), monitorización con alertas antes de llegar al 90 % y aplicaciones que no reserven bloques enteros sin necesidad. En un servidor sin swap, este incidente terminaría con el OOM killer matando algo por su cuenta.

Incidente 4 — El cliente ya no puede entrar por SSH

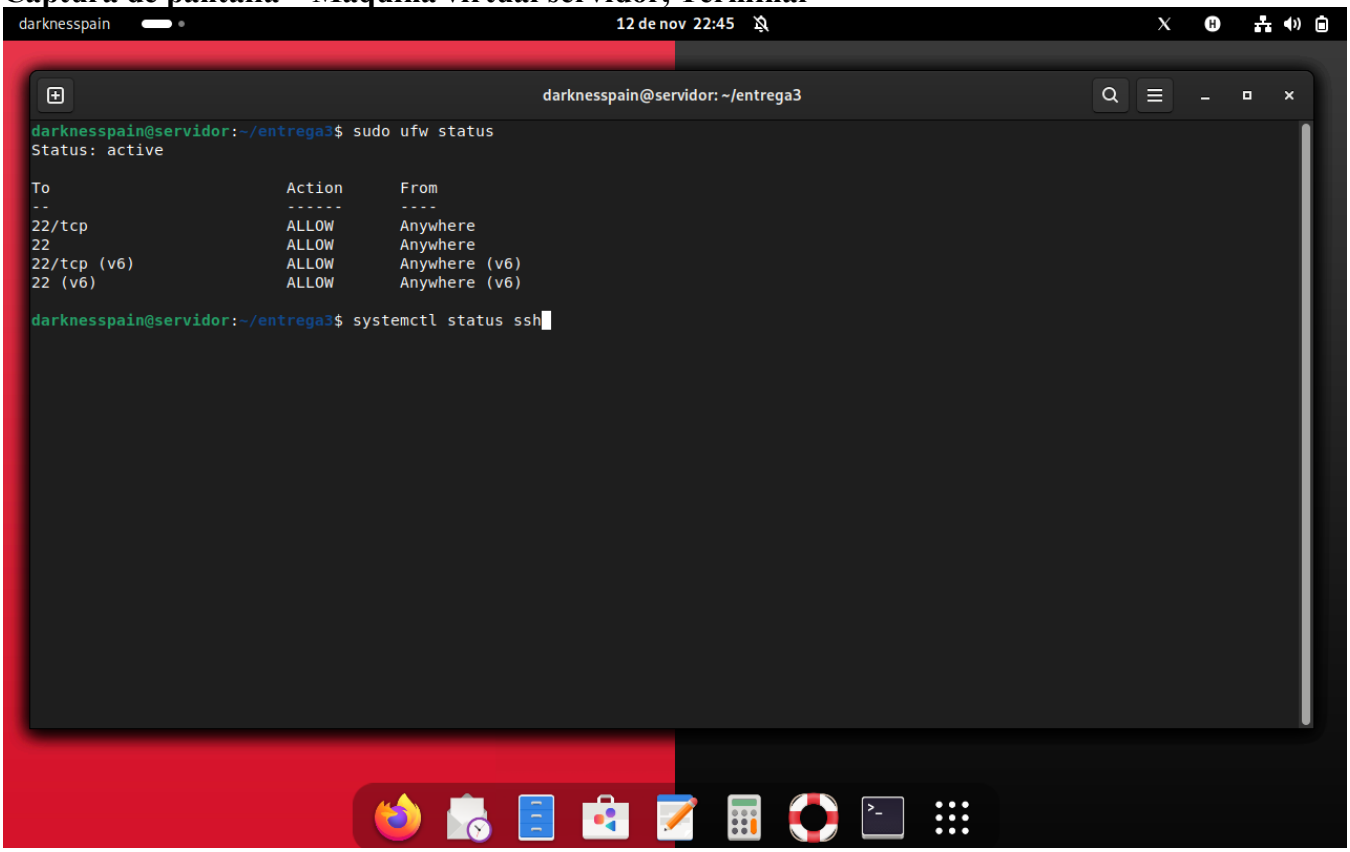
Figura 162
Captura de pantalla – Máquina virtual servidor, Terminal



Nota. «sudo ufw delete allow 22/tcp» responde «Rule deleted» y «sudo ufw status» muestra el cortafuegos activo sin ninguna regla para el puerto 22.

El sabotaje quita la excepción, no el servicio: SSH sigue escuchando en el puerto 22, pero el cortafuegos descarta los paquetes que intentan entrar. Esa distinción es la clave del diagnóstico, porque un administrador que solo mira «systemctl status ssh» concluye que todo está bien y busca el problema en otro sitio. El detalle de que la sesión ya establecida siga viva también enseña cómo funciona el filtrado: las reglas aplican a conexiones nuevas.

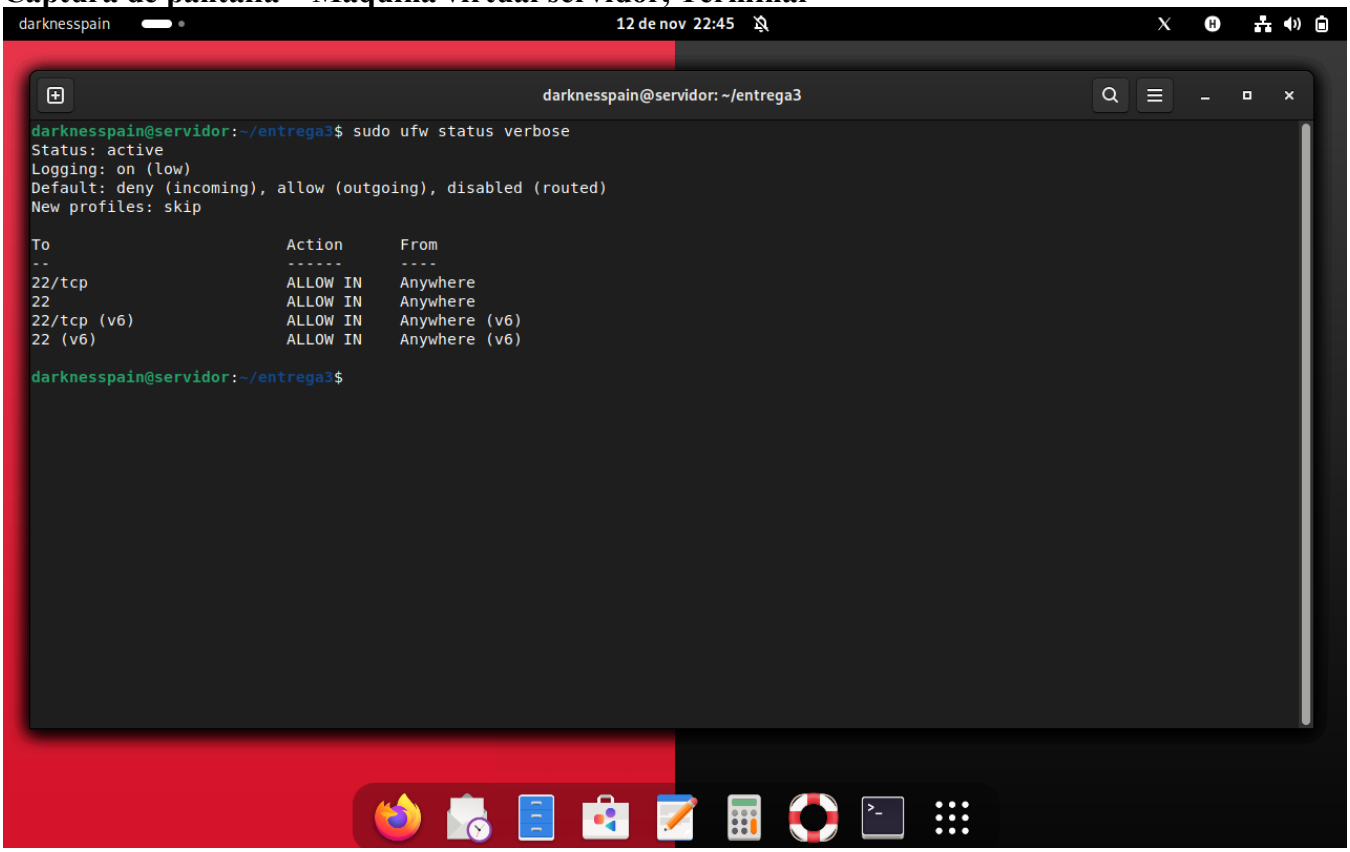
Figura 163
Captura de pantalla – Máquina virtual servidor, Terminal



Nota. «sudo ufw status» confirma que no hay regla para 22 y «systemctl status ssh --no-pager | head -5» muestra el servicio activo y en ejecución.

Estas dos órdenes juntas son el diagnóstico completo hecho desde el servidor: el servicio está bien y el cortafuegos es el que cierra. Si el servicio estuviera caído, el problema sería otro; si ufw estuviera inactivo, también. La conclusión es que la falla está en la capa de filtrado, no en la de aplicación, y eso reduce el problema a una sola regla que hay que volver a crear.

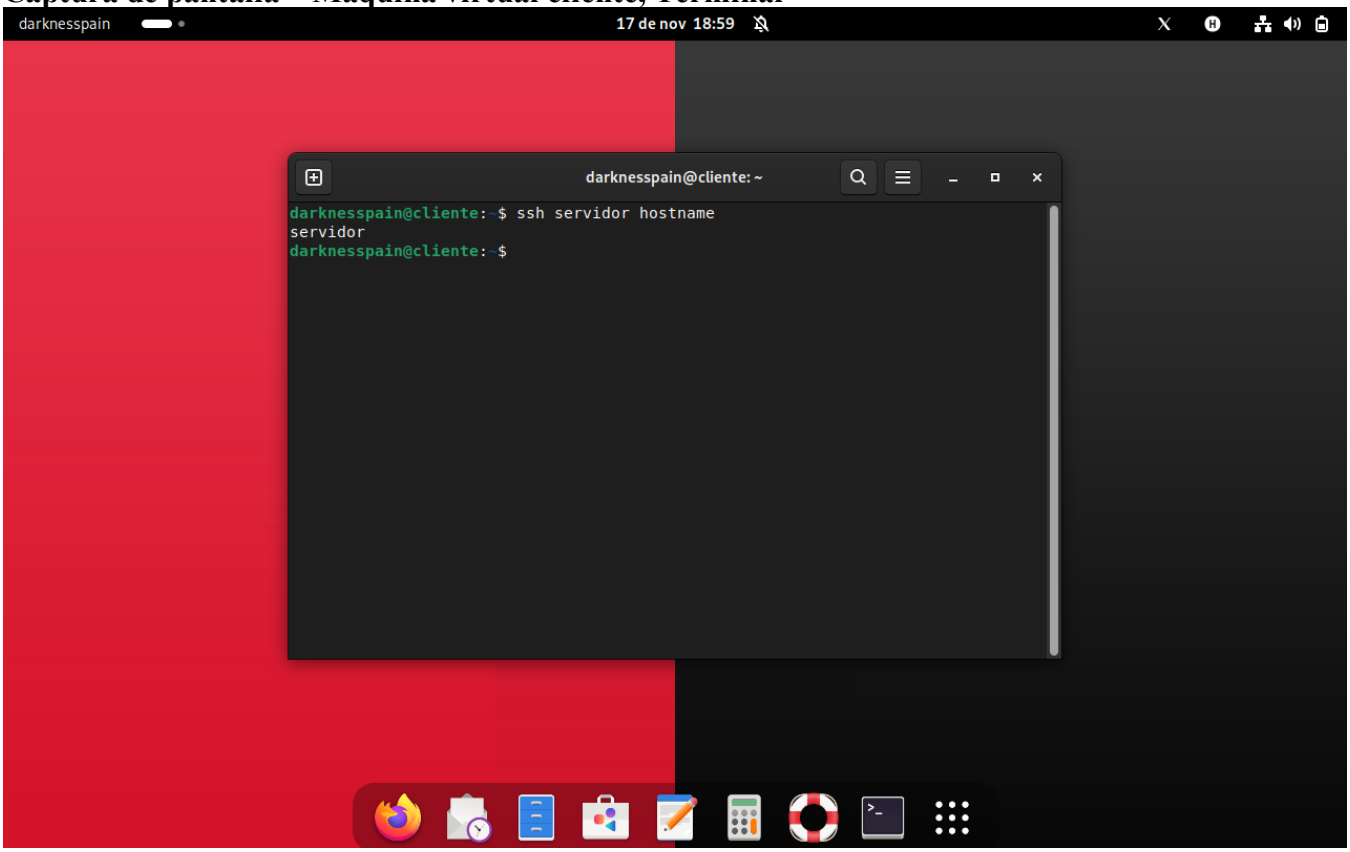
Figura 164
Captura de pantalla – Máquina virtual servidor, Terminal



Nota. «sudo ufw status verbose» muestra de nuevo «Status: active», «Default: deny incoming, allow outgoing» y la regla «22/tcp ALLOW IN» para IPv4 y IPv6.

La solución es volver a permitir el puerto y comprobar que la regla quedó escrita, no solo que el comando no dio error. La prevención es doble: usar la regla antes de activar el cortafuegos (como hicimos en D.1) y tener siempre una segunda vía de acceso —la consola de la máquina virtual— porque un error de este tipo deja al administrador fuera por definición. En la sustentación conviene mencionar que ufw permite además reglas por servicio (ufw allow OpenSSH), que evitan depender del número de puerto.

Figura 165
Captura de pantalla – Máquina virtual cliente, Terminal



Nota. Desde el cliente, «ssh servidor hostname» vuelve a responder «servidor».

La prueba final se hace desde el lado afectado: no basta con que el servidor diga que la regla existe, hay que comprobar que el cliente puede entrar otra vez. Es el mismo criterio que usamos en E.1 al desactivar las contraseñas: se cambia algo y se verifica inmediatamente desde la máquina que lo va a sufrir. Sin esta comprobación, el diagnóstico se queda en una hipótesis.

Tabla 9
Ficha del incidente 1 (disco lleno)

Aspecto	Respuesta
Síntoma	«No space left on device» al crear un archivo en /datos/proyecto
Comandos de investigación	df -h /datos ; df -i /datos ; sudo du -sh /datos/* ; ls -l /datos/incidente/registros
Evidencia clave	df -h: 973M usados de 989M, 0 disponibles; servicio.log de 1019318272 bytes
Causa raíz	Un archivo de registro creció sin límite hasta agotar los bloques
Solución aplicada	sudo rm del archivo; df vuelve a 1 %
Cómo evitarlo	Rotación de logs (logrotate), alertas de uso de disco y reserva para root

Nota. Los datos corresponden a las Figuras 146 a 149.

Tabla 10
Ficha del incidente 2 (inodos agotados)

Aspecto	Respuesta
Síntoma	El mismo «No space left on device», pero con 921 MB libres
Comandos de investigación	df -h /datos ; df -i /datos ; sudo ls -f /datos/incidente/sesiones | wc -l
Evidencia clave	df -i: 65536 de 65536 inodos usados; 65511 entradas en una carpeta

Causa raíz	Cientos de miles de archivos vacíos agotaron la tabla de inodos
Solución aplicada	sudo ./incidentes.sh limpiar; df -i baja a 24 usados
Cómo evitarlo	Cuotas de inodos, sistemas con inodos dinámicos y alertas de df -i

Nota. La comparación entre df -h y df -i es lo que separa este incidente del anterior.

Tabla 11

Ficha del incidente 3 (memoria agotada)

Aspecto	Respuesta
Síntoma	El servidor se vuelve lento y no responde con fluidez; swap en uso
Proceso culpable	python3 con PID 9238, identificado con ps -eo pid,comm,rss --sort=-rss
Evidencia de swap	free -m con swap ocupado y vmstat 1 con si/so distintos de cero
Solución	kill 9238; la memoria disponible se recupera
Prevención	MemoryMax en systemd, monitorización y alertas antes del 90 %

Nota. El diagnóstico se hizo desde una segunda terminal porque la primera estaba ocupada por el proceso.

Tabla 12

Ficha del incidente 4 (SSH bloqueado por ufw)

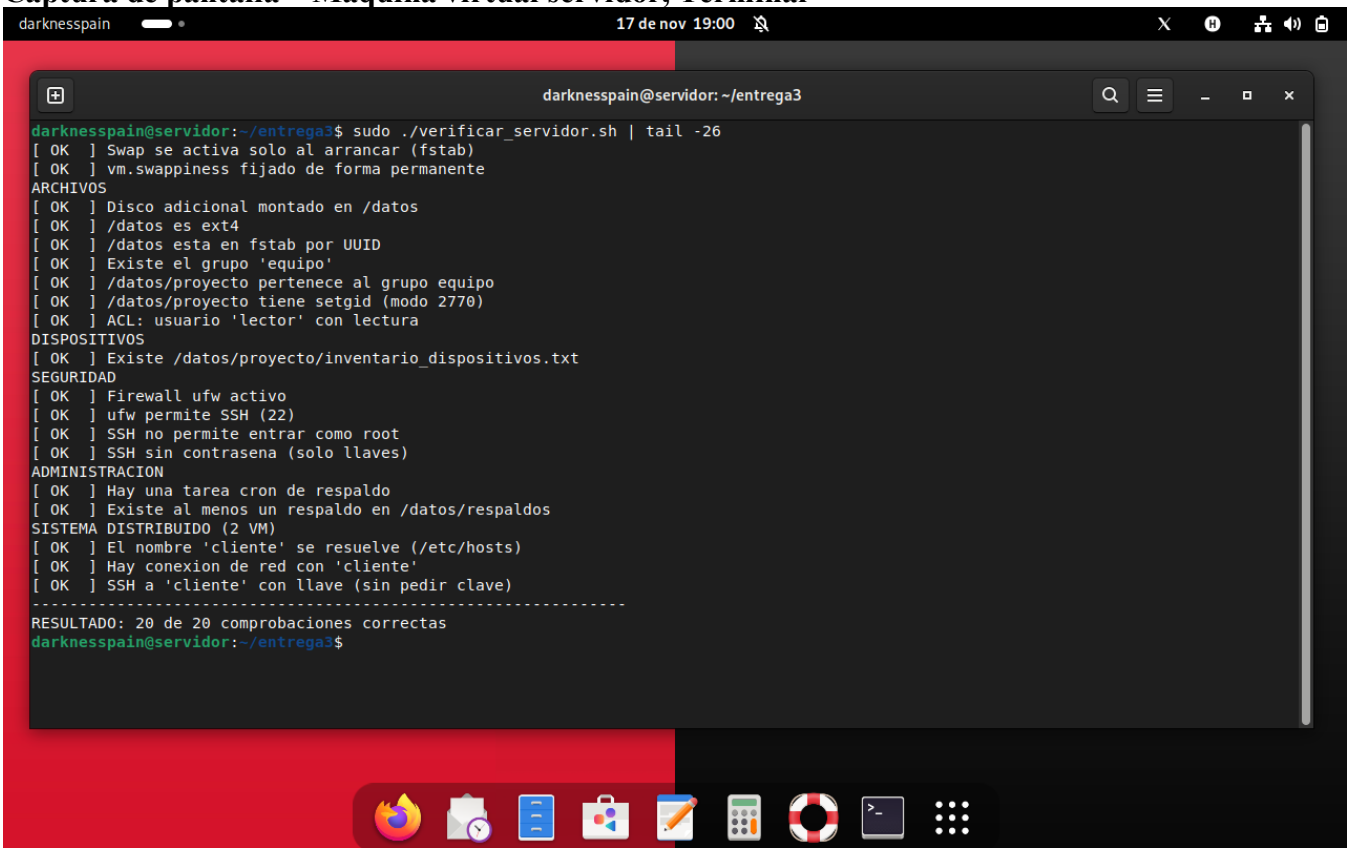
Aspecto	Respuesta
Síntoma	Desde el cliente, la conexión SSH no completa; el ping sí responde
Pruebas que descartaron causas	ping servidor (red bien) ; systemctl status ssh (servicio activo) ; ufw status (sin regla 22)
Causa raíz	Se eliminó la regla que permitía el puerto 22 en el cortafuegos
Solución	sudo ufw allow 22/tcp y verificación desde el cliente
Cómo evitarlo	Permitir antes de activar, reglas por servicio y consola como segunda vía

Nota. La sesión ya establecida siguió funcionando porque ufw filtra conexiones nuevas.

Parte G — Cierre: verificar, comparar y documentar

G.1 Autoevaluación del servidor

Figura 166
Captura de pantalla – Máquina virtual servidor, Terminal

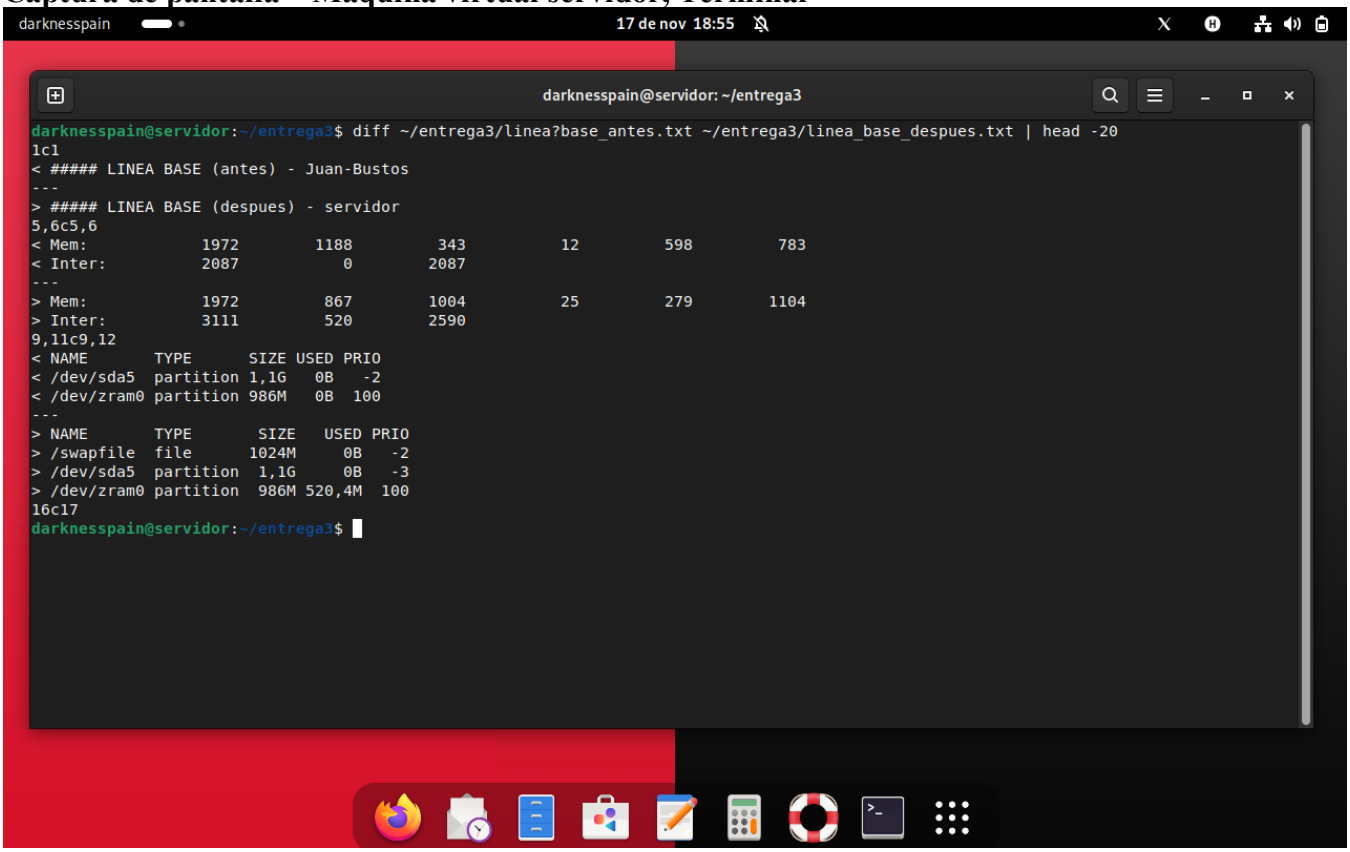


Nota. «sudo ./verificar_servidor.sh» imprime las 20 comprobaciones con «[OK]» en memoria, archivos, dispositivos, seguridad, administración y sistema distribuido, y cierra con «RESULTADO: 20 de 20 comprobaciones correctas».

El verificador no reemplaza la comprensión, pero es una red de seguridad: recorre en un minuto todo lo que la guía pide y avisa si algo quedó a medias. Las veinte líneas cubren los cinco bloques de la entrega, desde el swap hasta la llave hacia el cliente. La primera vez que lo ejecutamos detectó lo que faltaba (el cortafuegos y las llaves) y por eso el resultado final es 20 de 20 y no un aprobado a la primera.

G.2 Comparar con la línea base

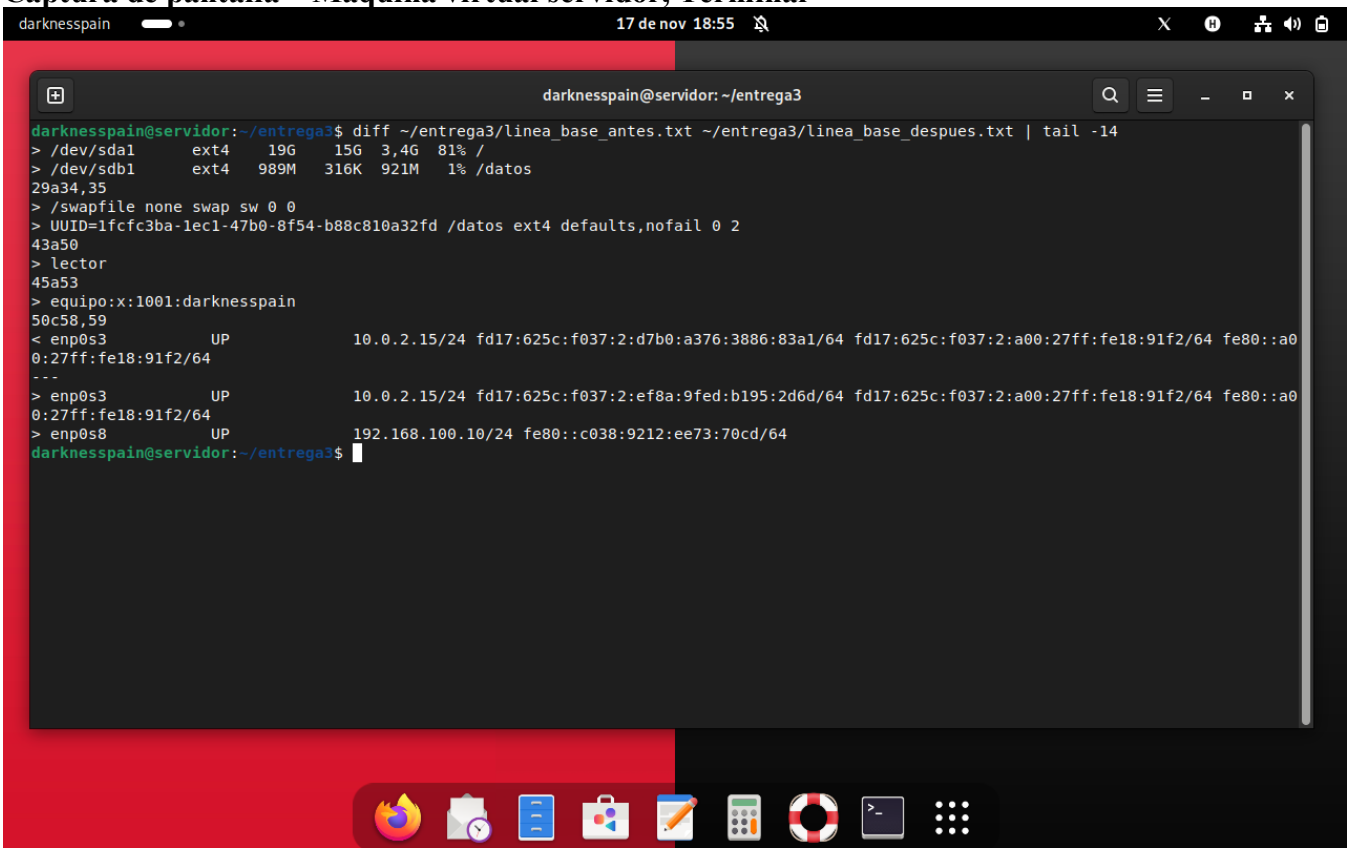
Figura 167
Captura de pantalla – Máquina virtual servidor, Terminal



Nota. «diff ~/entrega3/linea_base_antes.txt ~/entrega3/linea_base_despues.txt | head -20» muestra el comienzo de las diferencias: el nombre de la máquina, la memoria y el swap (3111 MB y tres áreas) y el cambio de tamaño de la raíz.

El diff es la prueba documental de lo que cambió en el servidor. Cada línea con «>» es algo que agregamos y cada «<» algo que ya no está; los cambios de memoria y de hora son ruido normal entre dos mediciones. La cabecera «1c1» confirma que también cambió el nombre de la máquina, de Juan-Bustos a servidor, y la línea de swap muestra que pasamos de 2087 a 3111 MB con tres áreas activas. Para que el comando quede a la vista se limita la salida con head; la cola se ve en la figura siguiente.

Figura 168
Captura de pantalla – Máquina virtual servidor, Terminal



Nota. «diff ... | tail -14» muestra el final de las diferencias: aparece sdb con la partición sdb1 ext4 etiquetada DATOS, las dos líneas nuevas de fstab y los usuarios y grupos agregados (lector y equipo).

La cola del diff completa la historia: un disco nuevo montado por UUID con la opción nofail, el archivo de swap añadido a fstab y el usuario lector con su grupo equipo. Partir la salida en dos capturas no altera la evidencia, porque ambas son el mismo comando sobre los mismos archivos y solo cambia el tramo que se muestra; además permite que la orden quede visible en la pantalla, que es lo que se pide al sustentar. Nada de esto existía en la línea base del primer día.

Tabla 13
Comparación de la línea base

Aspecto	Antes	Después
Swap activo	Partición sda5 (1,1 G) y zram0 (986 M); 2087 MB	Además /swapfile de 1024 MB; 3111 MB en tres áreas
Discos y montajes	sda (20 G) con raíz al 75 %; sr0	Se agrega sdb1 ext4 «DATOS» montado en /datos; raíz al 81 %
Puertos en escucha	Solo 22 (SSH)	Sigue solo 22, ahora filtrado por ufw con la regla 22 permitida
Usuarios y grupos	root, darknesspain	Se agregan lector y el grupo equipo (darknesspain como miembro)
Interfaces de red	enp0s3 (10.0.2.15)	enp0s3 y enp0s8 (192.168.100.10); hostname servidor

Nota. Valores tomados de linea_base_antes.txt y linea_base_despues.txt, que acompañan al informe.

G.3 Manual de operación (runbook)

Datos del sistema. El servidor se llama servidor, tiene 2048 MB de RAM y dos interfaces: enp0s3 en 10.0.2.15 (NAT, solo para instalar paquetes) y enp0s8 en 192.168.100.10/24 (red interna redso). El cliente se llama cliente, tiene 1024 MB y está en 192.168.100.11/24. El usuario administrador es darknesspain (grupo sudo y grupo equipo); existe además el usuario lector, sin privilegios, con acceso de solo lectura a /datos/proyecto. El disco de datos es /dev/sdb (1 GB), con una partición ext4 etiquetada DATOS y UUID 1fcfc3ba-1ec1-47b0-8f54-b88c810a32fd, montada en /datos por /etc/fstab con la opción nofail.

Cómo está protegido. ufw está activo con política «deny incoming, allow outgoing» y una única excepción, el puerto 22. SSH no permite entrar como root y no acepta contraseñas: solo llaves, con una llave autorizada para cada máquina. /datos/proyecto pertenece a root:equipo con modo 2770 (setgid), de forma que los archivos nuevos heredan el grupo equipo, y una ACL concede a lector solo lectura y ejecución, sin escritura. Los respaldos se guardan en /datos/respaldos y se copian al cliente.

Rutinas automáticas. Una tarea de cron del usuario darknesspain ejecuta $HOME/respaldo.sh cada cinco minutos. El script comprime /datos/proyecto en /datos/respaldos/proyecto_AAAAMMDD_HHMMSS.tar.gz, copia esa carpeta al cliente con rsync sobre SSH y conserva solo los cinco archivos más recientes; si la copia remota falla, anota la fecha y el error en /datos/respaldos/errores.log y no interrumpe el respaldo local.

Cómo restaurar un respaldo. Elegir el archivo con «ls -lt /datos/respaldos», crear una carpeta de prueba y extraer: «mkdir -p /tmp/prueba» y «tar xzf /datos/respaldos/proyecto_20261112_220827.tar.gz -C /tmp/prueba»; después comprobar el contenido con «ls -l /tmp/prueba/proyecto». La extracción se hace siempre en otra ruta y nunca encima de la carpeta original, para poder comparar antes de reemplazar. Si el respaldo está en el

cliente, primero se trae con scp y luego se extrae igual.

Tabla 14

Guía de diagnóstico rápido

Síntoma	Primer comando	Qué busco
«No space left on device»	df -h /datos	Uso al 100 %: es el incidente de bloques
El mismo error con espacio libre	df -i /datos	Inodos al 100 %: demasiados archivos
El servidor va lento y no responde	free -m y vmstat 1	Swap en uso y columnas si/so distintas de cero
Un proceso se come la memoria	ps -eo pid,comm,rss --sort=-rss | head	El RSS más alto y su PID para decidir si se detiene
El cliente no entra por SSH	ping servidor y systemctl status ssh	Si la red responde y el servicio está activo, mirar ufw
El respaldo no llega al cliente	cat /datos/respaldos/errores.log	La línea con fecha y «no se pudo copiar»

Nota. Cada fila corresponde a una falla que provocamos y resolvimos en la Parte F.

G.5 Bitácora de comandos

Tabla 15

Bitácora de comandos por funcionalidad

Función	Comando	Qué hace
Identidad del equipo	hostnamectl set-hostname	Cambia el nombre del servidor de forma permanente
Red	nmcli con add / nmcli con up	Crea y activa el perfil de la red interna con IP fija
Red	ip -br addr / ip -br link	Muestra direcciones y estado de las interfaces
Prueba de red	ping -c 3 cliente	Comprueba conectividad con la otra máquina
Memoria	free -m / cat /proc/meminfo	Resumen y contadores crudos de memoria y swap
Memoria	fallocate / mkswap / swapon	Crea, formatea y activa el archivo de intercambio
Memoria	sysctl vm.swappiness	Ajusta la tendencia del kernel a usar swap
Memoria	ps -eo pid,comm,vsz,rss --sort=-rss	Ordena procesos por memoria residente
Discos	lsblk / fdisk / mkfs.ext4	Lista discos, crea la partición y el sistema de archivos
Montajes	mount / mount -a / df -hT / df -i	Monta, verifica espacio por bloques y por inodos
Archivos	ln / ln -s / stat / ls -li	Crea enlaces y muestra inodo y número de enlaces
Permisos	chown / chmod 2770 / setfacl / getfacl	Define dueño, grupo, setgid y ACL
Dispositivos	ls -l /dev / lspci / lsmod / modinfo / dmesg	Identifica tipos de dispositivo, drivers y detección
Interrupciones	cat /proc/interrupts / diff	Compara contadores antes y después de una actividad
Cortafuegos	ufw default / allow / enable / status	Define política, permite SSH y activa el filtrado
SSH	sshd -t / sshd -T / systemctl restart ssh	Valida la configuración y la aplica
Registros	journalctl -p err / journalctl -u ssh / last	Consulta errores, actividad del servicio e ingresos
Llaves	ssh-keygen / ssh-copy-id	Genera el par de llaves e instala la pública en la otra máquina
Administración remota	ssh máquina 'orden' / scp	Ejecuta órdenes y copia archivos a distancia
Respaldos	tar czf / rsync -a / crontab -e	Comprime, copia al otro nodo y programa la tarea
Diagnóstico	du -sh / ls -f | wc -l / kill	Encuentra la carpeta pesada, cuenta entradas

		y detiene procesos
Verificación	verificar_servidor.sh / linea_base.sh / diff	Autoevalúa la configuración y compara estados

Nota. Todos los comandos fueron ejecutados en las máquinas virtuales durante esta entrega.

Conclusiones de la entrega

El servidor quedó operable, seguro y documentado: memoria administrada con un área de intercambio propia y una política de swappiness explícita, un disco de datos montado por UUID con permisos de grupo y ACL, un cortafuegos con una sola puerta abierta, SSH sin root y sin contraseñas, respaldos automáticos cada cinco minutos con copia en la segunda máquina y un verificador que responde 20 de 20. El recorrido no fue lineal: hubo que repetir capturas, rehacer el respaldo cuando el grupo equipo no estaba activo en la sesión y diagnosticar cuatro fallas provocadas, una de ellas del mismo tipo que pueden dejar a un administrador fuera de su servidor. Ese es, al final, el aprendizaje central: un servidor no es el que no falla, sino el que se puede diagnosticar, reparar y reconstruir cuando falla.

Preguntas de reflexión para las conclusiones

La guía de la práctica cierra con ocho preguntas que obligan a mirar la entrega completa y no un solo apartado. Las respondemos con los números que quedaron registrados en las figuras y en los archivos de evidencia, no con lo que dicen los libros.

- **Pregunta 13. ¿Por qué un proceso puede tener un espacio de direcciones virtual mucho mayor que la memoria física instalada y cómo lo resuelven la paginación y el swap?**

Porque el espacio de direcciones es una promesa, no una reserva. Cuando un proceso pide memoria, el kernel le entrega un rango de direcciones virtuales y anota que esas páginas existen, pero no le da marcos de RAM hasta que el proceso las toca; eso es la paginación por demanda. En nuestra máquina se veía a simple vista: la memoria instalada era de 1972 MB y el VSZ de gnome-shell rondaba los 5,3 GB, es decir, el escritorio «creía» tener más memoria que la máquina entera. La traducción de dirección virtual a marco la hace la MMU con las tablas de páginas y el TLB.

Cuando la RAM se llena, el kernel elige páginas víctimas con su algoritmo y las escribe en el área de intercambio para liberar marcos; si el proceso vuelve a pedirlas, se leen del swap y se cuenta un fallo mayor. Eso es exactamente lo que medimos en el experimento A.3: con 1125 MB reservados, el swap subió a 446 MB y aparecieron fallos mayores; con swappiness 60 el swap usado fue de 385 MB con 26 fallos mayores, y el sistema siguió respondiendo, solo más lento. El precio del swap es el tiempo: un fallo mayor cuesta órdenes de magnitud más que uno menor porque implica una lectura de disco.

- **Pregunta 14. Con los resultados de FIFO, LRU y Óptimo, ¿qué algoritmo recomendarían para un sistema real y por qué el Óptimo no se puede implementar? ¿Qué enseñó la anomalía de Belady?**

En el simulador, con la cadena de la guía, FIFO produjo 9 fallos con 3 marcos y 10 con 4; LRU bajó a 7 y 8; el Óptimo dio 6 en los dos tamaños. Recomendaríamos LRU o su versión barata en el kernel real, el reloj de segunda oportunidad, porque cumple la propiedad de pila: al aumentar los marcos nunca empeora, y se puede aproximar mirando el bit de referencia que ya mantiene el hardware. El Óptimo no se puede implementar porque exige conocer el futuro: para decidir qué página expulsar tendría que saber cuál se va a usar más tarde, y eso solo se sabe después; sirve como cota de comparación.

La anomalía de Belady enseñó que «más memoria» no garantiza «menos fallos» si el algoritmo es malo: con FIFO pasamos de 9 a 10 fallos al subir de 3 a 4 marcos. Es la razón por la que un administrador no debe limitarse a ampliar RAM cuando un servidor va lento, sino revisar primero qué está paginando y con qué política; y es también el argumento de los libros para preferir algoritmos de pila como LRU.

- **Pregunta 15. Un disco muestra espacio libre pero no deja crear archivos. ¿Qué dos causas distintas descubrieron en los incidentes y cómo se distinguen?**

Descubrimos dos causas que se parecen en el síntoma y son completamente distintas por dentro. La primera fue quedarse sin bloques: el archivo de registros servicio.log llegó a 1 019 318 272 bytes, «df -h /datos» marcó 100 % y cualquier escritura respondía «No space left on the

device». La segunda fue quedarse sin inodos: la carpeta de incidentes acumuló 65 511 entradas hasta agotar los 65 536 inodos del sistema de archivos, y ahí «df -h» seguía mostrando espacio libre (1 %) mientras «df -i» marcaba 100 % y touch también fallaba.

Se distinguen con una sola orden: «df -i» informa inodos usados y libres, y «df -h» informa bloques. Si df -h está al 100 % el problema son datos; si df -h tiene espacio y df -i está al 100 % el problema son millones de archivos diminutos. Para confirmarlo usamos «ls -f» y «du» sobre la carpeta sospechosa, y para resolverlo borrar el archivo grande o limpiar las entradas sobrantes, respectivamente.

- **Pregunta 16. ¿Qué ventaja ofrece la herencia de grupo (setgid) y las ACL frente a dar permisos amplios con chmod? Den un ejemplo de su servidor.**

La herencia de grupo hace que los archivos nuevos de una carpeta nazcan ya con el grupo dueño de la carpeta, sin que cada usuario tenga que hacer chgrp después. Eso lo vimos en /datos/proyecto: con setgid y modo 2770, el archivo mi_archivo.txt creado por nosotros quedó con el grupo equipo y no con nuestro grupo principal, y cualquier integrante del equipo puede leerlo y escribirlo sin depender de que el creador se acuerde de cambiar el grupo.

Las ACL permiten conceder permiso a un usuario concreto sin abrir la carpeta a «otros». En el servidor dimos a lector «r-x» sobre /datos/proyecto con una ACL y dejamos «otros» en cero; con chmod la única alternativa habría sido dar permiso al mundo entero. La diferencia práctica es la granularidad: chmod reparte entre dueño, grupo y todos, mientras que la ACL nombra personas. El costo es que hay que documentarla, porque un «ls -l» normal no la muestra y hace falta «getfacl» para leerla.

- **Pregunta 17. Expliquen con un ejemplo la diferencia entre encuesta e interrupciones y qué papel cumple el driver entre el hardware y las aplicaciones.**

Encuesta es preguntar una y otra vez si hay algo, aunque no haya nada; interrupción es que el dispositivo avise cuando ocurre un evento. En la vida diaria, encuesta es mirar el timbre cada diez segundos y la interrupción es que suene. En el servidor lo vimos con la red: la tarjeta

avisa por interrupción cuando llega un paquete y el procesador deja de gastar ciclos preguntando. La encuesta se prefiere cuando el evento es muy frecuente y predecible, porque evita el costo de atender una interrupción por cada evento; en un servidor con tráfico constante, los kernels modernos combinan las dos cosas con interrupciones moderadas (NAPI).

El driver es el traductor: conoce los registros del dispositivo y expone una interfaz uniforme al sistema operativo. Nuestra tarjeta es una Intel 82540EM emulada por VirtualBox y su driver es el módulo e1000, que se puede cargar y descargar con modprobe; por debajo habla con el hardware y por arriba entrega paquetes a la pila de red, de modo que ping o ssh no necesitan saber qué tarjeta hay. Sin driver, el hardware existe pero el sistema no lo puede usar.

- **Pregunta 18. Al caer el cliente, el servidor siguió funcionando. ¿Qué características de un sistema distribuido se observaron y cuáles faltan en este montaje?**

Se observó la tolerancia a fallos en su forma mínima: apagamos el cliente y el servidor siguió prestando su servicio, porque cada máquina tiene su propio kernel y su propia memoria y no dependen de que la otra esté viva para ejecutar. También se vio la independencia de fallos que da la red interna: el ping dejó de responder y el ssh se quedó esperando, pero el resto del servidor no se enteró; y quedó claro que el reloj y el estado no se comparten solos, hay que sincronizarlos.

Faltan las piezas que hacen que un sistema distribuido sea algo más que dos máquinas conectadas: no hay réplica del servidor, así que si cae el servidor el servicio se detiene; no hay detector de fallos ni heartbeat que avise que el cliente desapareció; no hay quórum, consenso ni elección de líder; y el respaldo se guarda en el mismo equipo cliente, así que un incendio se lleva las dos copias. Para la entrega bastaba demostrar que el servicio sobrevive a la caída del cliente, pero un montaje real necesitaría redundancia y un monitor externo.

- **Pregunta 19. ¿Qué cambió entre su línea base de la Entrega 1 y el servidor final? ¿Qué parte de esa diferencia mejora la seguridad y cuál la disponibilidad?**

La línea base mostraba un Debian recién instalado: un solo disco, sin área de intercambio propia (solo el zram de 986 MB), sin cortafuegos y con SSH aceptando contraseña. El servidor

final tiene 3111 MB de swap en tres áreas, un disco de datos montado por UUID con la opción nofail, la carpeta /datos/proyecto con setgid y ACL, ufw activo con una sola regla para el puerto 22, sshd sin acceso de root y sin contraseñas, respaldos automáticos cada cinco minutos con copia en el cliente y un verificador que responde 20 de 20. El diff de la línea base lo muestra línea por línea.

Mejoran la seguridad el cortafuegos con una sola puerta, el endurecimiento de SSH y los permisos con grupo y ACL, porque reducen la superficie de ataque y el alcance de un usuario comprometido. Mejoran la disponibilidad el área de intercambio y la política de swappiness, el montaje con nofail para que un disco ausente no impida arrancar, los respaldos automáticos y el verificador, porque permiten seguir funcionando y detectar a tiempo lo que se rompió. La lección es que seguridad y disponibilidad no son opuestas: cada cambio bien hecho aporta a una y no debería quitarle a la otra.

- **Pregunta 20. Relacionen esta entrega con las anteriores: ¿cómo se conectan la administración de procesos (Entrega 2) y la gestión de memoria y discos al diagnosticar un servidor lento?**

Cuando un servidor va lento, las tres cosas aparecen juntas. El primer síntoma suele ser un proceso que consume: en la Entrega 2 aprendimos a leerlo con ps y top, a mirar PPID, estado y tiempo de CPU, y a terminarlo con kill; en esta entrega el mismo proceso de Python que reservaba 1125 MB apareció con RSS alto y nos obligó a mirar el swap. La cadena de diagnóstico es siempre la misma: carga alta, se busca el proceso, se mira su memoria real, se comprueba si el sistema está paginando y se revisa el disco, porque un disco lleno o sin inodos también hace que todo parezca lento.

El incidente de memoria lo confirmó: el proceso 9238 tenía la memoria agotada, el sistema había bajado parte de su memoria al swap y el servidor respondía con retardo; matarlo devolvió la normalidad sin reiniciar. Los incidentes de disco se diagnosticaron con df, du y df -i, las mismas herramientas de la Entrega 2 para ver el estado del sistema. Por eso la conclusión de

las tres entregas es una sola: administrar un servidor es saber preguntar en el orden correcto —qué proceso, cuánta memoria, qué disco— y no adivinar.

Referencias

Red Hat. (2026, August 6). Hipervisor: ¿qué es, ¿cómo funciona y qué tipos hay? https://www.redhat.com/es/topics/virtualization/what-is-a-hypervisor

Scale Computing. (2025, January 14). Virtual machine advantages and disadvantages. https://www.scalecomputing.com/resources/understanding-virtual-machine-advantages-and-disadvantages

Flinders, M., & Smalley, I. (s. f.). ¿Qué es el kernel de Linux? IBM. https://www.ibm.com/mx-es/think/topics/linux-kernel

IBM. (s. f.). ¿Qué es un sistema operativo? IBM Think. https://www.ibm.com/mx-es/think/topics/operating-systems

IBM. (s. f.). Herramientas de diagnóstico (Linux y UNIX). IBM Documentation. https://www.ibm.com/docs/es/db2/11.5.x?topic=tools-diagnostic-linux-unix