1. ¿Qué es el kernel de Linux y qué es una distro? Mencione la versión de ambos.

- **Kernel de Linux**: Es el núcleo del sistema operativo. Gestiona recursos como memoria, CPU, y dispositivos de hardware. Es el intermediario entre el hardware y el software.
- **Distro (distribución)**: Es una versión completa de Linux que incluye el kernel, herramientas de sistema, gestores de paquetes, y aplicaciones. Ejemplo: Ubuntu.

Comandos:

```
uname -r           # Versión del kernel
lsb_release -a     # Info de la distro
```

Ejemplo de salida:

```
5.15.0-91-generic          # Kernel
Ubuntu 22.04.4 LTS         # Distro
```

---

2. ¿Qué tipo de arquitectura tiene su sistema operativo?

- Se refiere a si tu sistema es de 32 o 64 bits.

Comando:

```
uname -m
```

- `x86_64` → 64 bits
- `i686` o `i386` → 32 bits

---

3. ¿Qué es la shell? ¿Qué shell utiliza su distro?

- **Shell**: Es el programa que interpreta los comandos que escribís. Permite interactuar con el sistema operativo.
- En Ubuntu, por defecto se usa **bash**, pero vos mencionás que **no usás bash**, así que posiblemente estés usando **sh**, **dash**, o **zsh**.

Comando para saber la shell actual:

```
echo $SHELL
```

---

4. ¿Qué es la GUI?

- **GUI (Graphical User Interface)**: Es la interfaz gráfica del sistema operativo, con ventanas, íconos, menús, etc. En Ubuntu, el entorno gráfico suele ser **GNOME**, pero como dijiste, no lo estás usando.

---

5. ¿Por qué utilizaría el CLI si tiene la GUI?

- El **CLI (Command Line Interface)** es más rápido, consume menos recursos, permite automatizar tareas con scripts, y es más potente para usuarios avanzados. Aunque haya GUI, muchas tareas de administración se hacen mejor por CLI.

---

6. ¿Qué es el usuario root?

- Es el **usuario administrador** del sistema. Tiene todos los permisos y puede hacer cualquier cambio, incluso cosas peligrosas como borrar el sistema.

---

7. ¿Cómo crear un nuevo usuario?

Comando:

```
sudo adduser nombre_usuario
```

Ejemplo:

```
sudo adduser juan
```

---

8. ¿Cómo cambiar la clave de un usuario?

Comando:

```
sudo passwd nombre_usuario
```

Ejemplo:

```
sudo passwd juan
```

---

9. ¿Qué es el gestor de paquetes? ¿Qué gestor tiene su distro?

- Es una herramienta que permite instalar, actualizar o eliminar software.
- En Ubuntu se usa **APT (Advanced Package Tool)**.

Ejemplo de uso:

```
sudo apt update
sudo apt install nombre_paquete
```

---

10. ¿Para qué se usa el comando sudo?

- Permite ejecutar comandos como **superusuario (root)**. Es necesario para tareas administrativas.

Ejemplo:

```
sudo apt update
```

---

11. ¿Cómo puede moverse entre directorios?

Comando:

```
cd ruta_del_directorio
```

Ejemplos:

```
cd /etc
cd ..
cd ~       # Ir al home
```

---

12. ¿Cómo crear un directorio?

Comando:

```
mkdir nombre_directorio
```

Ejemplo:

```
mkdir proyectos
```

---

13. ¿Cómo listar los archivos y directorios contenidos en un directorio?

Comando:

```
ls
ls -l        # Con detalles
ls -a        # Incluye archivos ocultos
```

---

14. ¿Cómo crear un archivo de texto?

Comando:

```
touch archivo.txt
```

O también con redirección:

```
echo "Hola" > archivo.txt
```

---

15. ¿Cómo puedo solamente visualizar por pantalla un archivo de texto?

Comando:

```
cat archivo.txt
```

O:

```
less archivo.txt
more archivo.txt
```

---

16. ¿Qué alternativas tiene para poder crear y editar un archivo de texto con su sistema operativo?

- Editores desde la terminal:
  - `nano`
  - `vi` o `vim`
  - `sed` (para edición por comandos)
  - `echo` con redirección

Ejemplo con nano:

```
nano archivo.txt
```

---

17. ¿Cómo se puede copiar un archivo de un directorio a otro?

Comando:

```
cp origen destino
```

Ejemplo:

```
bash
Copiar código
cp archivo.txt /home/juan/
```

---

18. ¿Cómo se puede mover un archivo de un directorio a otro?

Comando:

```
mv origen destino
```

Ejemplo:

```
mv archivo.txt /home/juan/
```

---

1. ¿Cómo se puede visualizar la información vinculada a los procesos?

Podés ver información de los procesos en tiempo real o de forma puntual con varios comandos:

Comandos comunes:

```
ps aux           # Lista todos los procesos
top              # Muestra procesos en tiempo real
htop             # Igual que top pero más visual (puede requerir
instalación)
```

Para ver procesos específicos:

```
ps -u usuario     # Procesos de un usuario
ps -p PID         # Info de un proceso por ID
```

---

2. ¿Qué datos vinculados a esta unidad puede extraer?

Desde `ps`, `top`, o `/proc`, podés ver:

- **PID**: Identificador del proceso
- **PPID**: PID del proceso padre
- **USER**: Usuario que inició el proceso
- **%CPU y %MEM**: Uso de CPU y memoria
- **STAT**: Estado del proceso (`R`, `S`, `Z`, `T`, etc.)
- **TIME**: Tiempo total de CPU que usó
- **COMMAND**: Comando ejecutado

También se puede inspeccionar `/proc/[PID]/` para ver más detalles técnicos del proceso.

---

3. ¿Cómo se puede cambiar la prioridad de un proceso?

La prioridad se maneja con el valor de **"nice"** (cuanto más bajo, más prioridad) y **"renice"** para modificarla.

Iniciar un proceso con prioridad específica:

```
nice -n 10 comando
```

Cambiar la prioridad de un proceso ya en ejecución:

```
sudo renice -n 5 -p PID
```

---

4. ¿Cómo se puede forzar la finalización de un proceso? ¿Qué alternativas existen?

Usar `kill`:

```
kill PID                  # Envío de señal SIGTERM (terminar)
kill -9 PID               # Envío de señal SIGKILL (forzado)
```

Otras alternativas:

- `killall nombre_proceso` → Mata todos los procesos con ese nombre
- Desde `top` o `htop` podés también terminar procesos (con `k` en `top`)
- `pkill nombre_proceso`

---

5. ¿Qué sistemas de planificación de procesos tiene su SO? Describa su funcionamiento.

Ubuntu (Linux en general) usa varios **planificadores de CPU**. El más común es:

CFS (Completely Fair Scheduler) – *Planificador por defecto en Linux*:

- Asigna tiempo de CPU a los procesos de manera equitativa (justa).
- Usa árboles red-black para organizar los procesos según su "tiempo virtual".
- Puede manejar prioridades con "nice" y "cgroups".

Otros disponibles (menos comunes en desktops):

- **Deadline** (para RT)
- **FIFO**
- **RR (Round Robin)**

---

6. ¿Cómo se puede modificar el planificador de CPU de su instalación?

Se puede cambiar a nivel de proceso con `chrt`, o usar configuraciones en tiempo de arranque del kernel.

Para ver y cambiar la política de planificación:

```
chrt -p PID                     # Ver política de un proceso
sudo chrt -r -p 50 PID          # Cambiar a round-robin con prioridad
50
```

Políticas comunes:

- `SCHED_OTHER` (CFS, por defecto)
- `SCHED_FIFO`
- `SCHED_RR`
- `SCHED_DEADLINE`

Se requiere superusuario para usar `chrt` con políticas de tiempo real.

. ¿Cómo maneja la Sección Crítica el sistema operativo utilizado? Describa los mecanismos que utiliza.

Una **sección crítica** es un bloque de código donde se accede a recursos compartidos (por ejemplo, memoria, archivos, dispositivos) que no deben ser modificados simultáneamente por múltiples procesos o hilos.

🔒 **Mecanismos que utiliza Ubuntu/Linux para proteger secciones críticas:**

1. **Semáforos (Semaphores)**
   - Permiten controlar el acceso de múltiples procesos a recursos compartidos.
   - Son contadores que se incrementan o decrementan con operaciones atómicas (`sem_wait`, `sem_post`).
   - Pueden ser **binarios** (0/1, como un mutex) o **contadores**.
2. **Mutexes (Mutual Exclusion)**
   - Se usan en programación multihilo con `pthreads` para asegurar que un solo hilo pueda ejecutar una sección crítica.
   - Operaciones: `pthread_mutex_lock`, `pthread_mutex_unlock`.
3. **Spinlocks**
   - Similares a los mutexes, pero en vez de bloquear el hilo, hace "busy waiting" (espera activa).
   - Útiles en el **kernel**, cuando el recurso se desbloquea muy rápido.
4. **Monitores**
   - **Monitores** son una abstracción de más alto nivel. Es una estructura que encapsula variables compartidas, procedimientos y mecanismos de sincronización.
   - En Linux, no existen como entidad explícita en el kernel, pero se implementan mediante **mutexes + variables de condición**.
   - Ejemplo en POSIX:

     ```
     pthread_mutex_t lock;
     pthread_cond_t cond;
     ```

   - Combinan exclusión mutua con espera y señalización para control fino entre hilos.
5. **Variables de condición (Condition Variables)**
   - Permiten que un hilo se duerma mientras espera una condición, y otro lo despierte.
   - Siempre se usan junto a mutexes.
6. **Atomic Operations**

- Operaciones a nivel de CPU que garantizan consistencia sin necesidad de locks (por ejemplo, `__sync_fetch_and_add()` en C).

7. Secciones Críticas en el Kernel

- Usa mecanismos como `preempt_disable()` y locks como `spinlock_t`, `mutex`, `rwlock`, etc., para proteger secciones críticas a nivel del sistema operativo.