

Introducción a los sistemas operativos

Gustavo Romero

Arquitectura y Tecnología de Computadores

21 de octubre de 2010

Índice

- 1 Abstracciones
- 2 Llamadas al sistema

Lecturas recomendadas

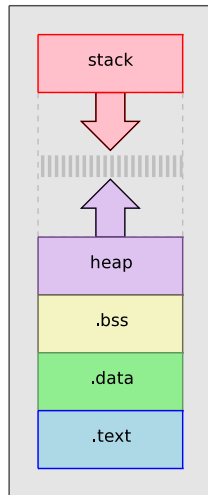
Tanuenbaum Sistemas Operativos Modernos (1)

Silberschatz Fundamentos de Sistemas Operativos (2)

Stallings Sistemas Operativos (2)

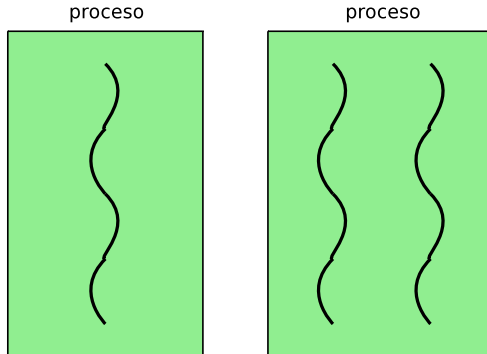
Espacio de direcciones

- **Memoria** utilizable por un proceso.
- Unidad de **protección** de memoria.
- Habitualmente formado por 3 componentes:
 - **código**: `.text`
 - **datos**: `.data`, `.bss` y `heap`
 - **pila**: `stack`
 - variables locales
 - marcos de procedimientos
- **Tamaño**:
 - código: tamaño fijo
 - datos: pueden crecer hacia **arriba**
 - pila: pueden crecer hacia **abajo**



Procesos y hebras

- (Porción de) un **programa** en ejecución.
- **Instancia** de un programa ejecutándose.
- **Hebra/hilo/fibra**: mínima unidad de **ejecución**.
- **Proceso/tarea**: unidad de posesión/protección de **recursos**.



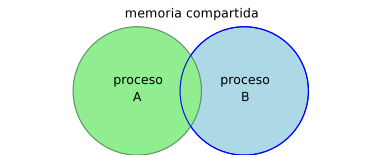
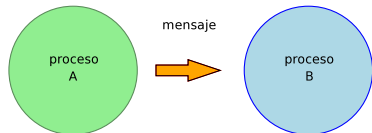
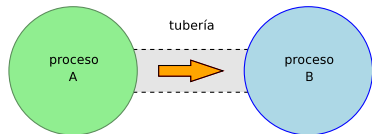
Comunicación entre procesos

- Tipos:

- **Ficheros:** tuberías, sockets.
- **Paso de mensajes.**
- **Memoria compartida.**

- Ejemplo:

- Comunicación de dos procesos a través de una tubería.
- `cat alumnos.txt | sort`
- Funcionamiento:
 - “A” deja de escribir cuando la tubería está llena.
 - “B” deja de leer cuando la tubería está vacía.



Concurrencia y paralelismo

- Varios **procesos** pueden ejecutarse...
 - en **paralelo** en un **multiprocesador**.
 - de forma **concurrente** en un **uniprocesador**.
- Los procesos **multihebra** pueden ser ejecutados de forma paralela o concurrente en función de...
 - el número de procesadores
 - el modelo de hebras: usuario o núcleo
- Podrán aparecer “**condiciones de carrera**” si no manejamos cuidadosamente la concurrencia $\implies$ necesitaremos métodos de **sincronización** de procesos y hebras.

Gestión de memoria

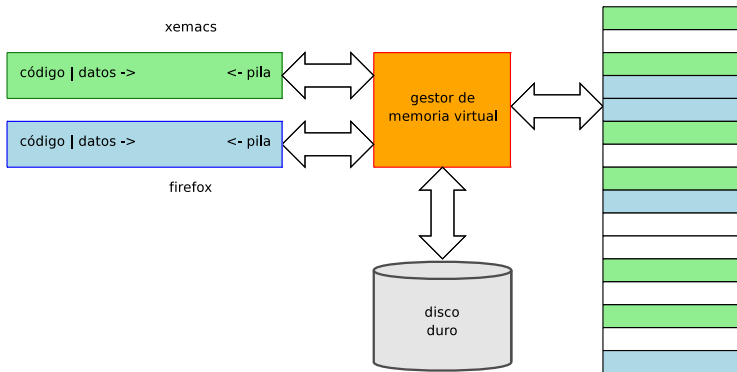
- La memoria RAM es **limitada**.
- Problema: los procesos no saben qué posición ocuparán en la RAM.
 - Responsabilidad del compilador y del SO.
 - Solución: $\implies$ **código relocizable**.
- Problema: las necesidades de memoria de todos los procesos activos pueden ser mayores que la memoria disponible.
 - Solución: $\implies$ **memoria virtual**.

Memoria virtual

- Permite al programador direccionar memoria de forma razonable:
 - Hace **crear** a los procesos que disponen de **toda la RAM** para ellos.
 - Los espacios de direcciones son **independientes** unos de otros:
 - una misma dirección virtual en dos espacios de direcciones puede ser mapeada en diferentes posiciones en la memoria principal.
 - El **mapeado** de porciones de memoria virtual en memoria física se hace de forma **automática** $\Rightarrow$ liberación para el programador.
- El funcionamiento **eficiente** de la memoria virtual necesita soporte **hardware**.

Memoria virtual

- Las aplicaciones creen tener un espacio de direcciones **plano**.
- La memoria física se **divide** en porciones.
- Las regiones **no** necesitan mapearse de forma **contigua**.
 - excepción: E/S mapeada en memoria (controladores).



Planificación/gestión de recursos

- Equidad:
 - Proporcionar cantidades iguales de recurso a todos los usuarios.
- Tiempo de respuesta:
 - Discriminar entre clases de trabajos: tiempo real, interactivos, por lotes...
- Eficiencia:
 - Maximizar el rendimiento.
 - Minimizar el tiempo de respuesta.
 - Permitir tantos usuarios simultáneos como sea posible.
- Diferencia entre **políticas** y **mecanismos**:

planificación	↔	despacho
paginación	↔	reemplazo
interacción	↔	comunicación

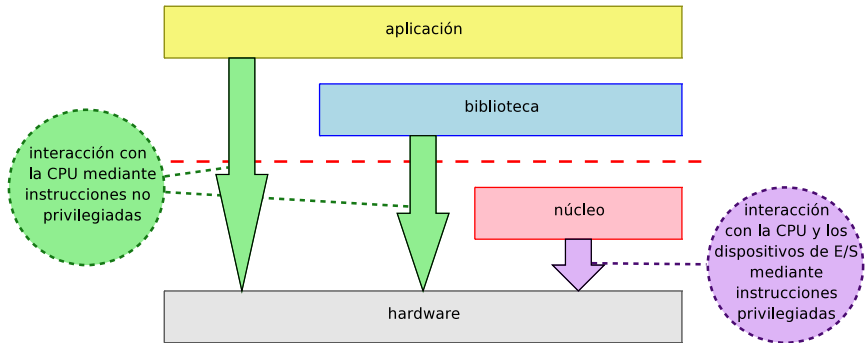
Gestión de E/S

- **Clasificación** de dispositivos de E/S:
 - Dispositivos de **caracteres**: puerto serie, módem, ratón,...
 - Dispositivos de **bloques**: discos duros, tarjetas de red,...
- **Papel** de la gestión de dispositivos $\Rightarrow$ **interfaz**:
 - Proporcionar una interfaz genérica... como en UNIX.
 - Proporcionar una interfaz específica para cada tipo de dispositivo... como en Windows.
- **Componentes** de un controlador de dispositivo (software):
 - Código de inicialización.
 - Llamada al sistema para responder a las peticiones de usuario.
 - Manejador de interrupción para responder a las peticiones del controlador de dispositivo (hardware).

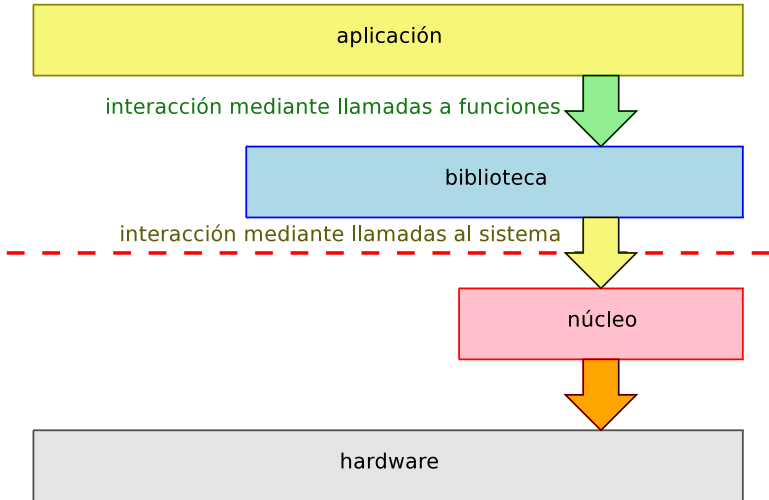
Ficheros

- Implementación del **almacenamiento persistente** o a largo plazo.
- Las unidades almacenadas persistentemente son objetos como...
 - **Ficheros.**
 - **Directorios.**
- Tipos de ficheros:
 - Tradicionales: gestionados a través de llamadas al sistema.
 - **Mapeados en memoria:** gestionados como el resto de la memoria principal.

Interacción de los componentes del sistema



Interacción de los componentes del sistema



Interfaz de programación de aplicaciones (API)

- Existen dos interfaces de programación para acceder a los servicios proporcionados por el núcleo del SO:
 - **Llamadas al sistema**: interfaz directa con el núcleo.
 - **API**: interfaz indirecta (biblioteca) normalmente escrita en algún lenguaje de alto nivel como C o C++.
- Las 3 APIs más conocidas son:
 - **Win32**.
 - **POSIX**: **P**ortable **O**perating **S**ystem **I**nterface (UNIX).
 - **Java**.

API: ejemplo de uso de POSIX en Linux

```
.data
msg:      .string "cognito , ergo sum\n"

.text
.globl _start
_start:
    movl   $4,%eax    # write
    movl   $1,%ebx    # salida estandar
    movl   $msg,%ecx   # cadena
    movl   $18,%edx   # longitud
    int    $0x80      # llamada al sistema
    movl   $1,%eax    # exit
    xorl   %ebx,%ebx  # 0
    int    $0x80      # llamada al sistema
```

```
#include <iostream>
```

```
using namespace std;
```

```
int main()
{
    cout << "cognito , ergo sum" << endl;

    return 0;
}
```

Paso de parámetros

- A menudo es necesaria más **información** que la identificación de la llamada al sistema
 - El tipo y cantidad de información varía entre llamadas y SOs.
- Las tres formas habituales de paso de parámetros son:
 - **Registros:** método rápido pero de poca capacidad.
 - **Memoria:** la aplicación agrupa los parámetros en un área de memoria y pasa al SO la localización de dicha área de memoria mediante un registro.
 - **Pila:** los parámetros son apilados por la aplicación y desapilados por el SO.
- El empleo de memoria tiene la ventaja de no limitar el número ni el tamaño de los parámetros, pero es más lenta que los registros (velocidad+copia).

Gestión de procesos

llamada	descripción
<code>pid = fork()</code>	crea un proceso hijo
<code>waitpid(pid, estado, opciones)</code>	espera la finalización de un proceso hijo
<code>estado = execve(fichero, argumentos, entorno)</code>	reemplaza el núcleo de un proceso
<code>exit(estado)</code>	finaliza un proceso y devuelve un valor de estado

Gestión de ficheros

llamada	descripción
descriptor = open (nombre, modo)	abre un fichero
estado = close (descriptor)	cierra un fichero
cantidad = read (descriptor, bufer, bytes)	lee datos de un fichero
cantidad = write (descriptor, bufer, bytes)	escribe datos en un fichero
posicion = lseek (descriptor, desplazamiento, desde)	mueve el puntero del fichero
estado = stat (nombre, &búfer)	obtener información de estado

Gestión de directorios

llamada	descripción
estado = mkdir (nombre, modo)	crea un nuevo directorio
estado = rmdir (nombre)	borra un directorio vacío
estado = link (nombre1, nombre2)	crea un enlace
estado = unlink (nombre)	borra una entrada de directorio
estado = mount (origen, destino, tipo, opciones)	monta un sistema de ficheros
estado = unmount (destino)	desmonta un sistema de ficheros

Otras llamadas al sistema

llamada	descripción
estado = chdir (nombre)	cambia el directorio de trabajo
estado = chmod (nombre, modo)	cambia los bits de protección
estado = kill (pid, señal)	envía una señal a un proceso
segundos = time (&segundos)	nº de segundos desde 1/1/1970

Comparativa POSIX/Win32

POSIX	Win32	descripción
chdir	SetCurrentDirectory	cambia el directorio actual
chmod		cambia los permisos
close	CloseHandle	cierra un fichero
execve	CreateProcess	cambia la imagen de un proceso
exit	ExitProcess	finaliza la ejecución de un proceso
fork	CreateProcess	crea un nuevo proceso
kill		envía una señal
link		crea un enlace
lseek	SetFilePointer	mueve el puntero de fichero
mount		monta un sistema de ficheros
mkdir	CreateDirectory	crea un directorio
open	OpenFile	crea/abre un fichero
read	ReadFile	lee de un fichero
rmdir	RemoveDirectory	borra un directorio
stat	GetFileAttributesEx	obtener atributos de un fichero
time	GetLocalTime	obtiene la hora
unlink	DeleteFile	borra un fichero
unmount		desmonta un sistema de ficheros
waitpid	WaitForSingleObject	espera la finalización de un proceso
write	WriteFile	escribe en un fichero