Programación Avanzada (TC2025)

Tema 2. Arquitectura de un sistema operativo

Tecnológico de Monterrey. Campus Santa Fe
Departamento de Computación
Dr. Vicente Cubells (vcubells@tec.mx)

Temario

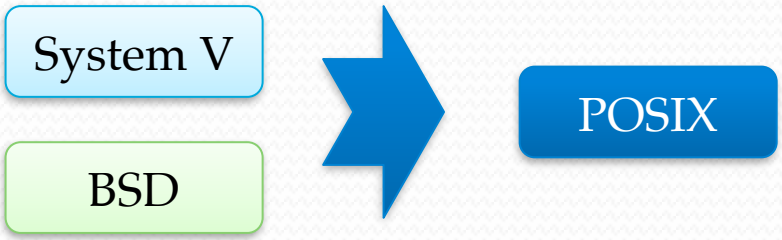
- Arquitectura de UNIX
- Programas y procesos
  - Llamadas al sistema
- Administración de la memoria
  - Llamadas al sistema

Introducción a Unix…

- MULTICS, UNICS….UNIX
- Ken Thompson y Dennis Ritchie
- UNIX y C
- La PDP-11
- XENIX y Microsoft
- System V de AT&T…Novell..SCO
- UNIX Berkeley….La evolución
  - Memoria virtual y Paginación
  - Conectividad TCP/IP
  - Sistema de archivos mejorado (nombres con > 14 chars)
  - vi, csh, compilador Pascal y Lisp

Introducción a Unix

- La estandarización

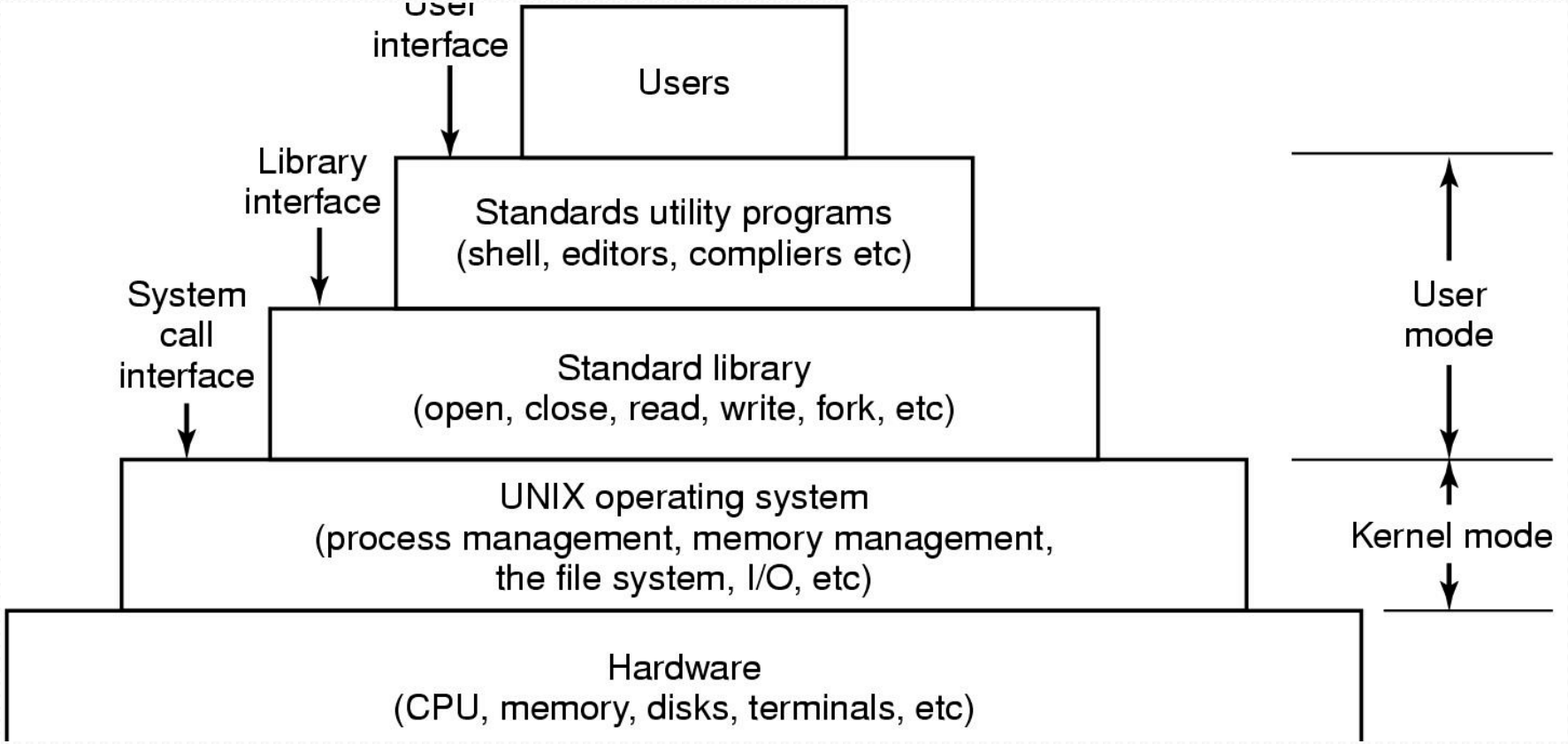


- Tanenbaum y MINIX
  - Un regreso a la idea original: la simplicidad
  - Un microkernel
- Linux: un kernel monolítico
  - 80% de las 150 llamadas al sistema son copias de las de UNIX
- FreeBSD vs Linux

Generalidades de UNIX

- ¿Qué buscan los buenos programadores?
  - Sistemas sencillos, elegantes y consistentes
    - Diferentes tipos de archivos es solo un estorbo
    - Principio de mínima sorpresa
  - Potencia y flexibilidad
    - Número reducido de elementos básicos que pueden combinarse infinitamente
    - Principio básico: todo programa debe hacer una sola cosa y hacerla bien
  - Un sirviente, no una niñera

Interfaces con UNIX



Programas utilitarios de UNIX

Program	Typical use
cat	Concatenate multiple files to standard output
chmod	Change file protection mode
cp	Copy one or more files
cut	Cut columns of text from a file
grep	Search a file for some pattern
head	Extract the first lines of a file
ls	List directory
make	Compile files to build a binary
mkdir	Make a directory
od	Octal dump a file
paste	Paste columns of text into a file
pr	Format a file for printing
rm	Remove one or more files
rmdir	Remove a directory
sort	Sort a file of lines alphabetically
tail	Extract the last lines of a file
tr	Translate between character sets

Algunos programas de UNIX requeridos por POSIX

UNIX Kernel

<table>
<tr><td colspan="5">System calls</td><td colspan="3">Interrupts and traps</td></tr>
<tr><td colspan="2">Terminal handing</td><td>Sockets</td><td>File naming</td><td>Map-ping</td><td>Page faults</td><td rowspan="2">Signal handling</td><td rowspan="2">Process creation and termination</td></tr>
<tr><td rowspan="2">Raw tty
vi, emacs</td><td>Cooked tty
sh</td><td>Network protocols
TCP/IP</td><td>File systems</td><td colspan="2">Virtual memory</td></tr>
<tr><td>Line disciplines</td><td>Routing</td><td>Buffer cache</td><td colspan="2">Page cache</td><td colspan="2">Process scheduling</td></tr>
<tr><td colspan="2">Character devices</td><td>Network device drivers</td><td colspan="3">Disk device drivers</td><td colspan="2">Process dispatching</td></tr>
<tr><td colspan="8">Hardware</td></tr>
</table>

Estructura aproximada de un kernel genérico

Procesos en UNIX

Creación de procesos en UNIX

```
pid = fork( );                  /* if the fork succeeds, pid > 0 in the parent */
if (pid < 0) {
        handle_error( );        /* fork failed (e.g., memory or some table is full) */
} else if (pid > 0) {
                                /* parent code goes here. /*/
} else {
                                /* child code goes here. /*/
}
```

Ejemplo: sort <f | head

POSIX

Señales requeridas por POSIX

Signal	Cause
SIGABRT	Sent to abort a process and force a core dump
SIGALRM	The alarm clock has gone off
SIGFPE	A floating-point error has occurred (e.g., division by 0)
SIGHUP	The phone line the process was using has been hung up
SIGILL	The user has hit the DEL key to interrupt the process
SIGQUIT	The user has hit the key requesting a core dump
SIGKILL	Sent to kill a process (cannot be caught or ignored)
SIGPIPE	The process has written to a pipe which has no readers
SIGSEGV	The process has referenced an invalid memory address
SIGTERM	Used to request that a process terminate gracefully
SIGUSR1	Available for application-defined purposes
SIGUSR2	Available for application-defined purposes

Llamadas al sistema para administración de procesos

System call	Description
pid = fork( )	Create a child process identical to the parent
pid = waitpid(pid, &statloc, opts)	Wait for a child to terminate
s = execve(name, argv, envp)	Replace a process' core image
exit(status)	Terminate process execution and return status
s = sigaction(sig, &act, &oldact)	Define action to take on signals
s = sigreturn(&context)	Return from a signal
s = sigprocmask(how, &set, &old)	Examine or change the signal mask
s = sigpending(set)	Get the set of blocked signals
s = sigsuspend(sigmask)	Replace the signal mask and suspend the process
s = kill(pid, sig)	Send a signal to a process
residual = alarm(seconds)	Set the alarm clock
s = pause( )	Suspend the caller until the next signal

s es un código de error

pid es un process ID

residual tiempo restante en la alarma anterior

POSIX Shell

```
while (TRUE) {                                  /* repeat forever /*/
      type_prompt( );                           /* display prompt on the screen */
      read_command(command, params);            /* read input line from keyboard */

      pid = fork( );                            /* fork off a child process */
      if (pid < 0) {
            printf("Unable to fork0);           /* error condition */
            continue;                           /* repeat the loop */
      }

      if (pid != 0) {
            waitpid (–1, &status, 0);           /* parent waits for child */
      } else {
            execve(command, params, 0);         /* child does the work */
      }
}
```

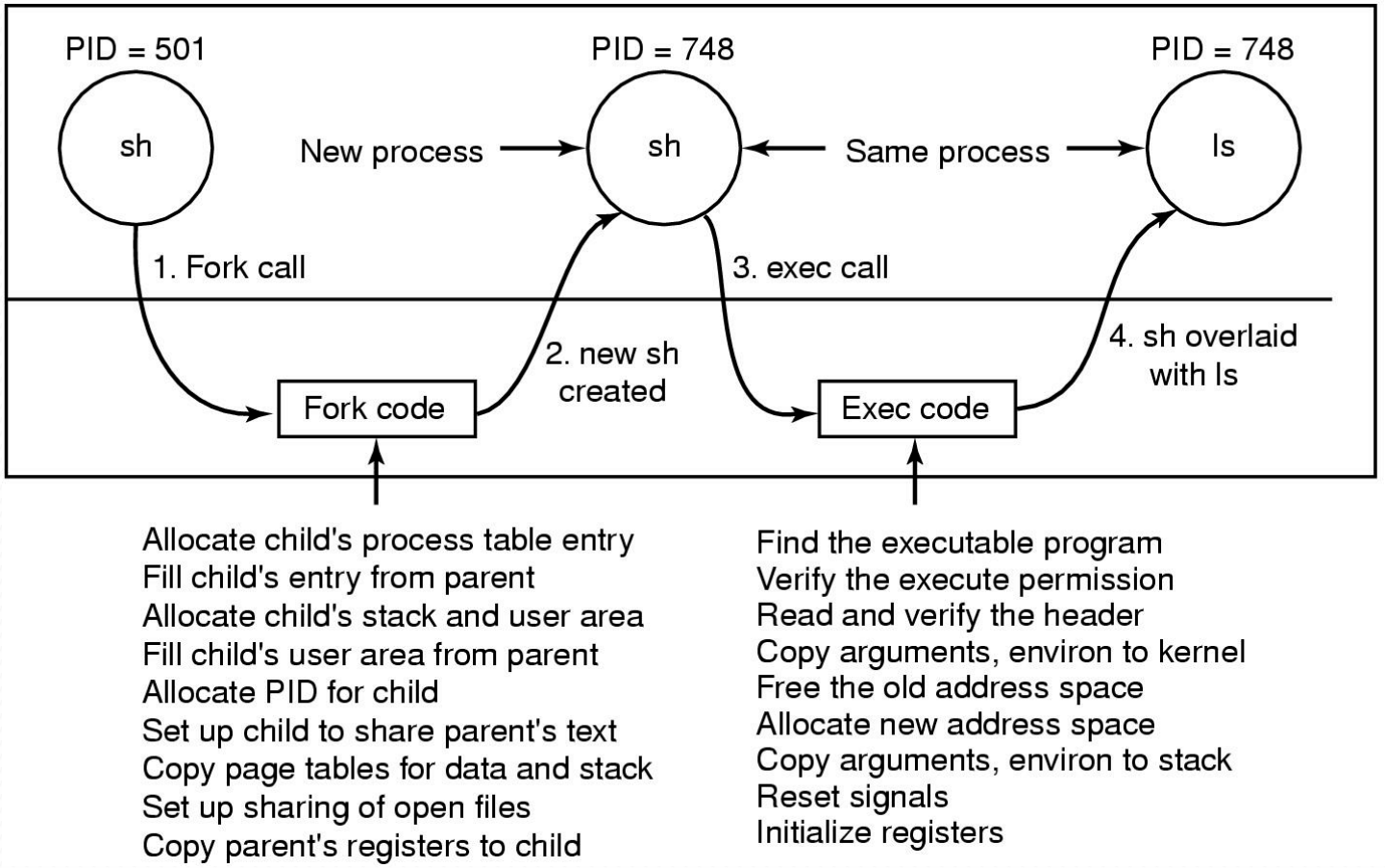
Un shell simplificado

Threads en POSIX

Llamadas para POSIX thread

Thread call	Description
pthread_create	Create a new thread in the caller's address space
pthread_exit	Terminate the calling thread
pthread_join	Wait for a thread to terminate
pthread_mutex_init	Create a new mutex
pthread_mutex_destroy	Destroy a mutex
pthread_mutex_lock	Lock a mutex
pthread_mutex_unlock	Unlock a mutex
pthread_cond_init	Create a condition variable
pthread_cond_destroy	Destroy a condition variable
pthread_cond_wait	Wait on a condition variable
pthread_cond_signal	Release one thread waiting on a condition variable

El comando *ls*



Pasos en la ejecución del comando *ls*

Problemas con los hilos

- Llamada *fork()*
  - Proceso hijo = proceso padre
    - ¿Los mismo hilos?
  - Ejemplo: teclado
    - Dos hilos bloqueados
    - ¿Quién lee, el padre, el hijo o ambos?
- Estructuras de datos compartidas
  - Ejemplo: archivos
- Solicitud de memoria
  - Reservación duplicada

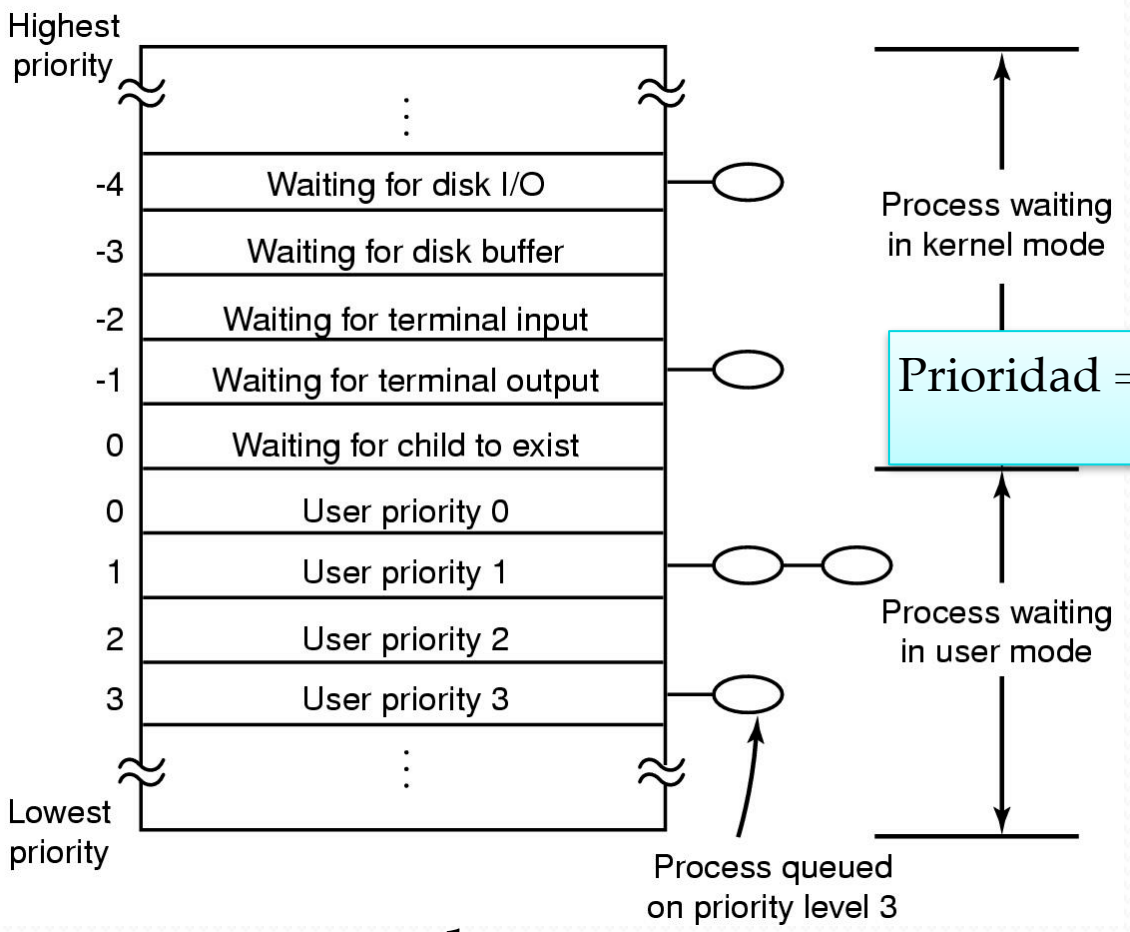
Flags para Linux clone

pid = clone(función, apunt_pila, flags, arg)

Flag	Meaning when set	Meaning when cleared
CLONE_VM	Create a new thread	Create a new process
CLONE_FS	Share umask, root, and working dirs	Do not share them
CLONE_FILES	Share the file descriptors	Copy the file descriptors
CLONE_SIGHAND	Share the signal handler table	Copy the table
CLONE_PID	New thread gets old PID	New thread gets own PID

Significado de los bits en flags

UNIX Scheduler

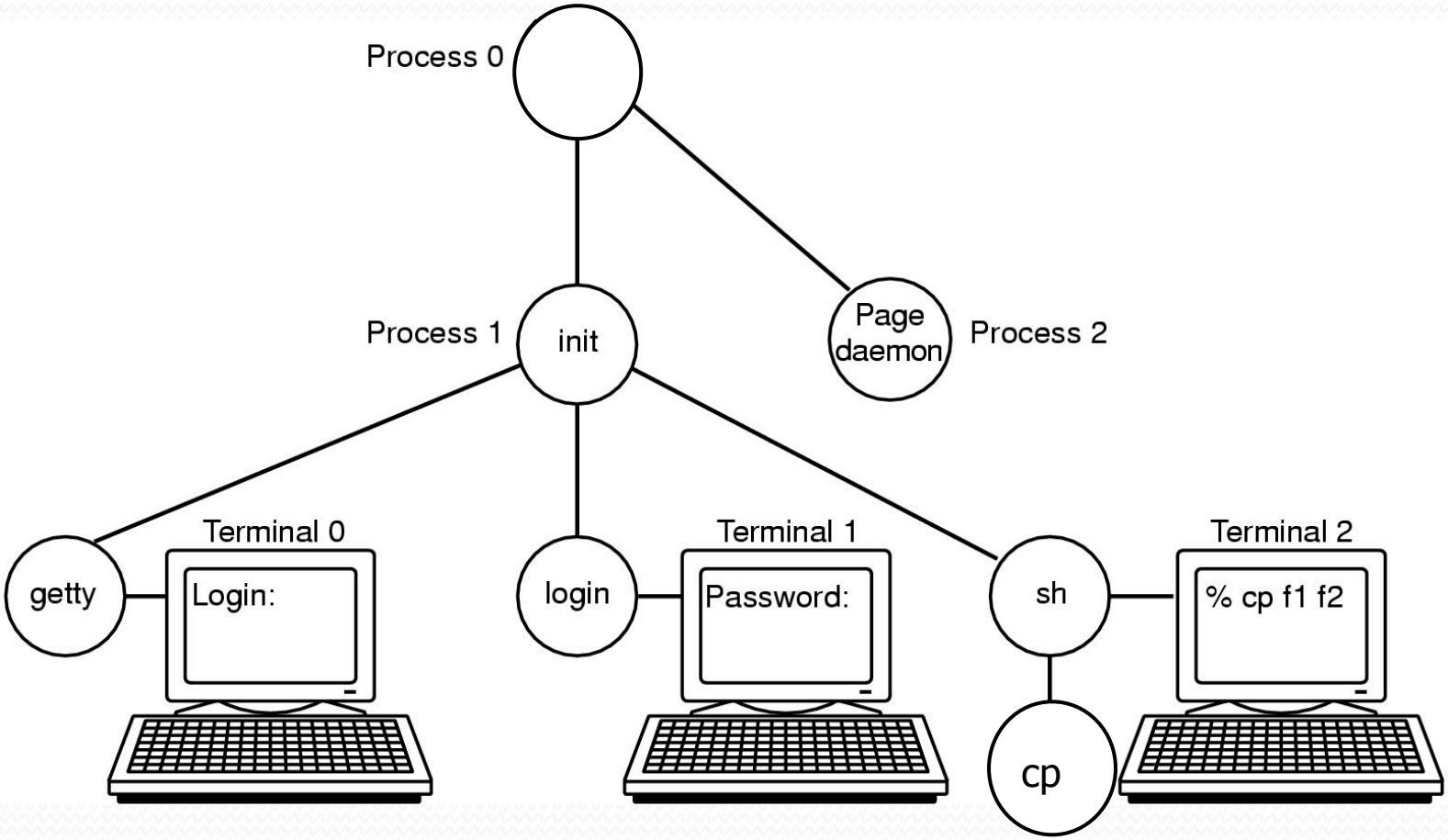


En las colas se utiliza round robin

Prioridad = Consumo_CPU + nice + base

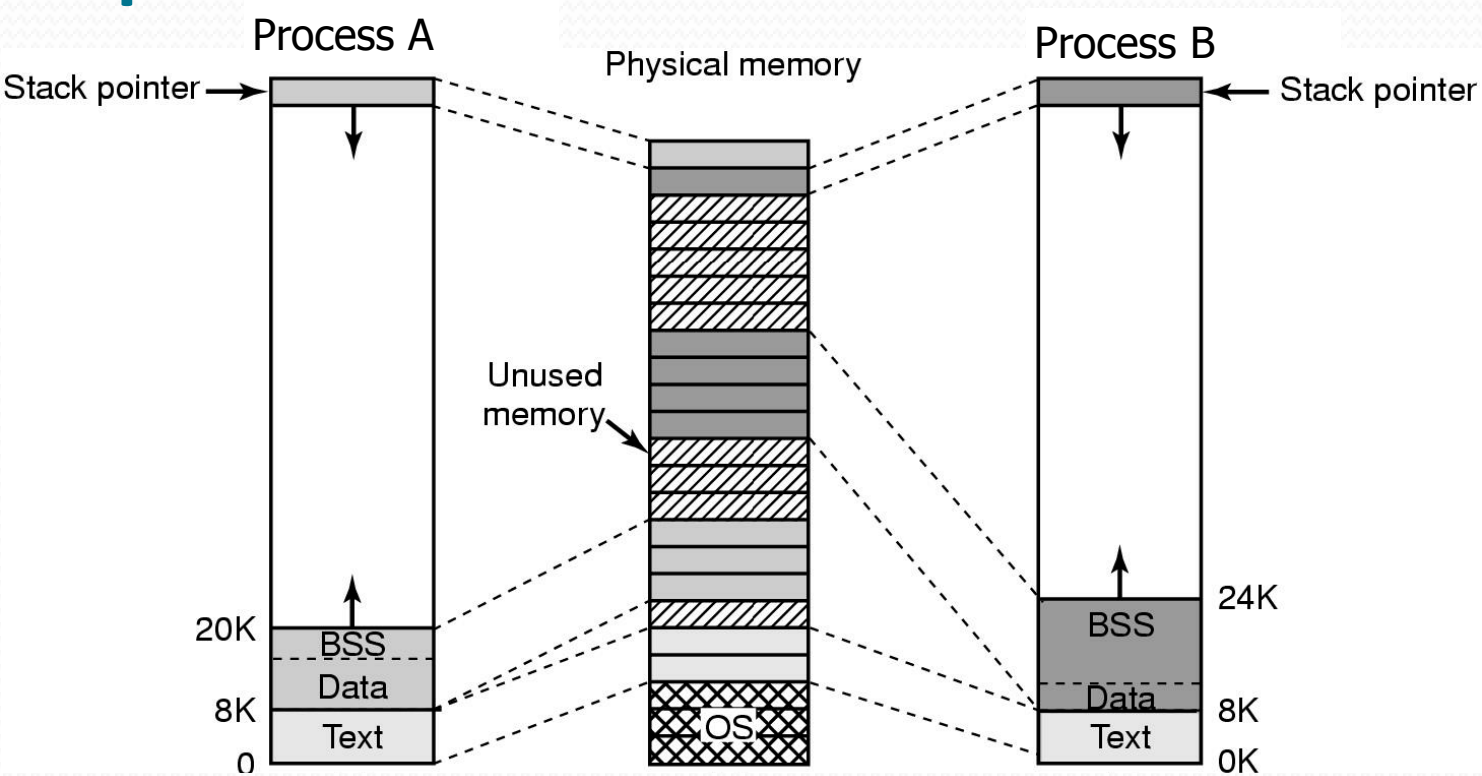
El UNIX scheduler se basa en una cola multinivel

Booting UNIX



Secuencia de procesos para iniciar un sistema

Manipulando la memoria

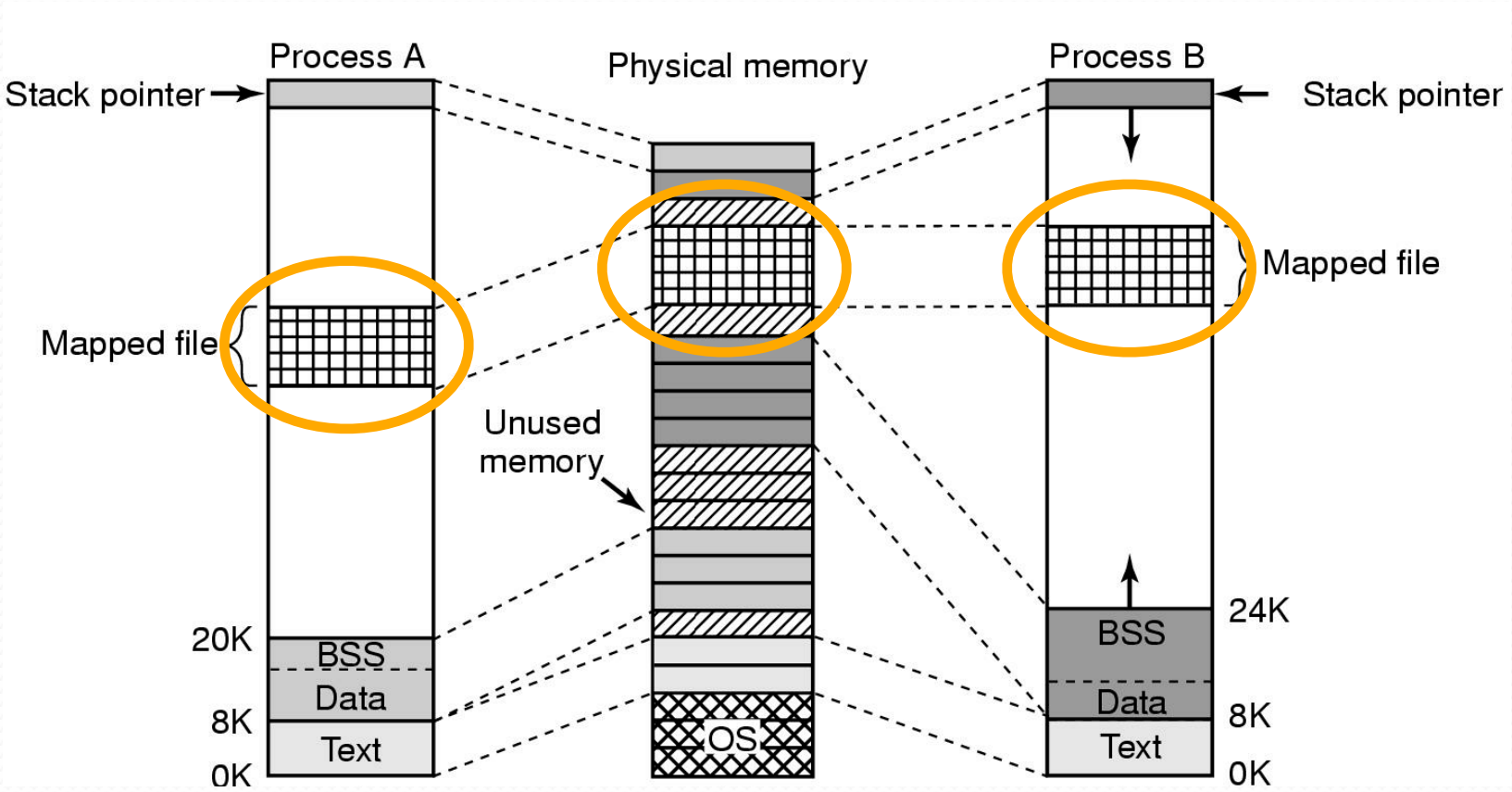


Espacio de direcciones virtuales del proceso A

Direcciones físicas

Espacio de direcciones virtuales del proceso B

Compartiendo archivos



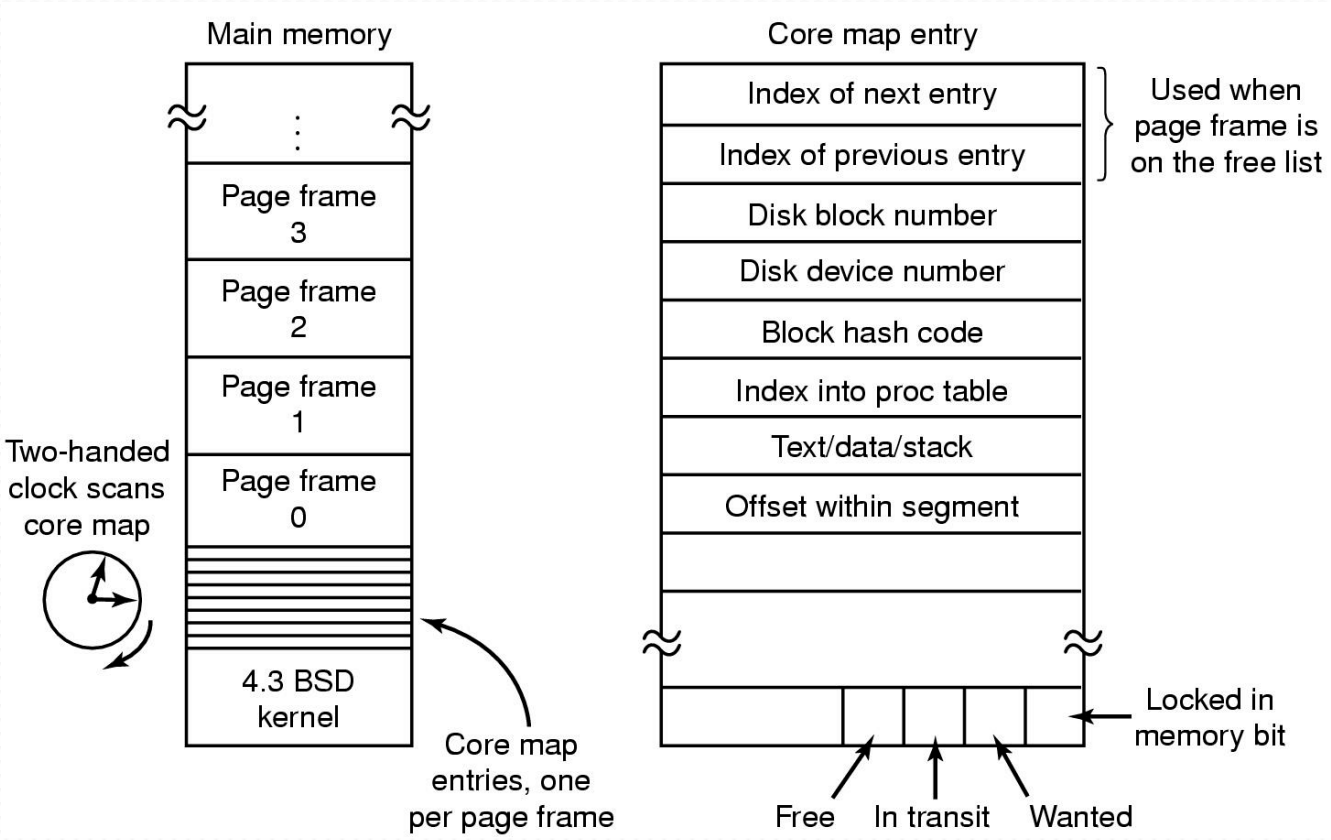
Archivo mapeado simultáneamente a dos procesos

Llamadas al sistema para la administración de memoria

System call	Description
s = brk(addr)	Change data segment size
a = mmap(addr, len, prot, flags, fd, offset)	Map a file in
s = unmap(addr, len)	Unmap a file

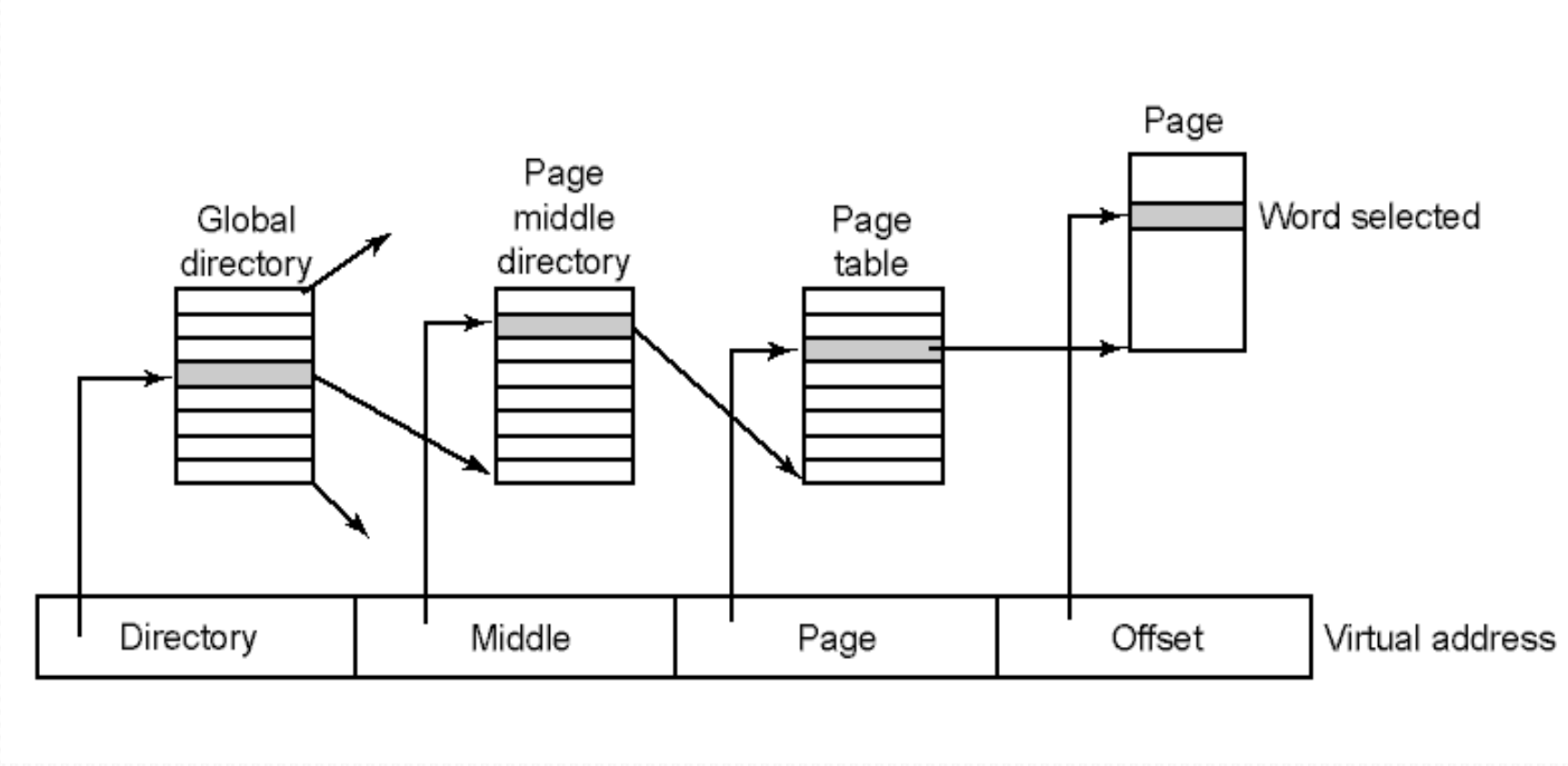
- **s** es un código de error (-1)
- **b** y **addr** son direcciones de memoria
- **len** es una longitud
- **prot** controla la protección
- **flags** bits indicadores
- **fd** es un descriptor de archivo
- **offset** es un desplazamiento

Paginación en UNIX



El mapa tiene una entrada por página

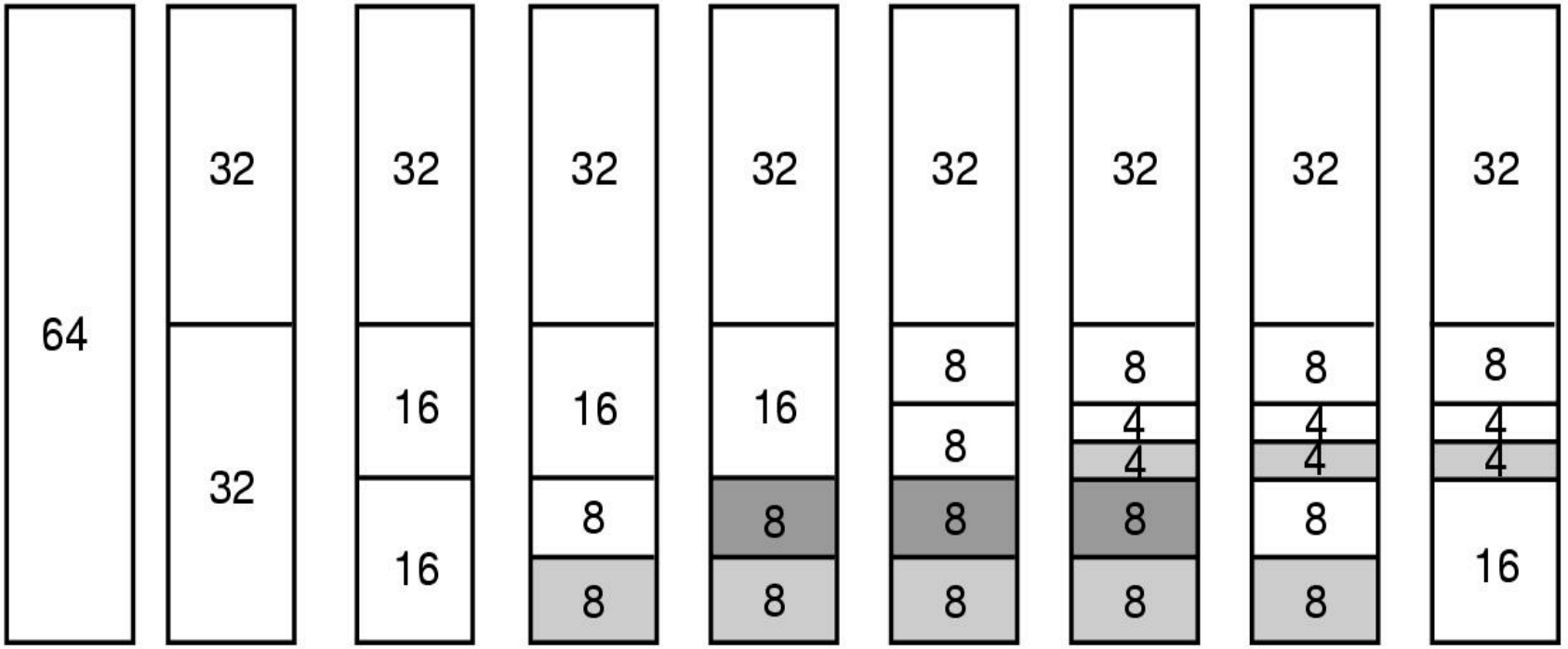
Paginación en Linux (1)



Linux usa tablas de páginas de tres niveles

Paginación en Linux (2)

Funcionamiento del algoritmo



Buddy algorithm

Resumiendo

- Nos concentraremos en sistemas POSIX
- Las llamadas al sistema permiten interactuar al programador con el sistema operativo
- Algunos elementos importantes:
  - Administracion de procesos e hilos
  - Señales
  - Administración de memoria