APUNTES 1er PARCIAL S.O

Tema 1: Introducción y estructura del sistema operativo

1. ¿Qué es un sistema operativo?

Un conjunto de programas que permiten usar y gestionar los recursos de la máquina, proporcionando servicios y abstracciones al usuario y a otros programas.

2. Menciona tres ventajas de utilizar un sistema operativo.

Reutilización de software, abstracción del hardware y gestión eficiente de los recursos.

3. ¿Qué es una máquina abstracta en el contexto de los sistemas operativos?

Una máquina ficticia creada por el SO que ofrece dispositivos virtuales (ficheros, procesos, conexiones, etc.) para ocultar los detalles del hardware.

4. ¿Qué es el kernel?

Es el núcleo del sistema operativo, responsable de gestionar el hardware y los servicios básicos, ejecutándose en modo privilegiado.

5. ¿Qué modos de ejecución existen en la CPU?

Modo usuario (ring 3) y modo kernel (ring 0).

6. ¿Qué es una llamada al sistema?

Es el mecanismo por el cual un programa solicita servicios del kernel pasando de modo usuario a modo kernel.

7. ¿Para qué sirve la API en un sistema operativo?

Permite a los programas interactuar con el sistema operativo mediante funciones estándar de acceso a servicios y recursos.

8. ¿Qué diferencia hay entre un proceso y un programa?

Un programa es un conjunto de instrucciones en reposo; un proceso es un programa en ejecución, con flujo de control y recursos propios.

9. ¿Qué abstracciones importantes proporciona un SO?

Procesos, ficheros, directorios, conexiones de red, ventanas, dispositivos virtuales, etc.

10. ¿Qué significa multiplexar recursos en tiempo y espacio?

Multiplexar en tiempo: repartir el uso de la CPU y otros recursos a diferentes procesos a lo largo del tiempo.
En espacio: repartir memoria, disco, etc., entre procesos sin que se solapen.

11. ¿Qué es el nivel de privilegio ring 0?

Es el modo privilegiado donde funciona el kernel permitiendo acceso a instrucciones privilegiadas y hardware.

12. ¿Qué hace el firmware al arrancar un sistema?

Realiza operaciones de comprobación y carga el cargador de arranque, que finalmente inicia el kernel.

13. ¿Qué ocurre tras arrancar el kernel?

Inicializa hardware, estructuras internas y crea los primeros procesos de usuario, como init o systemd.

14. ¿Qué relación padre-hijo se da entre procesos en el arranque?

Los procesos nuevos son creados por otros procesos siguiendo una relación jerárquica padre-hijo.

15. ¿Qué es un kernel monolítico?

Un único programa que contiene todos los servicios del SO. Ejemplo: Linux.

16. ¿Qué es un microkernel?

Un kernel reducido a lo mínimo (gestión de hardware, memoria y comunicación). El resto de los servicios van en servidores en espacio de usuario.

17. ¿Qué es un kernel híbrido?

Compromiso entre monolítico y microkernel; incluye componentes en el kernel y otros en espacio de usuario. Ejemplos: XNU (OSX), Windows NT.

18. ¿Qué ventajas tiene un kernel modular?

Se pueden cargar y descargar módulos dinámicamente, adaptando la funcionalidad a las necesidades.

19. ¿Cómo se listan los módulos cargados en Linux?

Con el comando **lsmod.**

20. ¿Qué es la virtualización?

Ejecutar múltiples sistemas operativos o instancias sobre el mismo hardware físico gracias a un hipervisor.

21. Diferencia básica entre máquina virtual de proceso y de sistema.

De proceso: ejecuta un único programa (ej. JVM).
De sistema: ejecuta sistemas operativos completos (ej. VirtualBox).

22. ¿Qué es la paravirtualización?

El sistema huésped está modificado para interactuar directamente con el hipervisor a través de hypercalls**.**

23. ¿Qué son los contenedores?

Entornos aislados dentro del mismo SO que utilizan recursos y abstracciones como si fueran máquinas independientes, pero comparten kernel.

24. Indica pros y contras de los contenedores frente a máquinas virtuales.

Pros: Arranque rápido, menor consumo de recursos, más ligeros.
Contras: Menos aislamiento, menor seguridad que una VM.

25. Ejemplo de sistemas operativos modernos y derivados de Unix.

Linux, FreeBSD, macOS, Solaris, Android, NetBSD, OpenBSD, entre otros

Tema 2: GNULinux y comandos básicos

1. **¿Qué es un comando en GNU/Linux?**
   Es una cadena de texto que identifica a un programa o una orden que el sistema puede ejecutar**.**
2. **¿Qué es un shell?**
   Es el programa que interpreta y ejecuta los comandos del usuario, como bash.
3. **¿Qué hace un shell al recibir una línea de comandos?**
   Lee la línea, realiza sustituciones necesarias, y crea procesos para ejecutar los comandos.
4. **¿Qué es el prompt?**
   El texto que indica que el shell está esperando una orden del usuario.
5. **¿Qué es el login name?**
   El nombre de usuario con el que se inicia sesión y bajo el que se ejecutan los programas.
6. **¿Quién es el usuario root?**
   El superusuario o administrador del sistema, con permisos totales.
7. **¿Cómo están organizados los ficheros y directorios en Linux?**
   En un árbol que parte del directorio raíz /.

8. **¿Qué es el 'directorio de trabajo'?**
   Es el directorio actual sobre el que opera el usuario; se consulta con el comando pwd.

9. **¿Qué es el 'directorio HOME'?**
   El directorio personal de cada usuario.

10. **¿Qué contiene el directorio /bin?**
    Los ejecutables esenciales del sistema.

11. **¿Para qué sirve /etc?**
    Contiene los archivos de configuración del sistema.

12. **¿Qué es /home?**
    Directorio donde se guardan los datos personales de los usuarios.

13. **¿Qué contienen /lib y /usr/lib?**
    Bibliotecas compartidas usadas por los programas ejecutables.

14. **¿Para qué sirven /proc y /sys?**
    Ofrecen una interfaz para interactuar con el núcleo del sistema y la información de procesos.

15. **¿Para qué sirve el directorio /tmp?**
    Almacenar ficheros temporales, que suelen borrarse al reiniciar.

16. **¿Qué diferencias hay entre ruta absoluta y relativa?**
    Absoluta: camino completo desde la raíz /.
    Relativa: desde el directorio actual.

17. **¿Cómo se accede al directorio anterior? ¿Y al actual?**
    .. accede al directorio padre**; .** al directorio actual.

18. **¿Qué es un fichero de texto plano?**
    Un fichero que almacena solo caracteres representables, legibles por humanos.

19. **¿Para qué sirve el comando man?**
    Para consultar el manual de uso de comandos y funciones del sistema.

20. **¿Cómo buscar ayuda sobre una palabra?**
    Con el comando **apropos <palabra>**

    Comandos(1), llamadas al sist(2), llamas a biblio(3)

21. **¿Comando para cambiar de directorio?**
    cd <directorio>

22. **¿Para qué sirve ls?**
    Listar el contenido de un directorio.

23. **¿Comando para copiar y mover ficheros?**
    Copiar**: cp origen destino**
    Mover**: mv origen destino**

24. **¿Comando para ver los usuarios conectados y el propio nombre de usuario?**
    Ver usuarios**: who**
    Ver tu usuario**: whoami**

25. **¿Qué son y para qué sirven los wildcards (globbing)?**
    Son caracteres especiales para referirse a múltiples archivos:
    ***** (cualquier secuencia)
    ? (cualquier carácter)
    [a-z] (rango de caracteres), útiles para búsquedas y operaciones masivas en la terminal.

Comandos por terminal:

- **cd**: cambia de directorio actual.
- **echo**: escribe sus argumentos por su salida.
- **touch**: cambia la fecha de modificación de un fichero. Si no existe el fichero, se crea.
- **ls**: lista el contenido de un directorio.
- **cp**: copia ficheros.
- **mv**: mueve ficheros.
- **rm**: borra ficheros.
- **mkdir**: crea directorios.
- **rmdir**: borra directorios vacíos.
- **date**: muestra la fecha
- **who**: muestra los usuarios que están en el sistema.
- **whoami**: muestra tu nombre de usuario.
- **sort**: ordena las líneas de un fichero.
- **wc**: cuenta caracteres, palabras y líneas de ficheros.
- **fgrep**, **grep**: buscan cadenas dentro de ficheros.
- **cmp**, **diff**: comparan ficheros.
- **cat**: escribe en su salida el contenido de uno o varios ficheros.
- **less**: permite leer un fichero de texto en el terminal usando Scroll
- **file**: da pistas sobre el contenido de un fichero.
- **od**: escribe en su salida en los datos de un fichero en distintos formatos.
- **head, tail:** escriben en las primeras/últimas líneas del fichero en su salida.
- **tar**: crea un fichero con múltiples ficheros dentro (comprimidos o no).
- **gzip/gunzip**: comprime/descomprime un fichero.
- **top**: muestra los procesos y el estado de sistema.
- **reset**: restablece el estado del terminal.
- **exit**: el shell termina su ejecución.

Tema 3: Programación en C sobre GNULinux

1. Características de C

- Lenguaje imperativo estructurado, bastante “cercano” al hardware.
- Tipado débil, la mayoría de errores de tipo se detectan en ejecución.
- Todo programa en C empieza ejecutando la función main.

2. Estructura mínima de un programa en C

```c
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char *argv[]) {
    printf("Hola mundo\n");
    return 0; // También válido: exit(EXIT_SUCCESS);
}
```

3. Tipos básicos y variables

int, char, unsigned, long, float ICULF

4. Control de flujo

```c
if (x > 0) { printf("Positivo\n"); }
else if (x < 0) { printf("Negativo\n"); }
else { printf("Cero\n"); }

for (int i = 0; i < 5; i++) { printf("%d\n", i); }

while (n > 0) { n--; }
do { n++; } while (n < 5);

switch (op) {
    case 1: ...; break;
    case 2: ...; break;
    default: ...;
}
```

5. Funciones

```c
int suma(int a, int b) {
    return a + b;
}
```

6. Arrays y cadenas

```c
int lista[5] = {1,2,3,4,5};
char saludo[] = "Hola";          // Equivalente: char saludo[5] = {'H','o','l','a','
printf("%c\n", saludo[0]);       // Primer carácter

// Recorrer array
for (int i = 0; i < 5; i++) {
    printf("%d ", lista[i]);
}
```

7. Punteros

```c
int x = 10;
int *p = &x;            // p apunta a x
printf("%d", *p);       // imprime el valor de x
```

8. Memoria dinámica

```c
#include <stdlib.h>
int *v = malloc(10 * sizeof(int));
v[0] = 1;
free(v);    // Liberar memoria reservada
```

9. Estructuras

```c
typedef struct {
    int x, y;
} Punto;

Punto p;
p.x = 1;
p.y = 2;
```

Tema 4: Procesos

1. ¿Qué es un proceso?

Un programa en ejecución, con su propio flujo de control, memoria, recursos y estado.

2. ¿Cómo se crea un proceso en sistemas UNIX?

Usando la llamada al sistema fork().

3. ¿Qué función cumple la llamada al sistema exec()?

Reemplaza el código y datos de un proceso por los de otro programa.

4. ¿Qué es un PID?

Número entero único que identifica a cada proceso en el sistema (Process ID).

5. ¿Qué diferencia hay entre un proceso y un programa?

Proceso: instancia en ejecución de un programa, con estado y recursos**.**

Programa: conjunto de instrucciones en disco.

6. ¿Qué es el estado de un proceso?

Información que el sistema mantiene sobre cada proceso: valores de registros, estado, memoria, etc.

7. ¿Qué estados posibles puede tener un proceso?

- Ejecutando (Running)
- Listo (Ready)
- Bloqueado (Blocked/Wating)

8. ¿Qué ocurre cuando un proceso llama a wait()?

El proceso padre espera que termine un proceso hijo y recoge su estado.

9. ¿Qué es un proceso zombie?

Proceso terminado cuyo estado no ha sido leído por su padre (aún en la tabla de procesos).

10. ¿Qué ocurre con los procesos huérfanos?

Son adoptados por el proceso init, que los recolecta usando wait() para evitar zombies.

11. ¿Qué hace el comando ps?

Muestra información sobre los procesos activos (PID, estado, memoria, etc.).

12. ¿Para qué sirve /proc en Linux?

Proporciona información y acceso a datos internos de los procesos y del kernel.

13. Diferencia entre enlace estático y dinámico de un ejecutable.

- Estático: todas las bibliotecas incluidas en el binario.
- Dinámico: las bibliotecas se cargan en tiempo de ejecución solo cuando sean necesarias.

14. ¿Qué es la memoria virtual por proceso?

Cada proceso cree que tiene su propio espacio de direcciones (aislamiento).

15. ¿Cuáles son los segmentos habituales en la memoria de un proceso?

- **.text:** código
- **.data:** datos inicializados
- **.bss:** datos sin inicializar
- **Heap:** memoria dinámica
- **Stack:** pila de ejecución

16. ¿Qué hacen las llamadas getpid(), getppid(), getuid() y getgid()?

- **getpid():** Obtiene PID propio.
- **getppid():** Obtiene PID del padre.
- **getuid():** Obtiene el UID (usuario).
- **getgid():** Obtiene el GID (grupo).

17. ¿Cómo se gestiona la terminación de un proceso?

Con la llamada **exit**() y la recolección de su estado por el padre con **wait**().

18. ¿Qué es el valor de retorno de un proceso y cómo lo recoge el padre?

El proceso hijo termina con exit(codigo).
El padre lo recoge con wait() y la macro **WEXITSTATUS**(status).

19. ¿Para qué sirve el comando strace?

Rastrea las llamadas al sistema que realiza un proceso en tiempo real.

20. ¿Qué es una condición de carrera entre procesos?

Cuando el resultado depende del orden en que se ejecutan los procesos concurrentes.

21. Nombra y explica algoritmos de planificación de la CPU.

- FCFS: primero en llegar, primero en servir.
- SJF: proceso con menor ráfaga de CPU primero.
- SRTF: proceso con menor tiempo restante primero.
- Round Robin: reparto circular con cuanto de tiempo fijo.

22. ¿Qué diferencia hay entre planificación expulsiva y no expulsiva?

Expulsiva: el SO puede quitar la CPU a un proceso antes de que termine su ráfaga.
No expulsiva: solo se cede la CPU voluntariamente o por bloqueo.

23. ¿Qué es el cambio de contexto?

El proceso de salvar el estado de un proceso (registros, pila, etc.) para cargar el de otro.

24. ¿Qué es el comando nice y renice?

nice: ejecuta un proceso con una prioridad específica.
renice: cambia la prioridad de un proceso en ejecución.

25. ¿Qué son los descriptores de fichero en procesos?

Identificadores de recursos abiertos (archivos, sockets) por un proceso.
Por defecto: 0 (stdin), 1 (stdout), 2 (stderr).

26. ¿Qué es la variable errno? ¿Y la función perror()?

errno: almacena el código de error de la última llamada fallida.
perror(): imprime el mensaje correspondiente a ese error.

27. ¿Para qué sirve el comando ltrace?

Para rastrear las llamadas a funciones de biblioteca que hace un proceso.

28. ¿Qué consecuencias tiene un mal uso de la planificación?

Puede aparecer inanición (starvation) o baja eficiencia del sistema.

29. ¿Qué es aging en el contexto de planificación?

Técnica para evitar que procesos de baja prioridad nunca sean atendidos, aumentando su prioridad cuanto más esperan.

30. ¿Qué hacen las señales en procesos?

Permiten comunicación asíncrona (ej: para terminar, pausar, continuar o capturar eventos).