Bitácora Semanal: Administración de Sistemas Unix

Equipo Alpine White

[2026-02-09 Mon]

Contents

1 Información General

- **Semana:** Del 3 de Febrero 2026 al 6 de Febrero 2026
- **Equipo:**
  - Arreguín Salgado Gael Emiliano
  - Gramer Muñoz Omar Fernando
  - López Pérez Mariana
  - Nieto Gallegos Isaac Julián

En esta nuestra primera semana de clases, el enfoque fue particularmente en entender el arranque del sistema operativo, el sistema de archivos y comenzar a prepararnos para elegir la distribución con la que iremos trabajando en el semestre.

2 GRUB Y SISTEMAS DE ARRANQUE

2.1 Conceptos nuevos

2.1.1 Proceso de arranque

- Ejecucion de firmware
- Localizacion de la EFI System Partition
- Carga del bootloader
- Selección y carga de kernel
- Inicio del sistema operativo

2.1.2 Firmware definicion

- BIOS y UEFI (y sus diferencias)

2.1.3 Bootloader / Gestores de arranque definicion

1. GRUB GRUB (GNU GRand Unified Bootloader) es un **cargador de arranque**; es el programa que aparece al iniciar y se encarga de cargar el sistema operativo (o una ISO) y pasarle el control.
2. Windows boot manager
3. Secure boot

   MOK database
4. Ventoy

   Ventoy es una herramienta que nos permite poder arrancar varias imágenes ISO desde una misma memoria USB. Se instala sobre la memoria USB, y recursivamente busca las imágenes ISO que contenga para mostrarlas en el menú de arranque. Se nos queda de tarea para la próxima semana traer una memoria USB con Ventoy instalado.

- **Ventoy y su GRUB:** Ventoy no usa exactamente el mismo GRUB que el del sistema instalado; incluye y arranca con su propia varainte de **GRUB2** para poder gestionar el menú y el arranque de múltiples ISOs desde una sola USB.
- **Consola de GRUB en Ventoy:** En el menú de Ventoy/GRUB, al presionar la tecla `C` se abre la **consola** (tipo terminal) de GRUB.
  - En esta consola podemos inspeccionar y explorar qué ve el bootloader (discos/particiones/archivo y consultar variables o configuración.
  - Comandos:

```
ls
help
set
ls (hd0,gpt1)/
cat (hd0,gpt1)/boot/grub/grub.cfg
```

– Esto sirve para tipo depurar, inspeccionar por qué algo no arranca como debe, confirmar rutas, o entender cómo Ventoy/GRUB está resolviendo la configuración.

2.2 Conceptos de repaso

- **Proceso de arranque (boot process):** Secuencia general: BIOS/UEFI → bootloader (GRUB) → kernel → init/systemd. Punto clave: el sistema operativo **no existe** mientras estamos en GRUB; solo existe firmware con bootloader.
- **Bootloader vs Kernel:** GRUB no es el sistema operativo ni el kernel. Su única responsabilidad es localizar, cargar el kernel (y el initramfs) en memoria y cederle el control. No ejecuta procesos ni gestiona recursos del sistema.
- **Separación de responsabilidades:** Ventoy delega completamente el arranque a GRUB y evita modificar el disco. Ejemplo de buen diseño: cada componente hace una sola cosa bien definida.
- **Acceso a disco antes del kernel:** GRUB puede leer sistemas de archivos (FAT, EXT, ISO) sin drivers del kernel.
- **Lectura de archivos sin montaje:** El bootloader accede directamente a estructuras del sistema de archivos. Montar un FS es una abstracción que pertenece al kernel, no al bootloader.

2.3 Máximas

2.3.1 Todo es un archivo

En los sistemas derivados de Unix, como Linux

2.4 Sección libre

- El comando ls nos permite listar los contenidos del directorio en el que nos encontramos actualmente. Usando los parámetros -la, podemos colocarlos en forma de lista y mostrar archivos ocultos para tener una vista más general y completa del directorio. Por ejemplo:

```
ls -la
```

```
total 464
drwxr-xr-x 1 mrtaichi mrtaichi    158 Feb 18 21:37 .
drwxr-xr-x 1 mrtaichi mrtaichi    186 Feb 18 21:37 ..
-rw-r--r-- 1 mrtaichi mrtaichi  23022 Feb 18 21:45 bitacora1.org
-rw-r--r-- 1 mrtaichi mrtaichi 259061 Feb 18 21:06 bitacora1.pdf
-rw-r--r-- 1 mrtaichi mrtaichi  35070 Feb 18 21:06 bitacora1.tex
-rw-r--r-- 1 mrtaichi mrtaichi   9900 Feb 18 21:06 bitacora2.org
drwxr-xr-x 1 mrtaichi mrtaichi    166 Feb 18 21:46 .git
-rw-r--r-- 1 mrtaichi mrtaichi     25 Feb 18 11:41 test.txt
-rw-r--r-- 1 mrtaichi mrtaichi 133687 Feb  7 12:15 ventoy_boot.jpg
```

- Existen dos formas de referenciar un archivo en Linux: Mediante su ruta relativa y mediante su ruta absoluta. Generalmente una ruta relativa empieza por un `./` o `../`, mientras que una ruta absoluta empieza por `/`, ya que es la ruta completa al archivo desde la raíz del sistema de archivos. Por ejemplo:

```
echo este es el mismo archivo > test.txt
cat /home/mrtaichi/Documents/escuela/adminUnix/test.txt # Ruta absoluta
cat ./test.txt # Ruta relativa


este es el mismo archivo
```

---

3 Disponibilidad del sistema: RAID y puntos únicos de falla

3.1 Conceptos nuevos

3.1.1 Alta disponibilidad

3.1.2 Redundancia

Concepto estrechamente relacionado con Punto único de falla. Es la respuesta directa a solucionar este problema. Siempre existe un componente listo para tomar el lugar y mantener el sistema funcionando.

3.1.3 Tolerancia a fallos

3.1.4 Punto único de falla (Single Point of Failure)

Cuando una de las partes del sistema es tan crítica que su fallo puede arriesgar la integridad de todo el sistema.

3.1.5 Técnicas de Protección de datos

1. Replicación
2. Paridad
3. Backups vs redundancia

3.1.6 RAID (Redundant Array of Independent Disks)

1. Concepto general
2. Objetivos: rendimiento vs seguridad
3. RAID por hardware vs RAID por software
4. Niveles de RAID
   (a) RAID 0

(b) RAID 1

(c) RAID 5

(d) RAID 6

(e) RAID 10

3.2 Conceptos de repaso

- **Proceso de arranque:** La disponibilidad del sistema empieza desde el encendido; si falla el firmware, el bootloader o el kernel, no hay sistema. RAID protege la capa de almacenamiento donde viven los datos.
- **Sistemas de archivos:** GRUB puede leer sistemas de archivos (FAT, EXT, ISO) en disco; los datos que la redundancia protege residen en particiones y FS.
- **Todo es un archivo:** Máxima ya vista en GRUB; los dispositivos que RAID agrupa se representan como archivos en /dev.

3.3 Máximas

3.3.1 Todos los Linux son iguales

En cualquier distribución de Linux podemos -a final de cuentas- hacer cualquier cosa. Todas funcionan esencialmente para lo mismo, y aunque evidentemente hay diferencias, se puede trabajar con ellas sin problema.

3.4 Sección libre

- Estuvimos hablando de la importancia de la redundancia para preservar la información. Como ejemplo usamos el sistema RAID, que utiliza varios discos duros para crear replicación de información y tolerancia a fallos.
- Generalmente la emulación es más lenta que la virtualización. El primer sistema de WSL usaba emulación de Linux, ahora el segundo utiliza virtualización.
- Distrowatch es un sitio web en el que se pueden observar todas las distribuciones de Linux y muchos datos acerca de ellas, incluyendo un ranking de cuáles son las distros más usadas.
- SCSI (que se pronuncia “scuzzy”) es una interfaz de intercambio de datos que se utiliza sobre todo en servidores y computadoras de alto desempeño. Hablando del sistema RAID, es generalmente sobre esta interfaz sobre la que se construyen varios controladores RAID que están en la industria.

---

4 De Unix a GNU/Linux: Origen, Filosofía y Evolución del Software Libre

4.1 Conceptos nuevos

4.1.1 Origen y evolución de Unix

1. Contexto académico
2. Nacimiento de Unix (1969)

   Unix nace en 1969 como un sistema operativo desarrollado en Bell Labs, al principio con fines académicos y distribuido para varias universidades. De ahí nace BSD cuando se vuelve comercial.
3. BSD

   Berkeley System Distribution. Nace como un Unix con código fuente original de Unix. De el nacen distribuciones todavía vivas hasta el día de hoy.
4. Linus Torvalds

4.1.2 Sistema Unix-like

1. Diferencia entre Unix y Unix-like Un sistema Unix es aquel que usa código fuente de Unix. Un Unix-like es aquel que está hecho para comportarse como Unix pero no usa código de Unix. Linux es Unix-like

4.1.3 Filosofía Unix (simplicidad, modularidad, herramientas pequeñas que hacen una sola cosa)

4.1.4 Proyecto GNU

1. Filosofía del software libre
2. Herramientas GNU (gcc, bash, coreutils)
3. Licencia GPL

4.1.5 GNU/Linux como sistema completo

1. Integración del kernel Linux con herramientas GNU
2. Concepto de distribución (Ubuntu, Debian, etc.)

4.1.6 Comercialización de Unix

1. Versiones propietarias
2. Impacto en la industria

4.2 Conceptos de repaso

- **Kernel:** En GRUB vimos que el bootloader carga el kernel; aquí vemos el origen de ese kernel (Linux, Linus Torvalds) y su integración con GNU.
- **Distribuciones:** Ya mencionadas al hablar de elegir distro (Información general) y en RAID (Distrowatch, "todos los Linux son iguales").
- **Filosofía de una sola cosa:** En GRUB repasamos la separación de responsabilidades (Ventoy, GRUB); la filosofía Unix (herramientas pequeñas que hacen una cosa) es la misma idea.

4.3 Máximas

4.4 Sección libre

---

5 Kernel

5.1 Conceptos nuevos

5.1.1 Definición de kernel

1. Intermediario entre hardware y software

5.1.2 Rol del kernel

1. Gestión de CPU, memoria y dispositivos
2. Planificación (scheduling)
3. Gestión de memoria
4. Control de dispositivos
5. Manejo de interrupciones

5.1.3 Llamadas al sistema (System Calls)

1. Interfaz entre espacio de usuario y kernel
2. Ejemplos: fork(), exec(), read(), write()
3. Cambio de modo usuario a modo kernel

5.1.4 Espacio de usuario vs espacio de kernel

1. Separación de privilegios
2. Seguridad y protección del sistema

5.2 Conceptos de repaso

- **Proceso de arranque:** En GRUB vimos la secuencia BIOS/UEFI → bootloader → **kernel** → init/systemd; aquí estudiamos qué es ese kernel que se carga y qué hace.
- **Linux como kernel:** En "De Unix a GNU/Linux" vimos que Linux es el kernel y GNU las herramientas de usuario; el kernel es el intermediario entre ambas capas.

5.3 Máximas

5.4 Sección libre

---

6 Modelo de Ejecución y Gestión de Procesos en Linux

6.1 Conceptos nuevos

- Programa vs Proceso
  - Definición de programa
  - Definición de proceso
  - Ciclo de vida de un proceso
- Procesos y Tareas en Linux
  - Qué es una tarea (task)
  - Concepto de hilo (thread)
  - Diferencia entre proceso e hilo (thread)
  - Identificador de proceso (PID)
  - Estados de un proceso (running, ready, waiting, stopped, zombie)
- Representación Interna de Procesos en Linux
  - struct $\text{task}_{\text{struct}}$
    - Estructura que representa un proceso en el kernel
    - Contiene:
      - PID
      - Estado
      - Prioridad
      - Información básica de memoria
- Qué es el scheduler
  - Asignación de CPU a procesos
  - Cambio de contexto (context switch)
  - Procesos con mayor o menor prioridad
  - Tiempo de CPU compartido

  - sched.h (visión general)
    * Definición de políticas de planificación
    * Constantes relacionadas con scheduling
    * Interacción con $\text{task}_{\text{struct}}$

- Comandos útiles
  - pstree
  - nohup

6.2 Conceptos de repaso

- Espacio de Usuario vs Espacio de Kernel
  - Separación de privilegios
  - Protección de memoria
  - Cambio de modo (user mode kernel mode)
- System Calls (Llamadas al Sistema)
  - Mecanismo para que un proceso solicite servicios al kernel
  - Interfaz entre usuario y kernel
  - Cambio de contexto
  - Ejemplos comunes de syscalls: fork(), exec(), wait(), read(), write(), open(), exit()
- Flujo básico de una system call
  - Interrupción o instrucción especial (syscall/sysenter)
  - El proceso solicita un servicio
  - El CPU cambia a modo kernel
  - El kernel ejecuta la operación
  - Se regresa al modo usuario

6.3 Máximas

6.4 Sección libre

- Un proceso es un programa que se está ejecutando en el sistema y constantemente le está pidiendo recursos al kernel. Los procesos se comunican con el kernel a través de syscalls.
- Linux tiene un directorio llamado /tmp el cual es volátil. Ahí se guardan archivos necesarios para el funcionamiento de programas y SO durante la sesión, pero que no es necesario conservar y por lo tanto se pueden borrar en el siguiente apagado. En Windows existe la misma implementación en una carpeta llamada Temp.
- **nohup** nos permite ejecutar procesos y evitar que mueran cuando el padre muere (ignora señales de hangup). Para guardar la salida en un archivo: `nohup COMANDO > FILE`.

- **pstree** muestra un árbol de los procesos en ejecución, dando una vista clara de las relaciones de parentesco. De ahí se observa que el proceso del que nacen todos los procesos en espacio de usuario es systemd.

```
pstree
```

```
systemd-+-NetworkManager---3*[{NetworkManager}]
        |-ananicy-cpp---3*[{ananicy-cpp}]
        |-avahi-daemon---avahi-daemon
        |-bluetoothd
        |-chrome_crashpad---2*[{chrome_crashpad}]
        |-containerd---12*[{containerd}]
        |-containerd-shim-+-postgres---5*[postgres]
        |                 `-10*[{containerd-shim}]
        |-containerd-shim-+-s6-svscan-+-s6-supervise---gitea---9*[{gitea}]
        |                 |           `-s6-supervise---sshd
        |                 `-10*[{containerd-shim}]
        |-containerd-shim-+-bash-+-/opt/lampp/bin/---6*[/opt/lampp/bin/]
        |                 |      |-mysqld_safe---mysqld---29*[{mysqld}]
        |                 |      |-proftpd
        |                 |      |-sshd
        |                 |      `-tail
        |                 `-10*[{containerd-shim}]
        |-2*[containerd-shim-+-mysqld---36*[{mysqld}]]
        |                    `-10*[{containerd-shim}]]
        |-containerd-shim-+-apache2---5*[apache2]
        |                 `-10*[{containerd-shim}]
        |-dbus-broker-lau---dbus-broker
        |-dockerd-+-4*[docker-proxy---6*[{docker-proxy}]]
        |         |-6*[docker-proxy---7*[{docker-proxy}]]
        |         |-2*[docker-proxy---8*[{docker-proxy}]]
        |         `-36*[{dockerd}]
        |-electron-+-electron---electron---16*[{electron}]
        |          |-electron---electron---electron---11*[{electron}]
        |          |-electron---7*[{electron}]
        |          |-electron-+-electron---11*[{electron}]
        |          |          `-23*[{electron}]
        |          |-2*[electron---18*[{electron}]]
        |          |-electron---20*[{electron}]
        |          |-electron-+-zsh
        |          |          `-19*[{electron}]
        |          `-37*[{electron}]
        |-emacs-+-bash---pstree
        |       `-7*[{emacs}]
        |-ksecretd
        |-msedge-+-2*[cat]
        |        |-msedge---msedge---20*[{msedge}]
        |        |-msedge---msedge-+-msedge---7*[{msedge}]
```

```
|        |                 |-6*[msedge---10*[{msedge}]]
|        |                 |-msedge---12*[{msedge}]
|        |                 |-3*[msedge---9*[{msedge}]]
|        |                 |-msedge---23*[{msedge}]
|        |                 |-msedge---14*[{msedge}]
|        |                 `-msedge---8*[{msedge}]
|        |-msedge---8*[{msedge}]
|        |-msedge---7*[{msedge}]
|        `-51*[{msedge}]
|-msedge_crashpad---2*[{msedge_crashpad}]
|-msedge_crashpad---{msedge_crashpad}
|-netbird---8*[{netbird}]
|-polkitd---3*[{polkitd}]
|-power-profiles----3*[{power-profiles-}]
|-rtkit-daemon---2*[{rtkit-daemon}]
|-sddm-+-sddm-helper---start-hyprland-+-Hyprland-+-Xwayland---4*[{Xwayland}
|      |                              |          |-alacritty-+-zsh
|      |                              |          |           `-9*[{alacritt
|      |                              |          |-hyprpaper---13*[{hyprpap
|      |                              |          |-waybar---37*[{waybar}]
|      |                              |          `-14*[{Hyprland}]
|      |                              `-{start-hyprland}
|      `-{sddm}
|-spotify-+-spotify---spotify---15*[{spotify}]
|         |-spotify-+-spotify---7*[{spotify}]
|         |         `-spotify---13*[{spotify}]
|         |-spotify---8*[{spotify}]
|         `-63*[{spotify}]
|-systemd-+-(sd-pam)
|         |-at-spi-bus-laun-+-dbus-broker-lau---dbus-broker
|         |                 `-4*[{at-spi-bus-laun}]
|         |-at-spi2-registr---3*[{at-spi2-registr}]
|         |-dbus-broker-lau---dbus-broker
|         |-dconf-service---3*[{dconf-service}]
|         |-pipewire---2*[{pipewire}]
|         |-pipewire-pulse---2*[{pipewire-pulse}]
|         |-wireplumber---5*[{wireplumber}]
|         |-2*[xdg-desktop-por---4*[{xdg-desktop-por}]]
|         |-xdg-desktop-por---7*[{xdg-desktop-por}]
|         |-xdg-document-po-+-fusermount3
|         |                 `-6*[{xdg-document-po}]
|         `-xdg-permission----3*[{xdg-permission-}]
|-systemd-journal
|-systemd-logind
|-systemd-resolve
|-systemd-timesyn---{systemd-timesyn}
|-systemd-udevd
|-systemd-userdbd---3*[systemd-userwor]
```

```
|-upowerd---3*[{upowerd}]
`-wpa_supplicant
```

Para entender mejor el funcionamiento de un proceso, se puede revisar el código fuente del kernel Linux. El término usado en el kernel para referirse a los procesos es una **task** (tarea), ya que la misma estructura sirve para procesos y para hilos. La estructura es `task_struct`, declarada en `include/linux/sched.h`. Contiene, entre otras cosas: estado ($\_{state}$), PID y TGID, prioridades (prio, $static_{prio}$), relaciones (parent, children, sibling), memoria (mm), archivos abiertos (files), credenciales (cred) y manejo de señales (signal, sighand). El proceso más importante y del que nacen todos los procesos en espacio de usuario es systemd.

---

7 Arquitectura de almacenamiento físico y su representación en Linux

7.1 Conceptos nuevos

7.1.1 Dispositivos de almacenamiento

- HDD vs SSD
- NVMe vs SATA
- Rendimiento, latencia e IOPS
- Persistencia vs volatilidad

7.1.2 Representación en Linux

- Dispositivos en /dev (/dev/sda, /dev/sdb, /dev/nvme0n1)
- Concepto de block device
- Uso de lsblk y blkid

7.1.3 Interfaces de comunicación de discos

- SCSI
- SAS (Serial Attached SCSI)
- SATA (Serial ATA)
- NVMe (Non-Volatile Memory Express)

7.1.4 Particiones

- ¿Qué es una partición?
- División lógica del disco
- Independencia de particiones
- Concepto de sector y bloque

7.1.5 Tablas de partición

- MBR (Master Boot Record)
- GPT (GUID Partition Table)
- Diferencias técnicas (límite 2TB, número de particiones, redundancia)

7.1.6 Tipos de particiones (MBR)

- Primaria
- Extendida
- Lógica

7.1.7 Herramientas de particionado

- fdisk, cfdisk, parted, gparted

7.2 Conceptos de repaso

- **GRUB y discos:** En la consola de GRUB vimos `(hd0,gpt1)`, discos y particiones; la EFI System Partition; el bootloader ya "ve" discos y tablas de partición (GPT).
- **Tablas de partición:** GPT apareció en el arranque (partición EFI); aquí se ven MBR y GPT con detalle (límites, número de particiones).
- **Todo es un archivo:** Los dispositivos se representan como archivos en /dev (máxima en GRUB); aquí vemos /dev/sda, /dev/nvme0n1, etc.
- **RAID:** En disponibilidad vimos varios discos independientes; aquí cómo se representan esos discos en Linux (block devices, lsblk).

7.3 Máximas

7.4 Sección libre

- Gparted es una herramienta con interfaz gráfica para crear y modificar particiones en un disco.
- En Linux, 2.3.1. La carpeta /dev guarda representaciones en archivo de todos los dispositivos conectados (almacenamiento, terminales, pseudoterminales).

---

8 Estructura lógica y administración de sistemas de archivos

8.1 Conceptos nuevos

8.1.1 ¿Qué es un sistema de archivos?

- Organización lógica de datos

- Relación con particiones
- Abstracción sobre el hardware

8.1.2 Sistemas de archivos comunes en Linux

- ext4, xfs, btrfs, vfat, ntfs

8.1.3 Estructura interna básica

- Superbloque
- Inodos
- Bloques de datos
- Journaling
- Metadatos

8.1.4 Creación y formateo de particiones

- mkfs, mkfs.ext4, mkfs.xfs
- Verificación con fsck

8.1.5 Montaje

- mount, umount
- Puntos de montaje (/mnt, /media)
- Montaje temporal vs permanente

8.1.6 Archivo /etc/fstab

- Montaje automático al inicio
- UUID vs nombre de dispositivo
- Opciones comunes (defaults, noatime, ro, rw)

8.2 Conceptos de repaso

- **GRUB y sistemas de archivos:** GRUB puede leer sistemas de archivos (FAT, EXT, ISO) sin montar; el bootloader accede a estructuras del FS. Montar es una abstracción del **kernel**, no del bootloader.
- **Particiones y almacenamiento:** Un sistema de archivos va sobre particiones; ya vimos /dev, block devices, particionado y tablas (MBR/GPT) en el capítulo anterior.
- **Kernel:** El kernel es quien monta y gestiona los FS; las syscalls (read, write, open) operan sobre archivos una vez montado el FS.

8.3 Máximas

8.4 Sección libre

---

9 Permisos y Propiedad

9.1 Conceptos nuevos

9.1.1 Permisos básicos

- Lectura (r), Escritura (w), Ejecución (x)
- Comando para ver permisos: `ls -la`

9.1.2 Usuarios y grupos

- Propietario, Grupo, Otros
- Comandos: chmod, chown, chgrp

9.2 Conceptos de repaso

- **`ls -la`:** Ya usado en GRUB (Sección libre) para listar en forma de lista y mostrar archivos ocultos; la misma salida muestra permisos (r/w/x) y propietario/grupo.
- **Rutas y archivos:** En GRUB vimos rutas absolutas y relativas para referenciar archivos; aquí quién (qué usuario/grupo) puede acceder a cada archivo.
- **Sistema de archivos:** Los archivos tienen metadatos (inodos, etc.); propiedad y permisos son parte de esa información que el FS guarda.
- **Procesos y kernel:** En `task_struct` vimos `cred` (credenciales); el kernel usa UID/GID del proceso para comprobar permisos en cada acceso.

9.3 Máximas

9.4 Sección libre