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

2 Sesión 1: 3 de Febrero 2026

Esta sesión fue una introducción general a la clase y a conceptos iniciales de Linux.

2.1 Conceptos nuevos

- 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 556
drwxr-xr-x 1 mrtaichi mrtaichi    132 Feb  8 20:57 .
drwxr-xr-x 1 mrtaichi mrtaichi     84 Feb  7 14:39 ..
-rw-r--r-- 1 mrtaichi mrtaichi  16965 Feb  8 20:59 bitacora1.org
-rw-r--r-- 1 mrtaichi mrtaichi 385556 Feb  8 20:57 bitacora1.pdf
-rw-r--r-- 1 mrtaichi mrtaichi  19353 Feb  8 20:57 bitacora1.tex
drwxr-xr-x 1 mrtaichi mrtaichi    128 Feb  8 20:59 .git
-rw-r--r-- 1 mrtaichi mrtaichi     25 Feb  8 20:57 test.txt
-rw-r--r-- 1 mrtaichi mrtaichi 133687 Feb  7 12:15 ventoy_boot.jpg
```

- 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.

- 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
```

- LAN? (en proceso)
- En Linux, 2.3.1. Esto incluye a los dispositivos, ya que la carpeta /dev guarda representaciones en archivo de todos los dispositivos que están conectados a la computadora. Esto incluye desde los mismos dispositivos de almacenamiento, hasta cosas más "virtuales" como las terminales y pseudoterminales del sistema.
- Dispositivo psar? (en proceso)
- Gparted es una herramienta con interfaz gráfica para crear y modificar particiones en un disco.

2.2 Conceptos de repaso

N/A (primera clase)

2.3 Máximas

2.3.1 Todo es un archivo

En los sistemas derivados de Unix, como Linux

2.4 Sección libre

3 Sesión 2: 4 de Febrero 2026

3.1 Conceptos nuevos

- 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.
- La necesidad principal de tener redundancia en la información es para no tener un sólo punto de falla en nuestros sistemas que pueda fallar. Si alguna parte de nuestro sistema falla, siempre existe otro componente listo para tomar el lugar y mantener el sistema funcionando.
- Generalmente la emulación es más lenta que la virtualización. El primer sistema de WSL usaba emulación de Linux, ahora el segundo utiliza virtualización.
- El propósito de que cada quien utilice una distribución distinta es demostrar la máxima: "3.3.1". Vamos a observar como -en esencia- todos las distribuciones de Linux tienen tantas cosas en común que podemos hacer cualquier cosa en cualquier distro.
- Distrowatch es un sitio web en el que se pueden observar todas las distribuciones de Linux y muchos datos acerca de ellas. Lo que resulta interesante es que incluso un ranking de cuáles son las distros más usadas hasta el mometo.

- 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.

3.2 Conceptos de repaso

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

4 Sesión 3: 5 de Febrero 2026

4.1 Conceptos nuevos

- 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
- Y de ahí también nace Linux, como un kernel Unix-like desarrollado en 1991 por Linus Torvalds, el cual se termina fusionando con GNU para formar un sistema operativo completo.
- 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

4.2 Conceptos de repaso

4.3 Máximas

4.4 Sección libre

Para entender mejor el funcionamiento de un proceso, se me ocurrió meterme al código fuente de Linux y, con ayuda de la IA, aprender acerca de la estructura que conforma un proceso como tal.

Primero había que ubicarla, y después de indagar un rato en la documentación me pude dar cuenta de que el término usado en el kernel para referirse a los procesos es una task (o tarea). Esto debido a que esta estructura sirve tanto para procesos de programas como para los hilos de un mismo programa. La estructura que buscamos es entonces $\text{task}_{\text{struc}}$, la cual se encuentra declarada en include/linux/sched.h

Como el código que define esta estructura es muy extenso, es aquí donde me tuve que servir de la IA para entenderle. Le pasé el archivo y le pedí una explicación sobre cada uno de los componentes de esta estructura. Este fue el resultado:

> Entiendo perfectamente, la ‘struct $\text{task}_{\text{struct}}$‘ es intimidante porque es el “corazón” de Linux y contiene prácticamente todo lo que el kernel sabe sobre un proceso. Basado en

el archivo 'sched.h' que compartiste, vamos a despedazarla en partes lógicas para que sea más fácil de digerir.

Piensa en la '$task_{struct}$' como el **expediente completo** de una persona en una oficina gubernamental; tiene su nombre, su familia, su dinero, sus propiedades y qué está haciendo en este momento.

1. Estado y Propósito (¿Qué está haciendo?)

Lo primero que el kernel necesita saber es si el proceso puede correr o si está esperando algo.

***'$\_{state}$'**: Define si el proceso está '$TASK_{RUNNING}$' (listo para usar la CPU), '$TASK_{INTERRUPTIBLE}$' (durmiendo pero puede despertar con una señal) o en otros estados como '$TASK_{DEAD}$'. ***'stack'**: Un puntero al stack del kernel para este proceso específico. ***'flags'**: Banderas que indican atributos especiales (por ejemplo, si es un hilo del kernel o si está saliendo).

2. Identificación (¿Quién es?)

***'pid'**: El **Process ID** único que ves cuando usas comandos como 'top'. ***'tgid'**: El **Thread Group ID**. En Linux, para el hilo principal, el PID y el TGID son iguales. Si creas más hilos en ese proceso, todos compartirán el mismo TGID pero tendrán PIDs distintos. ***'comm[$TASK_{COMM LEN}$]'**: El nombre del ejecutable (máximo 16 caracteres).

3. El Planificador (¿Cuándo le toca la CPU?)

Esta sección decide la prioridad del proceso.

***'prio', '$static_{prio}$', '$normal_{prio}$**': Las distintas prioridades que el planificador usa para decidir a quién darle la CPU. ***'se', 'rt', 'dl**': Estas son sub-estructuras ('$sched_{entity}$', '$sched_{rtentity}$', etc.) que contienen los datos específicos para diferentes tipos de procesos (normales, de tiempo real, etc.).

4. Relaciones Familiares (¿Quién lo creó?)

Linux organiza los procesos en un árbol genealógico.

***'parent'**: Puntero al proceso padre. ***'children'**: Una lista de todos los procesos hijos que este proceso ha creado. ***'sibling'**: Enlaces a sus "hermanos" (otros procesos creados por el mismo padre).

5. Recursos (¿Qué posee?)

Aquí es donde la estructura se conecta con el resto del sistema.

***'mm'**: Puntero a 'struct mm_{struct}'. Aquí reside toda la información de la **memoria virtual** del proceso (sus páginas, segmentos, etc.). ***'files'**: Puntero a 'struct $files_{struct}$'. Es la tabla que guarda todos los **archivos abiertos** (tus descriptores de archivo 0, 1, 2, etc.). ***'fs'**: Información del sistema de archivos, como el directorio actual ('pwd'). ***'cred'**: Las credenciales de seguridad (quién es el dueño: UID, GID).

6. Señales (¿Qué le han dicho?)

***'signal'** y **'sighand'**: Manejan qué debe hacer el proceso cuando recibe señales como un 'SIGKILL' o 'SIGTERM'. ***'pending'**: Una lista de señales que han llegado pero que el proceso aún no ha procesado.

Resumen rápido para no volverse loco: Si estás programando en el kernel y tienes un puntero a un proceso (usualmente llamado 'p' o 'current'), lo más común es que solo te

interesen ‘p->pid‘, ‘p->comm‘, ‘p->mm‘ (memoria) o ‘p->files‘ (archivos). El resto son detalles técnicos para que el kernel gestione la magia por debajo.

¿Hay alguna de estas secciones que te interese explorar más a fondo con el código que pasaste?

De esto ya pude entender mejor cómo funciona el tema de los procesos en Linux y la relación de parentesco que debe existir entre todos ellos que los lleva al proceso inicial (systemd).

5 Sesión 4: 6 de Febrero 2026

5.1 Conceptos nuevos

- Linux tiene un directorio llamado /tmp el cual es volátil. Ahí se guardan todos los archivos que se necesitan para el funcionamiento de los programas y SO durante la sesión, pero que no son necesarios de conservarse y por lo tanto se pueden borrar en el siguiente apagado. En Windows existe la misma implementación en una carpeta llamada Temp
- nohup es un comando que nos permite ejecutar procesos y evitar que mueran cuando el padre muera.

```
man nohup
```

```
NOHUP(1)                          User Commands                          NOHUP(

NAME
       nohup - run a command immune to hangups, with output to a non-tty

SYNOPSIS
       nohup COMMAND [ARG]...
       nohup OPTION

DESCRIPTION
       Run COMMAND, ignoring hangup signals.

       --help display this help and exit

       --version
              output version information and exit

       If  standard  input  is a terminal, redirect it from an unreadable fil
       If standard output is a terminal, append output to 'nohup.out' if poss
       ble, '$HOME/nohup.out' otherwise.  If  standard  error  is  a  termina
       redirect it to standard output.  To save output to FILE, use 'nohup CO
       MAND > FILE'.

       Your  shell  may have its own version of nohup, which usually supersed
       the version described here.  Please refer to your shell's  documentati
```

```
       for details about the options it supports.

   Exit status:
       125    if the nohup command itself fails

       126    if COMMAND is found but cannot be invoked

       127    if COMMAND cannot be found

       -      the exit status of COMMAND otherwise

AUTHOR
       Written by Jim Meyering.

REPORTING BUGS
       Report bugs to: bug-coreutils@gnu.org
       GNU coreutils home page: <https://www.gnu.org/software/coreutils/>
       General help using GNU software: <https://www.gnu.org/gethelp/>
       Report any translation bugs to <https://translationproject.org/team/>

COPYRIGHT
       Copyright © 2025 Free Software Foundation, Inc.  License GPLv3+: GNU G
       version 3 or later <https://gnu.org/licenses/gpl.html>.
       This  is  free  software:  you  are  free to change and redistribute i
       There is NO WARRANTY, to the extent permitted by law.

SEE ALSO
       Full documentation <https://www.gnu.org/software/coreutils/nohup>
       or available locally via: info '(coreutils) nohup invocation'

GNU coreutils 9.9               November 2025                          NOHUP(
```

- Con el comando pstree podemos ver un árbol de los procesos en ejecución actualmente en nuestro sistema. Esto nos da una vista muy clara de las relaciones de parentesco de todos estos.

```
pstree


systemd-+-NetworkManager---3*[{NetworkManager}]
        |-ananicy-cpp---3*[{ananicy-cpp}]
        |-avahi-daemon---avahi-daemon
        |-bluetoothd
        |-containerd---11*[{containerd}]
        |-containerd-shim-+-s6-svscan-+-s6-supervise---gitea---8*[{gitea}]
        |                 |           `-s6-supervise---sshd
        |                 `-11*[{containerd-shim}]
        |-2*[containerd-shim-+-mysqld---36*[{mysqld}]]
```

```
|                    `-10*[{containerd-shim}]]
|-containerd-shim-+-apache2---5*[apache2]
|                 `-10*[{containerd-shim}]
|-containerd-shim-+-postgres---5*[postgres]
|                 `-10*[{containerd-shim}]
|-containerd-shim-+-bash-+-/opt/lampp/bin/---7*[/opt/lampp/bin/]
|                 |      |-mysqld_safe---mysqld---29*[{mysqld}]
|                 |      |-proftpd
|                 |      |-sshd
|                 |      `-tail
|                 `-10*[{containerd-shim}]
|-dbus-broker-lau---dbus-broker
|-dockerd-+-3*[docker-proxy---6*[{docker-proxy}]]
|         |-2*[docker-proxy---8*[{docker-proxy}]]
|         |-6*[docker-proxy---7*[{docker-proxy}]]
|         |-docker-proxy---5*[{docker-proxy}]
|         `-45*[{dockerd}]
|-emacs---7*[{emacs}]
|-emacs-+-bash---pstree
|       |-bwrap---bwrap---glycin-svg---2*[{glycin-svg}]
|       |-zsh---git---vi
|       `-8*[{emacs}]
|-ksecretd
|-msedge-+-2*[cat]
|        |-msedge---msedge---15*[{msedge}]
|        |-msedge---msedge-+-2*[msedge---7*[{msedge}]]
|        |                 |-2*[msedge---10*[{msedge}]]
|        |                 |-msedge---11*[{msedge}]
|        |                 `-7*[msedge---9*[{msedge}]]
|        |-msedge---8*[{msedge}]
|        `-46*[{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}
|      |                              |          |-hyprpaper---13*[{hyprpap
|      |                              |          |-waybar---35*[{waybar}]
|      |                              |          `-14*[{Hyprland}]
|      |                              `-{start-hyprland}
|      `-{sddm}
|-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
|          |                 `-7*[{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
```

- De aquí podemos observar una vez más como el proceso más importante y del que nacen todos los procesos en espacio de usuario es el proceso systemd. Además, observamos algunas cosillas interesantes como los procesos que generan nuestro entorno de escritorio, pipewire para el sonido, etc.

5.2 Conceptos de repaso

- Me parece que nohup ya había sido visto?

5.3 Máximas

5.4 Sección libre