Unidad 3 - Gestión (de Sistemas Informáticos)

1. Introducción

Management: Procesos y herramientas que permiten controlar, supervisar y optimizar el funcionamiento interno de un SO

El sistema operativo gestiona los recursos físicos y lógicos de un ordenador: la memoria, el procesador, los dispositivos de entrada y salida, los archivos, y los procesos que ejecutan los programas.

3. Introduction: MultiTask vs MonoTask / MultiUser vs MonoUser

Proceso: Programa en ejecución

Monotarea: Solo un proceso puede ejecutarse a la vez (MS-DOS=

Multitarea: Permite ejecutar varios procesos de forma “simultánea” gracias al uso del procesador y a técnicas de planificación que alternan la ejecución de los procesos muy rápidamente

Aunque en realidad el CPU solo puede ejecutar una instrucción por núcleo en un instante, la velocidad es tan alta que el usuario percibe que varios programas corren a la vez.

Diferencia: Radica en cuántos usuarios pueden acceder al sistema al mismo tiempo.

Process Control Block (PCB): Almacena toda la información que el sistema necesita sobre cada proceso: su ID, estado, contador de programa, registros, uso de CPU, información de memoria, etc.

La multitarea también puede ser **cooperativa** (los procesos ceden voluntariamente el control) o **preemptiva** (el sistema interrumpe los procesos según prioridades). Los sistemas modernos usan la segunda, que es mucho más eficiente y justa.

4. ¿Cuántos procesos en paralelo?

Paralelismo, concepto fundamental para entender cómo los sistemas modernos logran ejecutar múltiples procesos simultáneamente

El grado de paralelismo depende directamente de los núcleos del procesador (cores). Cada procesador físico, o socket, puede contener uno o varios núcleos. Dentro de cada núcleo, a su vez, pueden existir varios hilos o threads, que son las unidades más pequeñas de ejecución.

- La CPU es el procesador completo, el chip.
- Cada core es una unidad de procesamiento independiente dentro de la CPU. Y cada hilo (thread) dentro del core representa una ejecución concurrente de instrucciones.

El SO usa planificadores (schedulers) que asignan procesos o hilos a los diferentes núcleos

GPU (Graphics Processing Units), son procesadores especializados con cientos o miles de núcleos más simples, diseñados para tareas altamente paralelas como el procesamiento gráfico o el cálculo científico

5. Monitorización de procesos (Daemons o Servicios)

Un proceso es una instancia de un programa en ejecución, pero no todos los procesos se comportan igual, otros se ejecutan en segundo plano sin intervención del usuario, estos se conocen como daemons (linux) o services (windows)

Los daemons suelen iniciarse automáticamente al arrancar el sistema.

Cada proceso atraviesa varios estados durante su vida útil:

- **User mode**: cuando se ejecuta código de usuario.
- **Kernel mode**: cuando el proceso requiere acceso a recursos del sistema y el control pasa al kernel.
- **Waiting:** el proceso espera un evento externo, por ejemplo, la lectura de un archivo.
- **Sleeping**: el proceso está inactivo temporalmente, esperando recursos.
- **Runnable**: listo para ejecutarse, esperando turno del procesador.
- **Virtual memory**: el proceso ocupa espacio en memoria virtual, lo que permite ejecutar más procesos de los que caben físicamente en la RAM.
- **New process**: estado inicial, cuando el proceso está siendo creado.
- **Zombie**: proceso que ya terminó pero cuyo registro aún no ha sido eliminado del sistema (espera que su proceso padre recoja su estado de salida).

Herramientas de monitoreo (monitorización)

Linux:

- **ps aux** para listar procesos con su consumo de recursos
- **top o htop** para verlos en tiempo real

- **vmstat** para estadísticas de memoria y CPU

En Windows, el Administrador de Tareas o el Monitor de Recursos cumplen funciones similares

Monitorizar procesos permite detectar cuellos de botella, identificar servicios bloqueados o zombies, y verificar el uso de CPU y memoria.

6. Scheduler

Su función es decidir qué proceso se ejecuta, cuándo y durante cuánto tiempo.

Cuando el Scheduler asigna la CPU a un proceso, se produce un cambio de contexto. Esto significa guardar el estado del proceso actual y cargar el del nuevo proceso que tomará el control del CPU.

Políticas de planificación

- **First Come First Served (FCFS):** Es la más simple: los procesos se ejecutan en el orden en que llegan. Es una política FIFO (First In, First Out).
- **Shortest Remaining Time First (SRTF) o Shortest Job First (SJF):** Los procesos con menor tiempo estimado de ejecución tienen prioridad.
- **Round-Robin (RR):** Se usa en sistemas de tiempo compartido. Cada proceso recibe un pequeño intervalo de tiempo (quantum). Cuando se agota, el Scheduler pasa al siguiente.
- **Real-Time Scheduling:** Se aplica en sistemas donde el tiempo de respuesta es crítico, como en dispositivos médicos o industriales. Usa políticas event-driven, donde las interrupciones se gestionan con máxima prioridad.

Políticas con y sin exclusividad

Sin exclusividad: los procesos pueden ser interrumpidos y reanudados más tarde

Con exclusividad: un proceso retiene la CPU hasta que termina o libera voluntariamente el control

7. Processes Management

Comandos principales:

- **& (Ampersand)**: al colocar & al final de un comando, el proceso se ejecuta en background, es decir, en segundo plano. Ejemplo: sleep 100 & ejecuta el proceso sin bloquear la terminal
- **Foreground y background**: puedes mover un proceso entre ambos estados con fg (foreground) y bg (background)

- **nohup**: ejecuta un proceso que no se detiene aunque cierres la sesión. Ideal para tareas largas en servidores. Ejemplo: nohup ./backup.sh &
- **jobs:** muestra los procesos en ejecución asociados a la sesión actual
- **kill**: termina un proceso usando su PID (Process ID). Se pueden usar señales específicas:
  - SIGTERM (15): solicita al proceso que termine
  - SIGKILL (9): lo fuerza a detenerse inmediatamente
  - SIGSTOP y SIGCONT: detienen o reanudan un proceso.

SubShell y Redirección: También se explican los SubShells, que son entornos de ejecución separados donde pueden lanzarse comandos sin afectar al shell principal. Ejemplo: (pwd; ls) > result.txt

8. System Calls Library

Llamadas al sistema (System Calls), son la interfaz entre el software de usuario y el kernel del sistema operativo.

Cuando un programa necesita acceder a un recurso del sistema no puede hacerlo directamente; debe hacerlo a través del kernel, usando una system call.

Las llamadas al sistema permiten que los programas de usuario utilicen servicios del sistema operativo de forma controlada. Ejemplos comunes son:

open() para abrir un archivo

read() y **write**() para operaciones de entrada/salida

fork() para crear nuevos procesos

kill() para enviar señales a otros procesos

time() para obtener la hora del sistema.

Estas funciones son implementadas en C por la librería **GLIBC** (GNU C Library)

La llamada mmap(), por ejemplo, se usa para mapear archivos o dispositivos en la memoria, permitiendo acceso directo y eficiente.

9. Nice, renice y el comando time (Gestión de Prioridades y Rendimiento)

Prioridades con nice

El comando nice permite iniciar un proceso con una prioridad diferente a la predeterminada. El valor de “niceness” va de -20 a +19:

- Un número más bajo (por ejemplo, -10) significa mayor prioridad
- Un número más alto (por ejemplo, +10) significa menor prioridad.

Ejemplo: nice -n 10 ./backup.sh

Cambiar prioridad con renice

Si un proceso ya está en ejecución, se puede modificar su prioridad con renice:

Ejemplo: renice -n -5 -p 1234

Aquí el proceso con PID 1234 obtiene mayor prioridad (-5). Los usuarios normales solo pueden aumentar el valor (bajar prioridad), mientras que los administradores (root) pueden disminuirlo.

Medir rendimiento con time

El comando time sirve para analizar el rendimiento de un proceso. Al ejecutarlo, muestra tres métricas clave:

- **real**: tiempo total de ejecución (desde inicio hasta fin)
- **user**: tiempo que el CPU dedicó al código del usuario.
- **sys**: tiempo que el CPU dedicó a llamadas al sistema operativo.

Ejemplo: time ls -l /home

10. Linux Kernel is written in C

El kernel es la parte **más crítica** del sistema operativo. Es el componente que interactúa **directamente** con el **hardware**. Todo lo que hace un programa, pasa antes por el kernel.

El **kernel** de **Linux** está escrito en el **lenguaje C**, con algunas **secciones específicas** en **ensamblador** (**assembly**) para tareas muy dependientes del hardware.

El código fuente está disponible públicamente bajo **licencia GPL** (General Public License)

Compilación y estructura del código

El código fuente se organiza en archivos .c y .h, y se compila mediante Makefiles, que automatizan el proceso. También puede usarse un archivo configure para generar el Makefile de acuerdo al sistema y hardware disponibles.

Los pasos básicos para compilar software son:

- Tener instaladas las herramientas de compilación (gcc, make, ld)
- Ejecutar make para generar los binarios
- Instalar los módulos o componentes resultantes

Incluso es posible analizar el código máquina generado con herramientas como objdump -d a.out

El uso del lenguaje C le da a Linux:

- **Velocidad** de ejecución, al ser código compilado
- **Control** total sobre la memoria
- **Portabilidad**, gracias a que el C es multiplataforma
- **Modularidad**, porque el kernel puede expandirse con módulos que se cargan dinámicamente (.ko)

11. Systemd vs SysVinit y Upstart

Systemd significa System Daemon, y es el sistema de inicio moderno que ha reemplazado a los clásicos SysVinit y Upstart.

Su función es inicializar los servicios y procesos necesarios durante el arranque. Mientras SysVinit utilizaba scripts secuenciales (/etc/init.d/), Systemd trabaja con unidades o targets, que pueden ejecutarse en paralelo, reduciendo drásticamente el tiempo de arranque.

Por ejemplo:

- multi-user.target → modo texto con red habilitada
- graphical.target → arranque completo con entorno gráfico

Cada unit (servicio, socket, dispositivo) se define mediante archivos .service, .target o .mount, lo que facilita su control con comandos como:

- systemctl start apache2
- systemctl status sshd

Ventajas de Systemd:

- **Arranque más rápido**: gracias a la ejecución paralela de servicios
- **Gestión unificada**: systemctl controla todo: servicios, logs y dependencias
- **Compatibilidad**: mantiene compatibilidad con SysVinit para sistemas antiguos
- **Monitoreo centralizado**: a través de journalctl, que unifica los registros del sistema

Systemd también incorpora el concepto de **targets**, que actúan como puntos de sincronización entre distintos grupos de servicios.

Relación con el kernel y PID 1

Systemd es el primer proceso que el kernel lanza después del arranque. A partir de él se inician todos los demás, incluyendo los daemons del sistema. Por eso, si Systemd falla, el sistema entero no puede continuar.

12. Acceso al kernel y medición del rendimiento (procfs y syscalls)

/proc – el sistema de archivos virtual

En Linux, el directorio /proc es un sistema de archivos virtual que no ocupa espacio físico. Contiene información en tiempo real sobre el estado del sistema y los procesos

Por ejemplo:

- /proc/cpuinfo muestra información del procesador
- /proc/meminfo detalla la memoria usada
- /proc/[PID]/status muestra los datos de un proceso específico

Medición de recursos y llamadas al sistema

El kernel ofrece estadísticas detalladas sobre:

- Uso de CPU por proceso
- Memoria virtual y física asignada
- Cantidad de system calls ejecutadas

Por ejemplo, cat /proc/stat muestra contadores globales del sistema

Las system calls (como read, write, fork, mmap) son las puertas de acceso desde el espacio de usuario al kernel

Aplicaciones prácticas

Estas herramientas permiten:

- **Identificar** procesos que consumen demasiada memoria o CPU
- **Detectar** fugas de memoria o bucles infinitos
- **Analizar** el rendimiento de aplicaciones específicas en entornos productivos.

En definitiva, el sistema /proc convierte al kernel en una fuente de datos dinámica accesible al administrador.

13. Systemd Units y Targets

¿Qué es una Unit en Systemd? Una Unit (unidad) es el bloque fundamental de configuración en Systemd.

Cada unidad representa un recurso del sistema o una tarea administrativa, como puede ser:

- Un servicio (por ejemplo, sshd.service o nginx.service)
- Un socket de comunicación (cups.socket)
- Un dispositivo o periférico (dev-sda1.device)
- Un punto de montaje (home.mount)
- O un target, que agrupa varias unidades bajo una misma finalidad

Systemd ve el sistema como un conjunto de unidades interdependientes, donde cada una puede tener dependencias, condiciones y acciones específicas.

Cada unit se define mediante un archivo de texto plano con la extensión correspondiente, ubicado normalmente en:

- /lib/systemd/system/
- /etc/systemd/system/

Tipos de Units

- **Service Units (.service)** Representan servicios o daemons. **Ejemplo**: network.service, sshd.service.

- **Socket Units (.socket)** Se asocian a servicios que se activan cuando se recibe una conexión (activación bajo demanda)
- **Target Units (.target)** Agrupan y sincronizan conjuntos de unidades. Por ejemplo, multi-user.target o graphical.target
- **Mount Units (.mount)** Representan puntos de montaje del sistema de archivos
- **Device Units (.device)** Asociadas a dispositivos detectados por el kernel

Archivos y comandos más usados

En la práctica, trabajamos con dos herramientas: service y systemctl. Ambas permiten iniciar, detener o consultar servicios, pero systemctl es la moderna y recomendada.

Comando clásico	Equivalente en systemctl	Descripción
`service <nombre> start`	`systemctl start <nombre>`	Inicia un servicio
`service <nombre> stop`	`systemctl stop <nombre>`	Detiene un servicio
`service <nombre> restart`	`systemctl restart <nombre>`	Reinicia un servicio
`service <nombre> reload`	`systemctl reload <nombre>`	Recarga configuración
`service <nombre> status`	`systemctl status <nombre>`	Muestra el estado
`service --status-all`	`systemctl list-units`	Lista servicios activos

Targets: agrupaciones de Units

Un **target** es una meta unidad que representa un estado global del sistema

Por ejemplo:

- **basic.target** se carga después de los servicios básicos
- **multi-user.target** prepara el entorno de línea de comandos con red
- **graphical.target** incluye además la interfaz gráfica (X11, GNOME, etc.)

Durante el arranque, Systemd va activando targets de forma progresiva, resolviendo dependencias entre ellos. Este mecanismo reemplaza a los antiguos runlevels de SysVinit.

- Podemos listar los targets con: **systemctl list-units –type=target**
- Y cambiar el modo de ejecución con: **systemctl isolate multi-user.target**
- O para modo gráfico: **systemctl isolate graphical.target**

Ejemplo de flujo durante el arranque

1. **Systemd** (**PID 1**) inicia tras el kernel.
2. Carga el target **default**, que suele ser **multi-user.target o graphical.target.**
3. Este target depende de otros: **network.target, local-fs.target**, etc.
4. A su vez, cada uno de ellos carga sus servicios (**sshd.service, dbus.service**, etc.)
5. Finalmente, el usuario puede iniciar sesión.

Todo este proceso ocurre **en paralelo**, optimizando los tiempos de arranque y evitando bloqueos innecesarios.