

Los procesos

Sistemes Operatius I

Oliver Díaz

Grau d'Enginyeria Informàtica

Que veremos hoy

- Temes pendientes
 - Visita virtual BSC-Marenostrum
- Procesos
- Llamadas a Sistema
- Interfície de programación
- `fork()` y `exec()`
- Fork-bomb



Asistencia

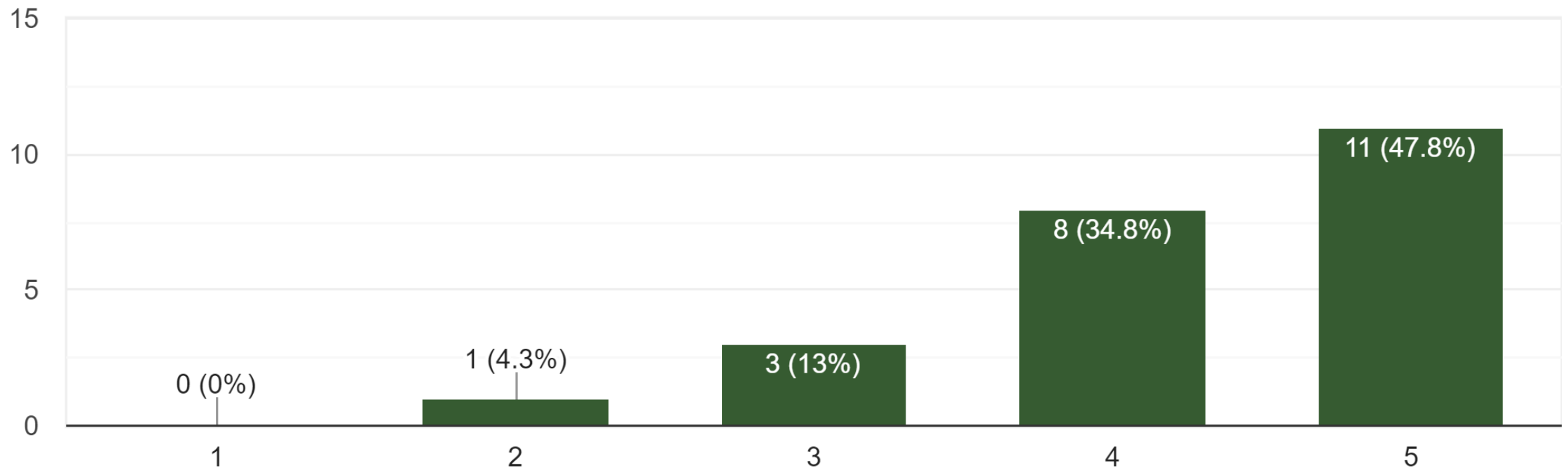
Formulario Google

CV – Teoria (Presencial) o QR

Visita Virtual BSC-Marenostrum

¿Qué te pareció la presentación (en una escala del 1 al 5)?

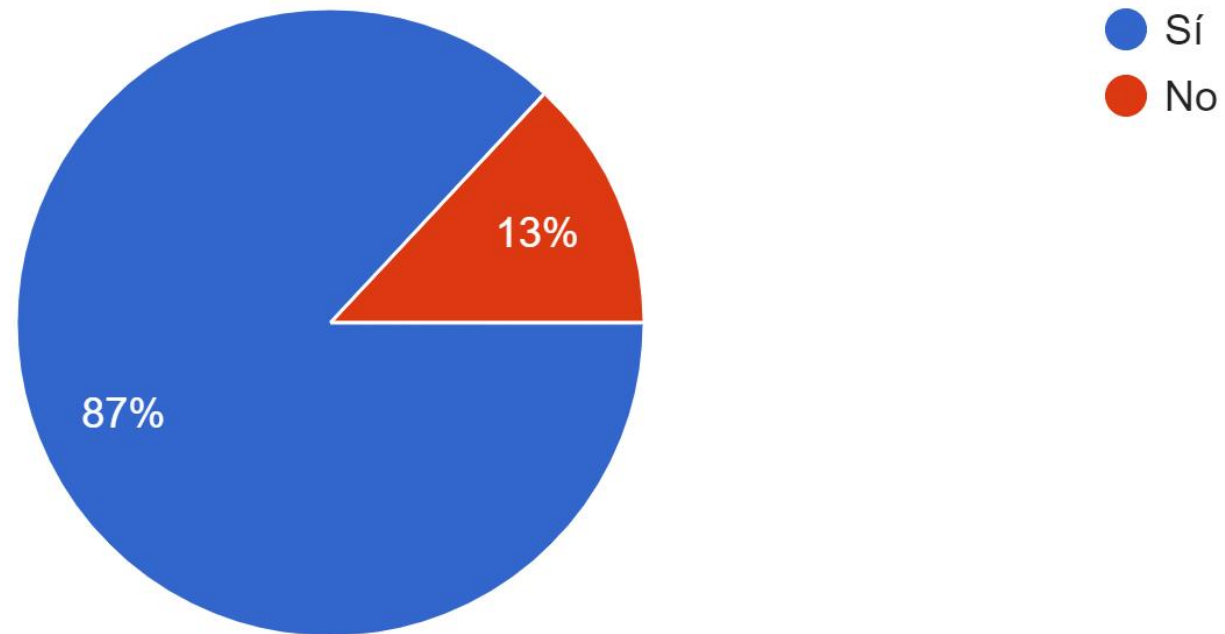
23 responses



Visita Virtual BSC-Marenostrum

¿Recomendarías la presentación para futuros cursos de la asignatura?

23 responses



Visita Virtual BSC-Marenostrum

- **Hi ha pràctiques que es puguin fer a l'estiu?**

A la pagina web surten totes les activitats, tant actuals com estiu:

<https://www.bsc.es/ca/uneix-te>

- **Por qué se utiliza Linux como SO? Es más rápido? Más fiable? Creo que es el punto más importante de cara a nuestra asignatura, sería muy interesante saberlo.**

MareNostrum 1 va ser el 1er supercomputadors (SC) en el top500 (www.top500.org) en utilitzar SO Linux i tecnologia comercial standard. Ara mateix tots els SCs del top500 en porten (qualsevol distribucio de linux). És un tema mes de eficiencia que de seguretat (que tambe).

Visita Virtual BSC-Marenostrum

- **¿Cuál es el proyecto que ve con mayor potencial dentro del centro Marenostrum?**

Els projectes en marxa actuals els tenim a: <https://www.bsc.es/ca/research-and-development/projects>

No m'atreveixo a destacar cap, es un tema molt subjectiu. Diríem que dins de la Comissió Europea hi han 2 grans projectes: Human Brain (en el que participen i collorem en molts grups) i en el Disseny de Primer Processador Europeu (liderem el projecte).

Tot i aixó tots els projectes relacionats amb la salut son (personalment) molt agrats i ben situats (i necessiten gran capacitat de càlcul): per exemple el simulador del cor.

Visita Virtual BSC-Marenostrum

■ ¿Han sufrido alguna brecha de seguridad en todos estos años?

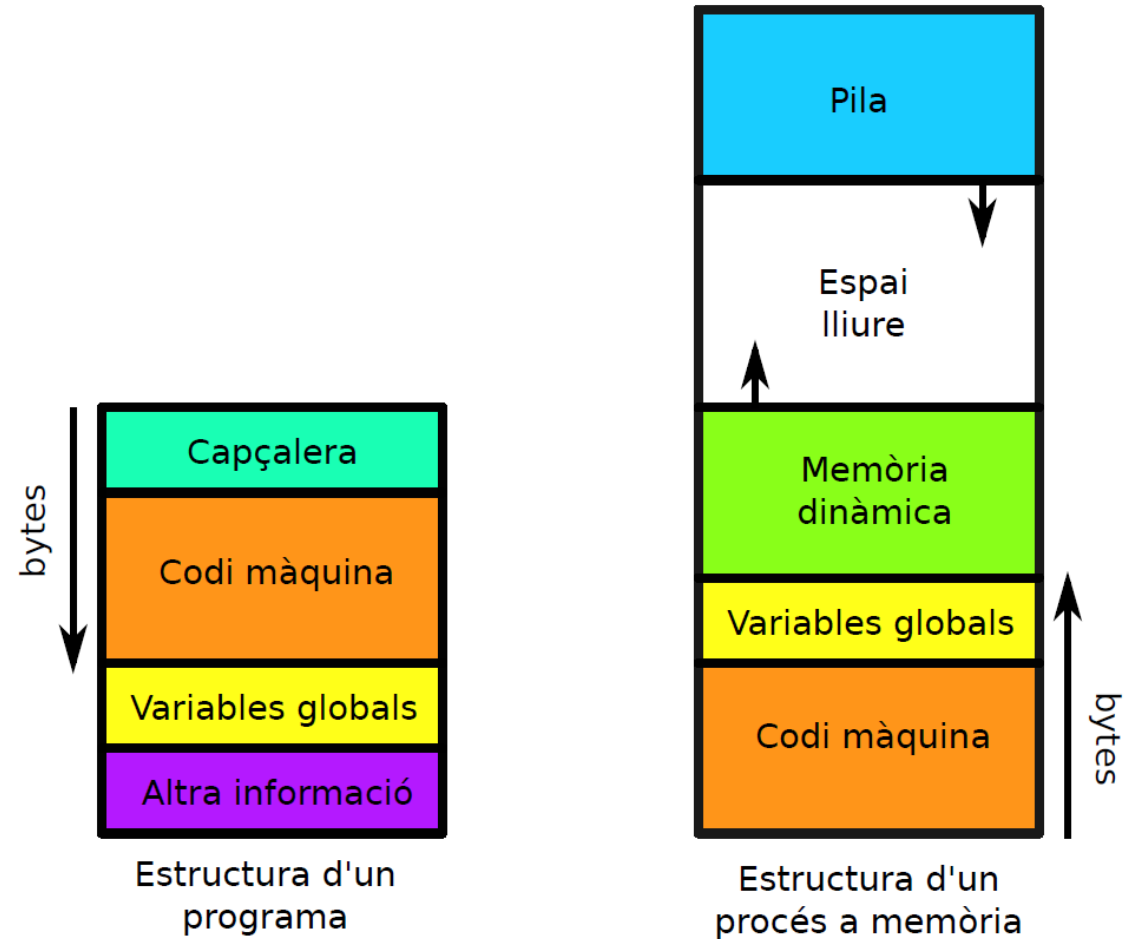
Cada dia passem els parxes de Seguretat habituals corresponents.

Els primers anys haviem tinguts atacs DOS però des de que vam canviar el Firewall ja es detectan abans de tenir el problema.

Tot i aixó aquest tipus de atacs no feien que la maquina deixes de funcionar, només que els usuaris els hi costes accedir més.

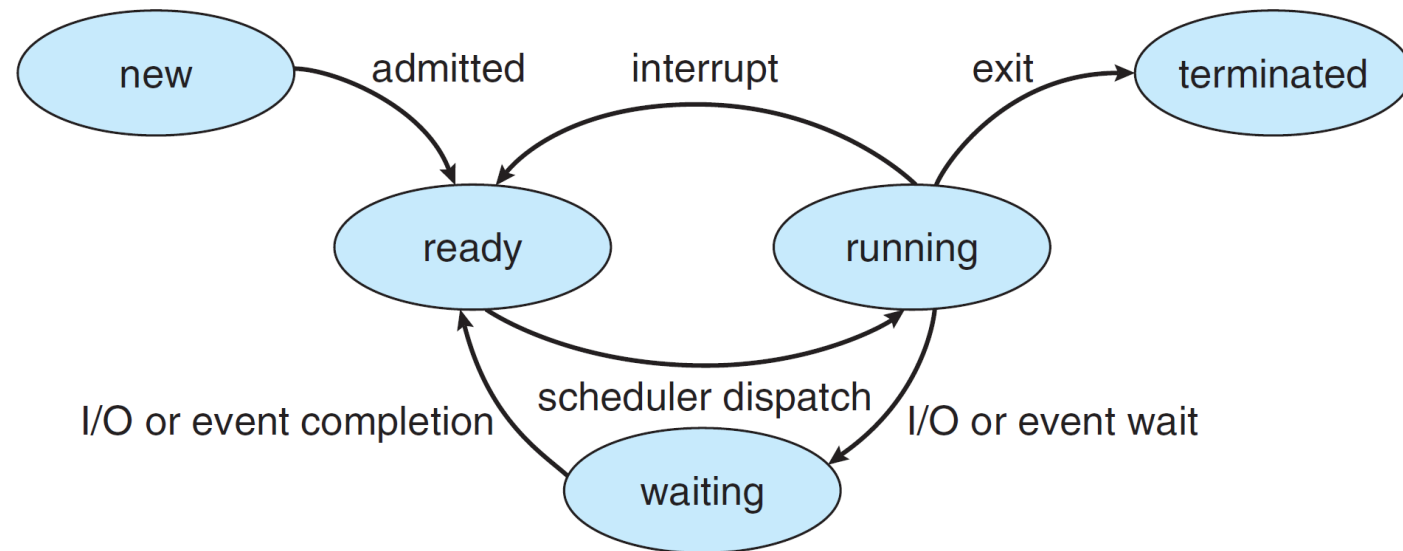
Los procesos

- **Proceso:** Programa en ejecución.



Los procesos

- A medida que el proceso se ejecuta, su estado cambia. El **estado** de un proceso viene determinado por su actividad actual
 - New: esta siendo creado
 - Ready: en espera de ser asignado a procesador
 - Running: esta siendo ejecutado
 - Waiting: en espera de algun evento
 - Terminate: finalización de ejecución



Silberschatz, A., Peterson, J. L., & Galvin, P. B. (2018). *Operating system concepts*. Wiley

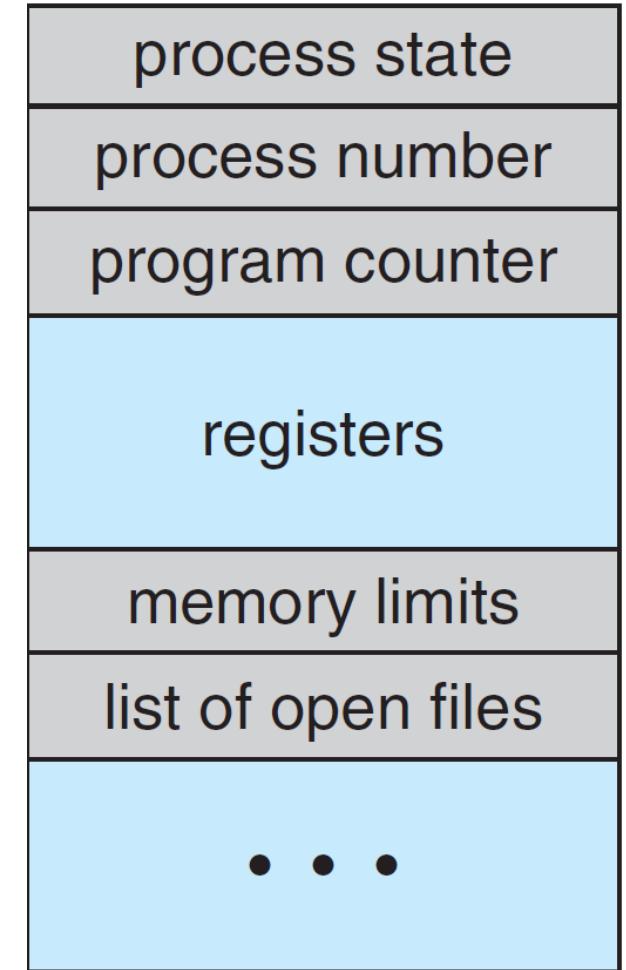
Los procesos

- Un único proceso se encuentra en estado “Running” en un procesador en un instante determinado.
- Muchos procesos pueden estar en modo “Ready” o “Waiting”

Silberschatz, A., Peterson, J. L., & Galvin, P. B. (2018). *Operating system concepts*. Wiley

Process control block (PCB)

- Cada proceso está representado en el SO por PCB.
- Sirve como registro de todos los datos necesarios para iniciar o reiniciar un proceso, junto con algunos datos.
 - Estado: New, Ready, Running,...
 - PID
 - Contador: Dirección siguiente instrucción a ser ejecutada
 - Tiempo CPU, tiempo real, limites tiempo, ...
 - ...




Silberschatz, A., Peterson, J. L., & Galvin, P. B. (2018). *Operating system concepts*. Wiley

Los procesos y llamadas a sistema

- Se requieren **llamadas a sistemas** para comunicar procesos entre sí
- Llamadas a sistemas:
 - Proporcionan una interfaz para los servicios disponibles por un SO.
 - Generalmente disponibles como funciones escritas en alto nivel (C y C++), aunque ciertas tareas de bajo nivel pueden estar escritas en lenguaje ensamblador.
 - Principalmente utilizadas por programas a través del **Application Programming Interface** (API), en lugar de una llamada directa al sistema (p. ej. [POSIX](#)). Ejemplo de API para UNIX:

```
#include <unistd.h>

ssize_t read(int fd, void *buf, size_t count)
```



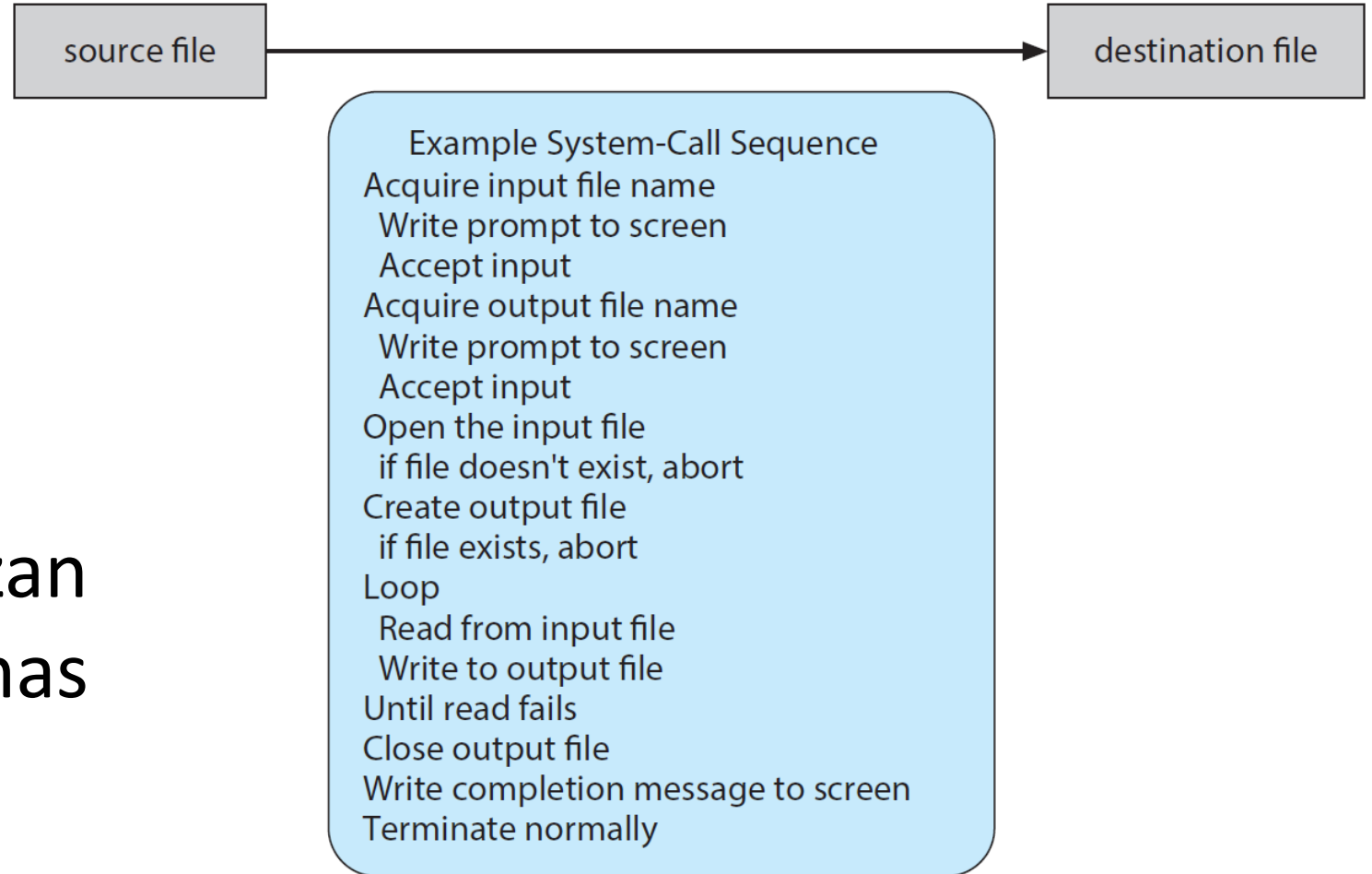
The diagram illustrates the components of the `read` function signature. It shows the return type `ssize_t`, the function name `read`, and the parameters `(int fd, void *buf, size_t count)`. Brackets are used to group these elements and label them as 'return value', 'function name', and 'parameters' respectively.

Silberschatz, A., Peterson, J. L., & Galvin, P. B. (2018). *Operating system concepts*. Wiley

Llamada a sistema

`cp in.txt out.txt`

Frecuentemente se realizan miles de llamadas a sistemas por segundo.



Silberschatz, A., Peterson, J. L., & Galvin, P. B. (2018). *Operating system concepts*. Wiley

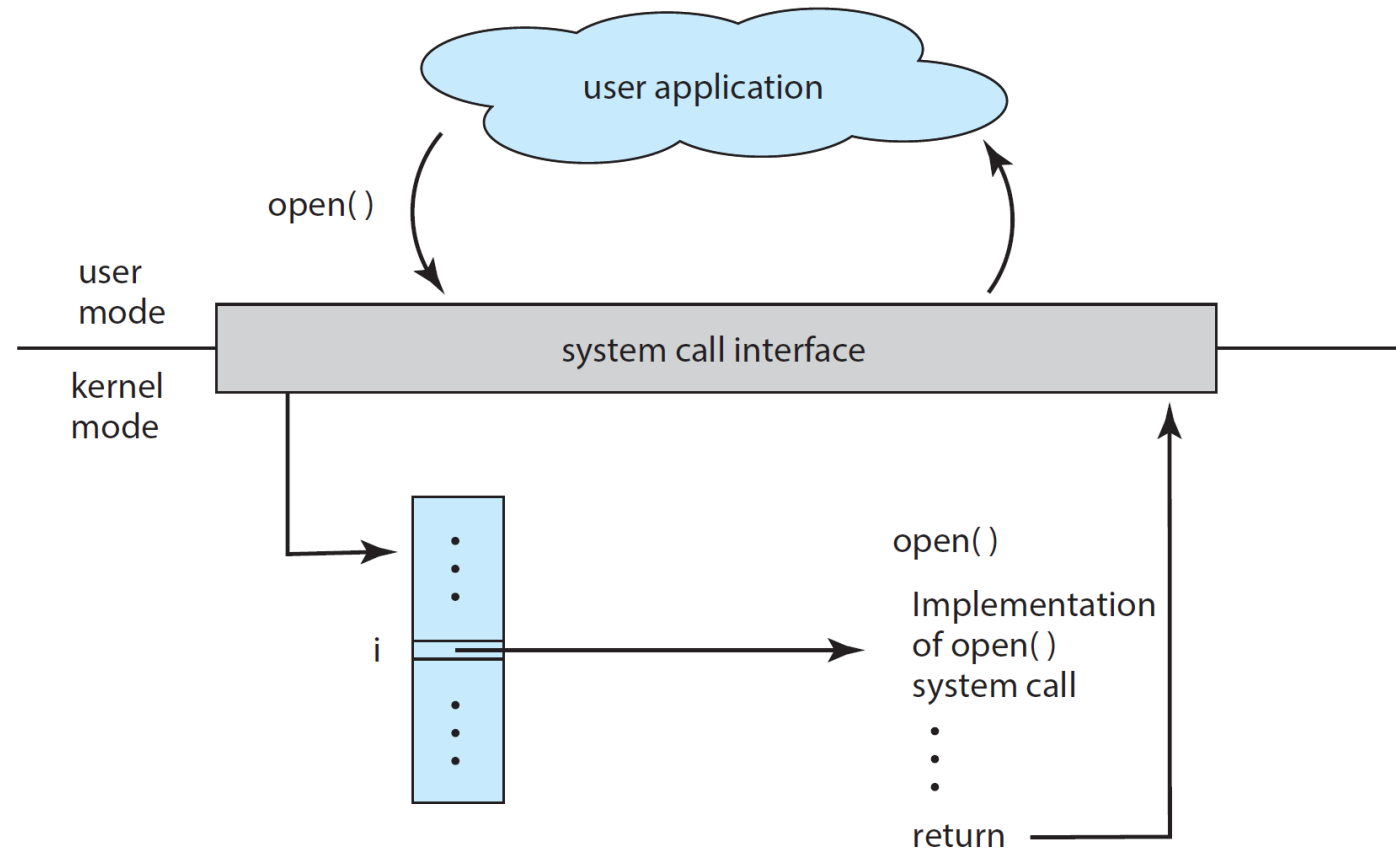
Tipos de llamadas a sistema

	Windows	Unix
Process control	CreateProcess() ExitProcess() WaitForSingleObject()	fork() exit() wait()
File management	CreateFile() ReadFile() WriteFile() CloseHandle()	open() read() write() close()
Device management	SetConsoleMode() ReadConsole() WriteConsole()	ioctl() read() write()
Information maintenance	GetCurrentProcessID() SetTimer() Sleep()	getpid() alarm() sleep()
Communications	CreatePipe() CreateFileMapping() MapViewOfFile()	pipe() shm_open() mmap()
Protection	SetFileSecurity() InitializeSecurityDescriptor() SetSecurityDescriptorGroup()	chmod() umask() chown()

Silberschatz, A., Peterson, J. L., & Galvin, P. B. (2018). *Operating system concepts*. Wiley

Interfície programación

- **API:** funciones disponibles al programador (argumentos y valores de retorno conocidos).
- La **interfície** sirve como enlace entre API y las llamadas al sistema disponibles por el SO.
- Intercepta llamadas a funciones en la API e invoca las llamadas al sistema necesarias dentro del SO



Silberschatz, A., Peterson, J. L., & Galvin, P. B. (2018). *Operating system concepts*. Wiley

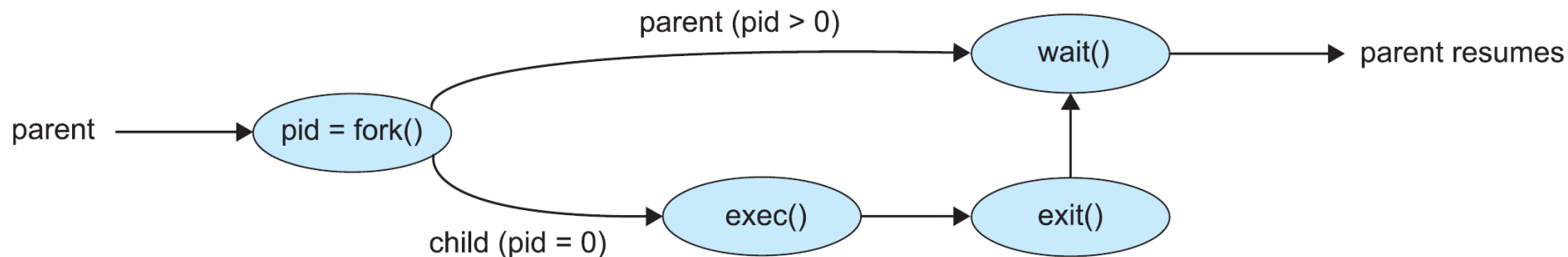
Fork y exec

■ `fork()`

- Crea un nuevo proceso *hijo*, independiente del *padre*
 - Variables propias
- Devuelve dos veces, una para el *padre* y otra para el *hijo*
- El *padre* puede controlar el contexto de su *hijo* (privilegios, tiempo max ejecución,..)

■ `exec()` :

- Después de `fork()`, una de las llamadas a sistema más común en UNIX es `exec()`.
- Carga un archivo binario a memoria e inicia su ejecución
- El padre se puede poner en `wait()` hasta que termine su hijo



Silberschatz, A.,
Peterson, J. L., &
Galvin, P. B.
(2018). *Operating
system concepts*.
Wiley

Fork y exec

- El proceso *hijo* consiste en una copia del espacio de direcciones del proceso *padre*.
- Esto permite que el proceso *padre* se comunique fácilmente con su *hijo*.
- Ambos procesos (*padre* e *hijo*) continúan ejecutándose después de la llamada `fork()`, con una diferencia: el código de retorno:
 - 0 para el nuevo proceso (*hijo*)
 - $\neq 0$ del hijo es devuelto a los padres.

```
int main(void)
{
    int ret;

    ret = fork();

    if (ret == 0) { // fill
        printf("Soc el fill i el meu id es %d\n", getpid());
        return 0;
    } else { // pare
        printf("Soc el pare del proces %d\n", ret);
        return 0;
    }
}
```

Silberschatz, A., Peterson, J. L., & Galvin, P. B. (2018). *Operating system concepts*. Wiley

Fork-bomb

```
0 #include <stdio.h>
1 #include <unistd.h>
2
3 int main(void)
4 {
5     while (1) { fork(); }
6 }
7
```

- ¿Qué pasa si se ejecuta un fork-bomb?
- Gestión de control de procesos. El *padre* puede controlar contexto de *hijo*.
- Ejercicio:
 - Fork-exec-setrlimit.c
 - ulimit

Silberschatz, A., Peterson, J. L., & Galvin, P. B. (2018). *Operating system concepts*. Wiley

¿Quieres saber más?

- Silberschatz, A., Peterson, J. L., & Galvin, P. B. (2018). *Operating system concepts*. Wiley

Gracias