Sistemas Operativos I

“A computer is a state machine. Threads are for people who can't program state machines.”

Alan Cox

2022

Rafael Ignacio Zurita <rafa@fi.uncoma.edu.ar>

Advertencia: Estos slides traen ejemplos.

No copiar (ctrl+c) y pegar en un shell o terminal los comandos aquí presentes.

Algunos no funcionarán, porque al copiar y pegar tambien van caracteres “ocultos” (no visibles pero que están en el pdf) que luego interfieren en el shell.

Sucedió en vivo :)

Conviene “escribirlos” manualmente al trabajar.

Sistemas Operativos I - Procesos y Threads

Contenido:

- Procesos
- Comunicación interprocesos
- Threads
- Planificación de procesos
- Sincronización de procesos
- Deadlocks

%20

Sistemas Operativos I - Procesos y Threads

Procesos:

- Creación de procesos
- Contexto
- Cambio de contexto
- Finalización de procesos
- Threads

Sistemas Operativos I - Procesos y Threads

Procesos:

El kernel tiene la capacidad de poner en ejecución a los programas que se encuentran almacenados en el sistema.

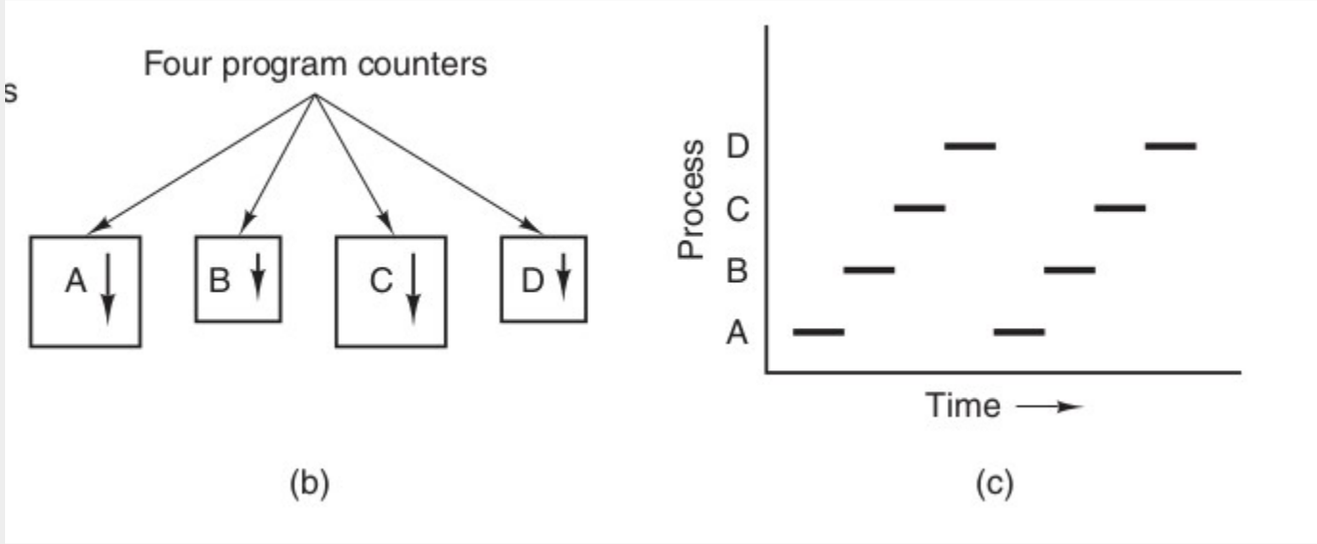
Cuando un programa está en ejecución, lo llamamos un proceso.

El sistema operativo controla la creación, ejecución y finalización de los procesos

Sistemas Operativos I - Procesos y Threads

El modelo de procesos

- procesos secuenciales
- multiprogramación



Sistemas Operativos I - Procesos y Threads

Creación de procesos:

El sistema operativo obtiene una porción de memoria para el proceso (segmentos de memoria de un proceso)
Tabla de datos administrativos para el proceso
Asignar un PID
Colocar al proceso en estado de listo o suspendido

Sistemas Operativos I - Procesos y Threads

Creación de procesos (cuando):

- En la secuencia de inicio del sistema
- Cuando una aplicación en ejecución ejecuta un system call para crear un proceso
- Cuando un usuario solicita ejecutar un programa (ej: en el shell)

Sistemas Operativos I - Procesos y Threads

Creación de procesos (code):

```
/* Creación de proceso en LINUX */


#include  <stdio.h>
#include  <unistd.h>
#include  <sys/types.h>


void  main(void)
{
    int  pid;
    int x = 0;

    pid = fork();

    if (pid == 0)
      printf("Proceso hijo %d\n", x++);
    else
      printf("Proceso padre. Mi hijo es el pid=%d \n", pid);
}


/* otras funciones de la biblioteca de C
 * (que realizan llamadas al sistema)
 *
 * wait()
 * exit()
 * execv()
 * getpid()
 */
```

```
/* Creación de proceso en XINU */

#include <xinu.h>

void    sndA(void);

/*------------------------------------------------------------------------
 * main  --  example of creating processes in Xinu
 *------------------------------------------------------------------------
 */
void    main(void)
{
        int pid;

           pid = create(sndA, 128, 20, "process 1", 0) );
        resume(pid);

}

/*------------------------------------------------------------------------
 * sndA  --  repeatedly emit 'A' on the console without terminating
 *------------------------------------------------------------------------
 */
void    sndA(void)
{
        while( 1 )
              putc(CONSOLE, 'A');
}
```

Sistemas Operativos I - Procesos y Threads

En sistemas de tipo UNIX

- Sistema jerárquico de procesos (árbol)
- El proceso padre puede esperar al hijo
- El proceso hijo puede reemplazar sus segmentos con el de un nuevo programa

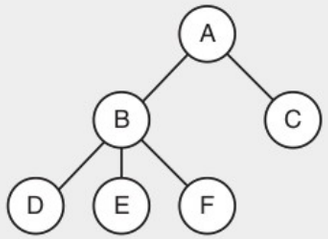


Figure 1-13. A process tree. Process *A* created two child processes, *B* and *C*. Process *B* created three child processes, *D*, *E*, and *F*.

En Linux, comandos útiles: ps, pstree, top, kill, killall

Sistemas Operativos I - Procesos y Threads

Finalización de procesos

- Finalización normal (voluntario).
- Salida con Error (voluntario).
- Error detectado por el OS (involuntario).
- Finalizado por otro proceso (ej: kill, involuntario)

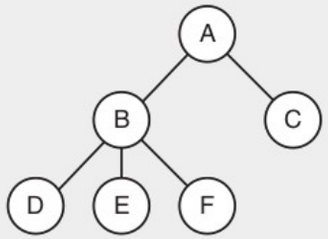


Figure 1-13. A process tree. Process *A* created two child processes, *B* and *C*. Process *B* created three child processes, *D*, *E*, and *F*.

Sistemas Operativos I - Procesos y Threads

Implementación de procesos

Que mantener? :

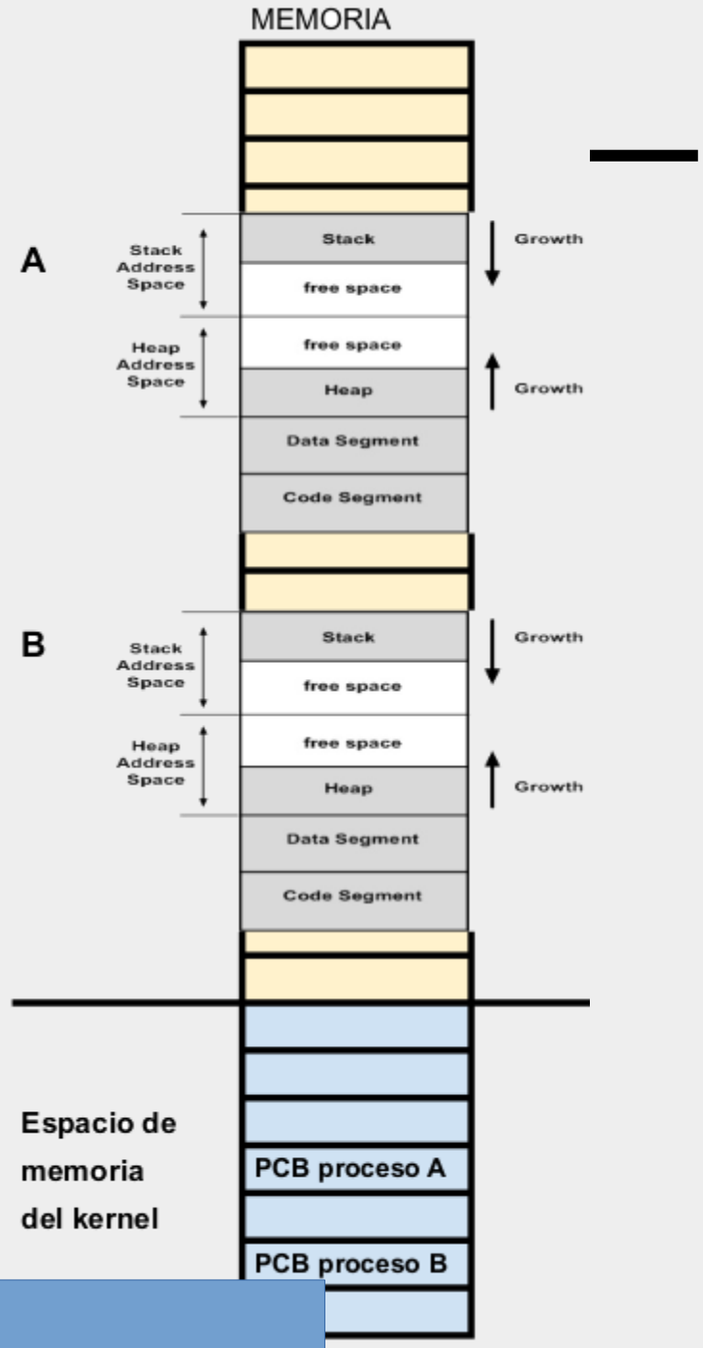
espacio de direcciones, registros del estado del proceso, lista de archivos abiertos, semáforos que espera, etc

Implementación

El kernel mantiene un Arreglo de estructuras. Tabla de procesos (PCB)

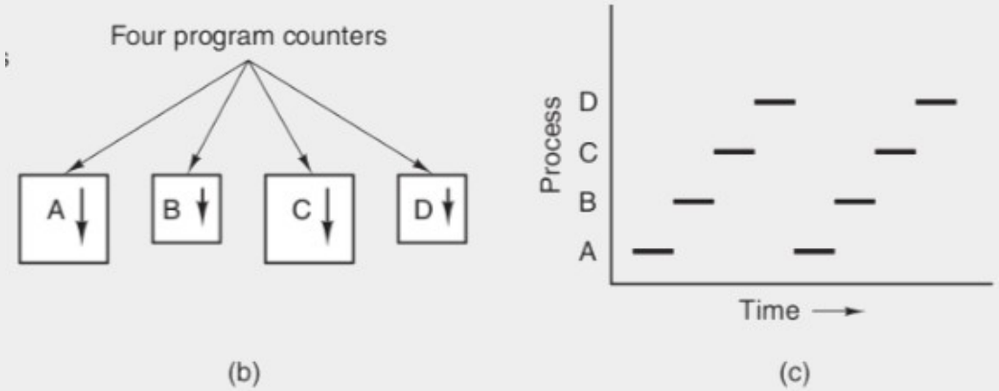
Cada PCB Contiene:

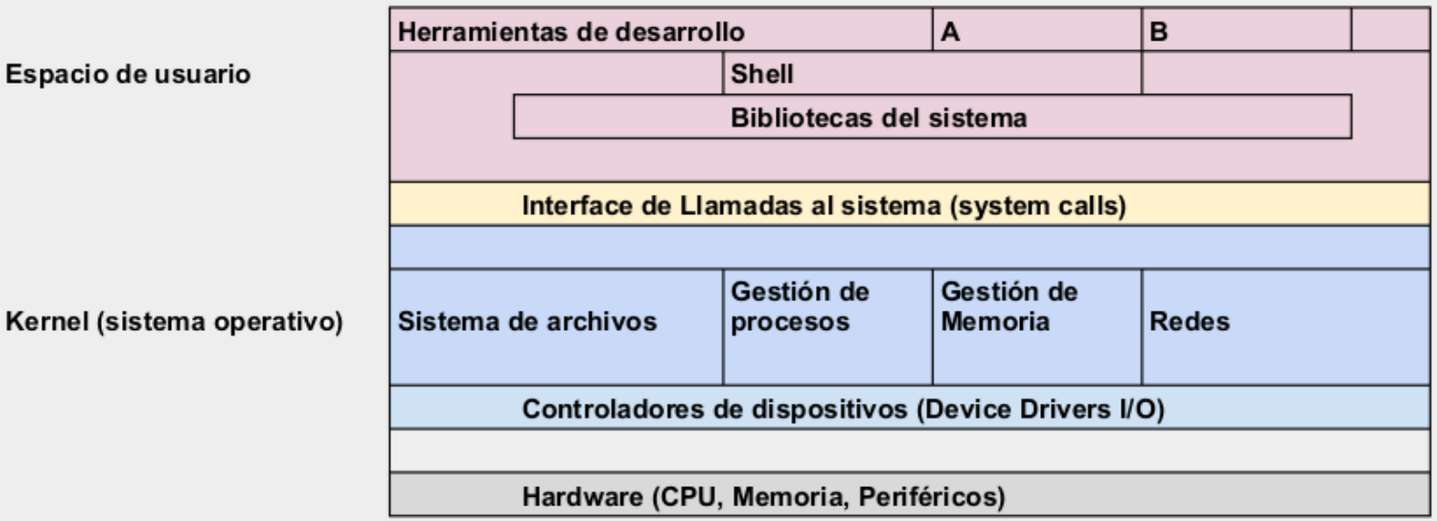
- PID
- Espacio para resguardar el contenido de los Registros de la CPU
  (pc, stack pointer, otros registros)
- datos sobre:
  - El espacio de direcciones de memoria del proceso
  - Archivos abiertos
  - Recursos en uso (semáforos, dispositivos E/S, etc)

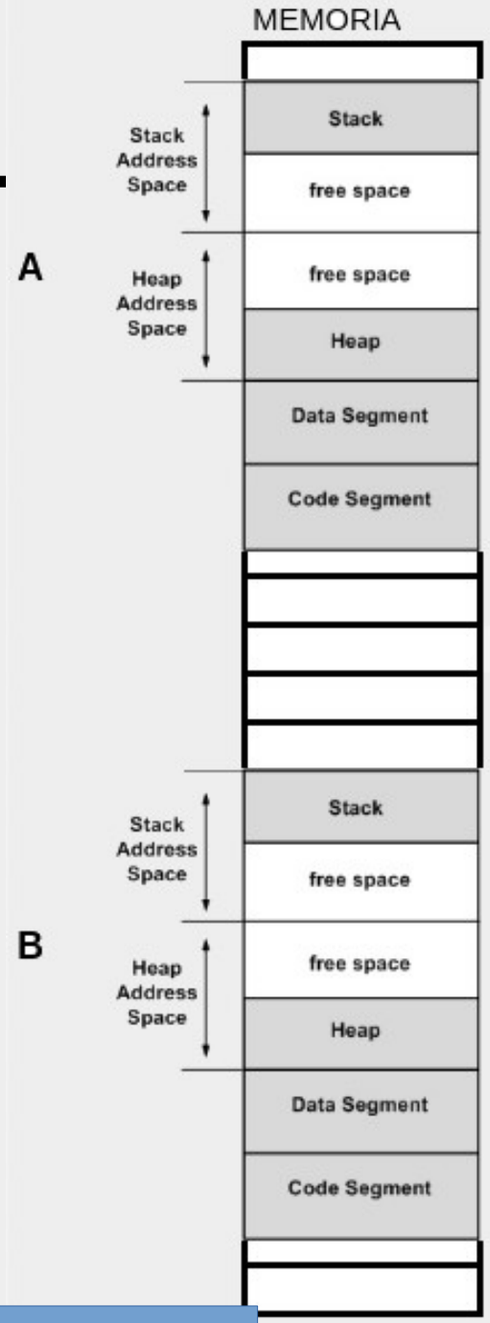


Sistemas Operativos I - Procesos y Threads

Ejecución concurrente - cambio de contexto







Sistemas Operativos I - Procesos y Threads

Implementación de procesos concurrentes

Cambio de contexto

Arreglo de estructuras.
Tabla de procesos (PCB)

Resguardar el estado del procesador
para el proceso A

Cargar el estado anterior del procesador
para el proceso B

Estado del procesador:
Registros (pc, stack pointer, otros registros)

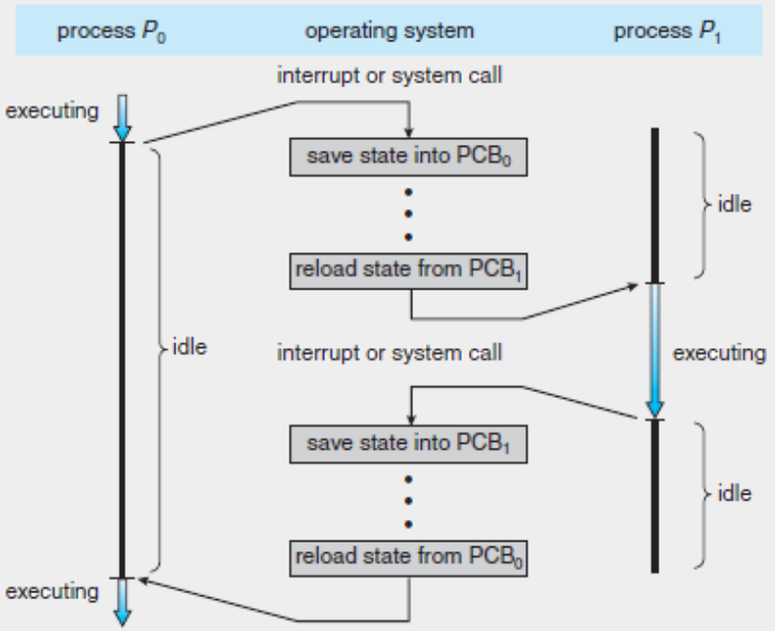
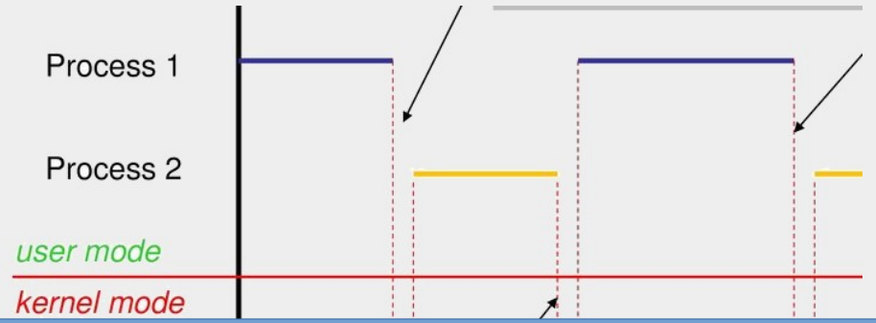
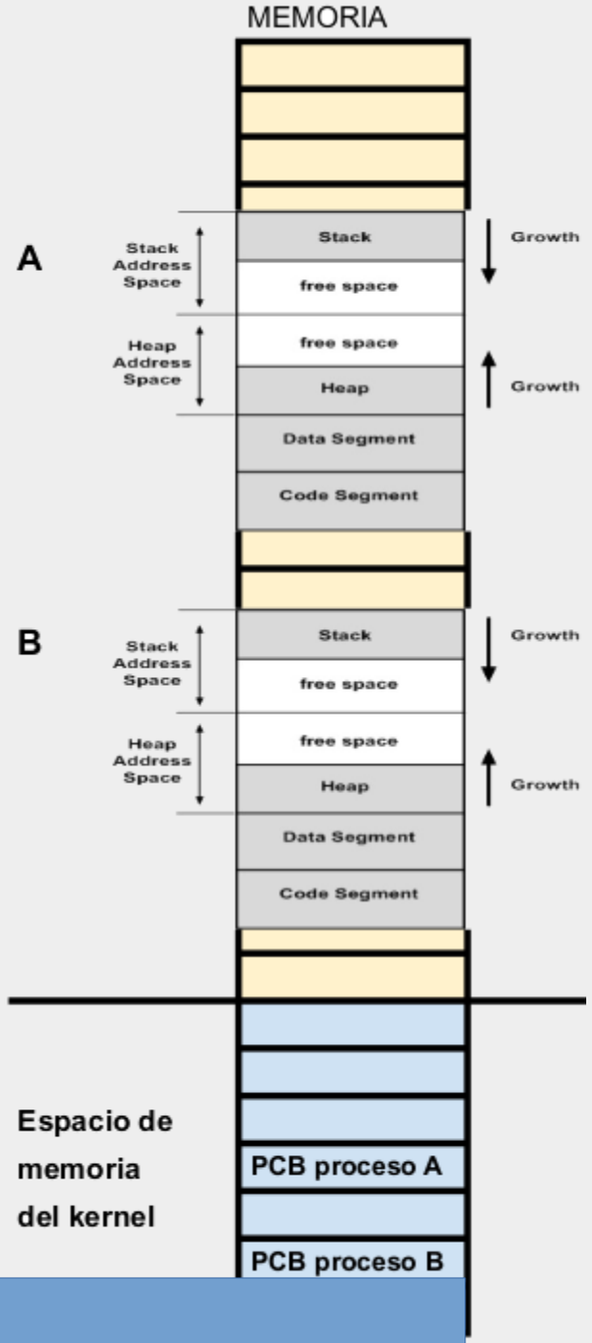


Figure 3.4 Diagram showing CPU switch from process to process.





CONTINUARÁ...