Sistemas Operativos I

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

Alan Cox

Clase: procesos

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

Contenido

- Procesos
- Creación y Finalización
- Introducción a estados
- Cambio de contexto
- Ejemplos
- Introducción a XINU

Sistemas Operativos I - Procesos

Concepto de Procesos

- Creación de procesos
- Jerarquía de procesos
- Finalización de procesos
- Ejemplos de comandos del sistema y llamadas al sistema

- Implementación de procesos
- Cambio de contexto
- Uso de Xinu

Procesos

Conceptos a relacionar luego de finalizada la clase

multiprogramación y tiempo compartido

mapa de memoria de un proceso (segmentos)

memoria general del sistema (kernel y procesos)

creación y finalización de procesos

PCB - tabla de procesos

quantum - reloj/timer - interrupciones

cambio de contexto

system calls

Concepto de Proceso

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.

Concepto de Proceso

1. Es una abstracción (implementada)

2. Operaciones o primitivas para gestionar 1. (system calls)

Creación de procesos (cuando)

- En la secuencia de inicio del sistema
- Cuando una aplicación realiza
  un system call para crear un proceso
- Cuando un usuario solicita ejecutar un programa
  (ej: en el shell o con “doble click”)

Tareas del SO al crear un proceso

- El sistema operativo obtiene una porción de memoria para el proceso (segmentos de memoria)
- Crear una estructura de datos para administrar el proceso (PCB)
- Asignar un process id (PID)
- Colocar al proceso en estado de nuevo, listo o suspendido

Sistemas Operativos I - Procesos

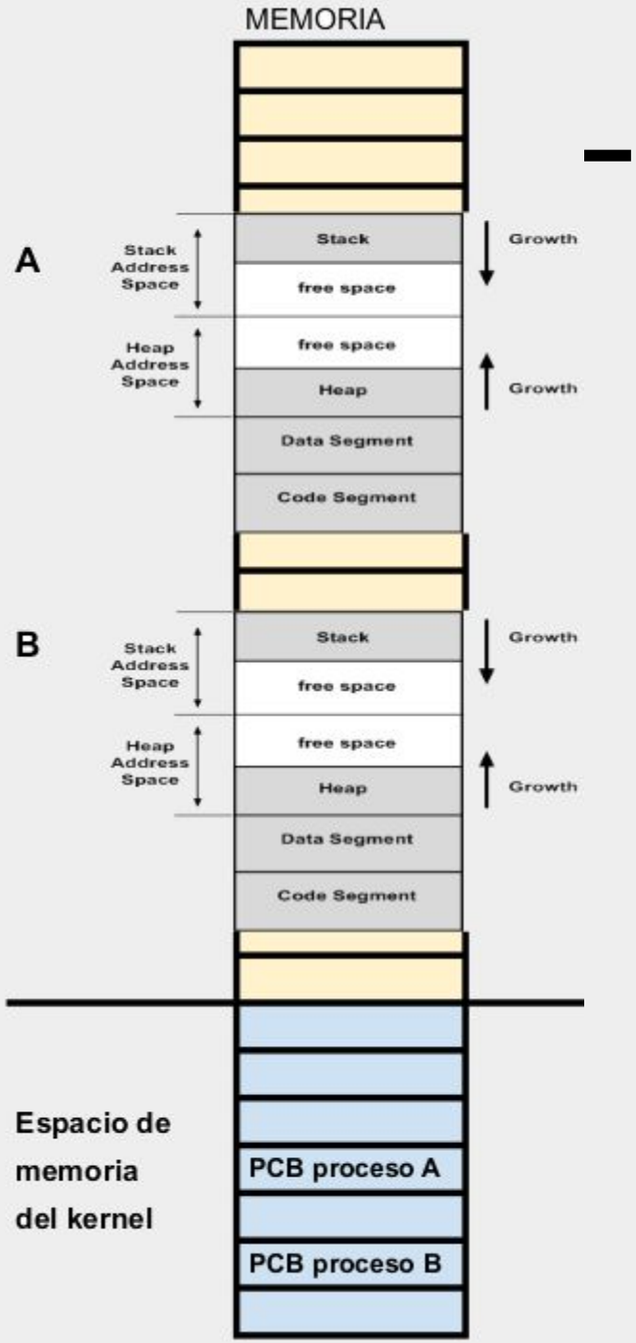
Implementación de procesos

¿Qué mantener?

El kernel mantiene un Arreglo/Lista de Estructuras, donde cada elemento es una Tabla o Bloque de Control proceso (PCB)

Cada PCB Contiene

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

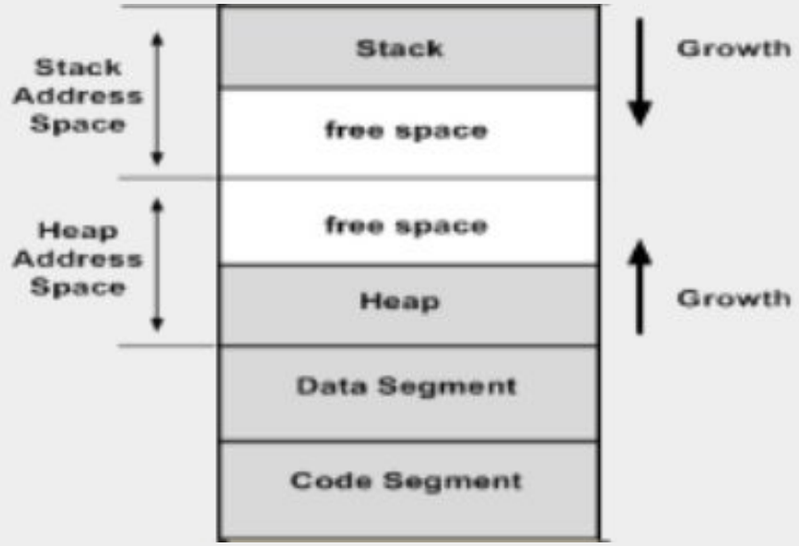


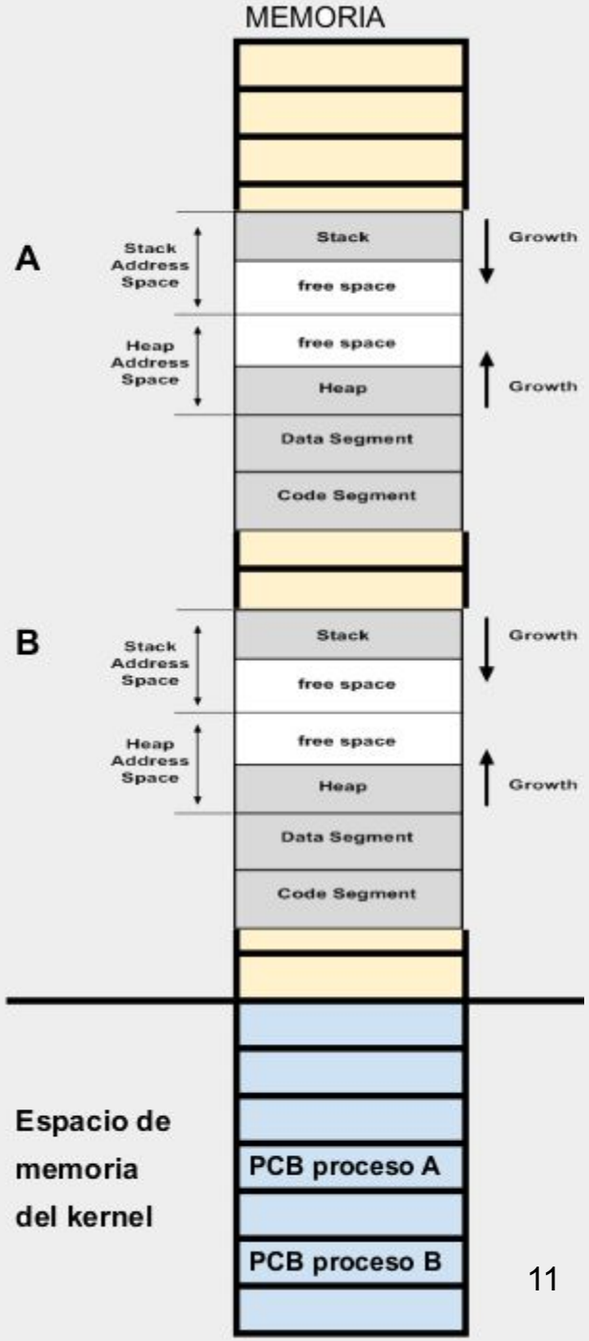
Implementación de procesos

mapa de memoria de un proceso (segmentos)

memoria general del sistema (kernel y procesos)

```
char edad = 45;
int DNI = 34563112;
char nota[] = "Isidoro Caniones";
char encrip[] = "Los Elefantes de Asia";

main () {
        int i = 0;
        int c = 0;

        for (i=0; i<60; i++)
          c = add_elem(i);
}

int add_elem(int n)
{
    int val;

    val = nota[n] + encrip[n];
    return val;
}
```





Estados de un proceso

```c
/* Creación de proceso en UNIX (LINUX, MAC OS) */

#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", getpid());
     else
        printf("Proceso padre. Mi hijo es pid=%d \n", pid);
}
```

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

Creación de procesos (code)

```c
/* Creación de proceso en UNIX (LINUX, MAC OS) */

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

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

     pid = fork();

     if (pid == 0)
        /* aquí se puede utilizar la función execv()
         * para ejecutar otro programa
         */
     else
        printf("Proceso padre. Mi hijo es pid=%d \n", pid);
}
```

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

Creación de procesos (code)

```
/* Creación de proceso en XINU */

#include <xinu.h>
void    sndA(void);

void    main(void)
{
    int pid;

    pid = create(sndA, 8192, 20, "process 1", 0) );
    resume(pid);

    sleep(5);
    kill(pid);
}

/* proceso sndA */
void    sndA(void)
{
        while( 1 )
        putc(CONSOLE, 'A');
}
```

Creación de procesos (code)

Sistemas Operativos I - Procesos

Jerarquía de procesos : en sistemas de tipo UNIX

- Sistema **jerárquico de procesos** (árbol).
- El proceso padre puede esperar al hijo.
- Implementan finalización de procesos en cascada

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

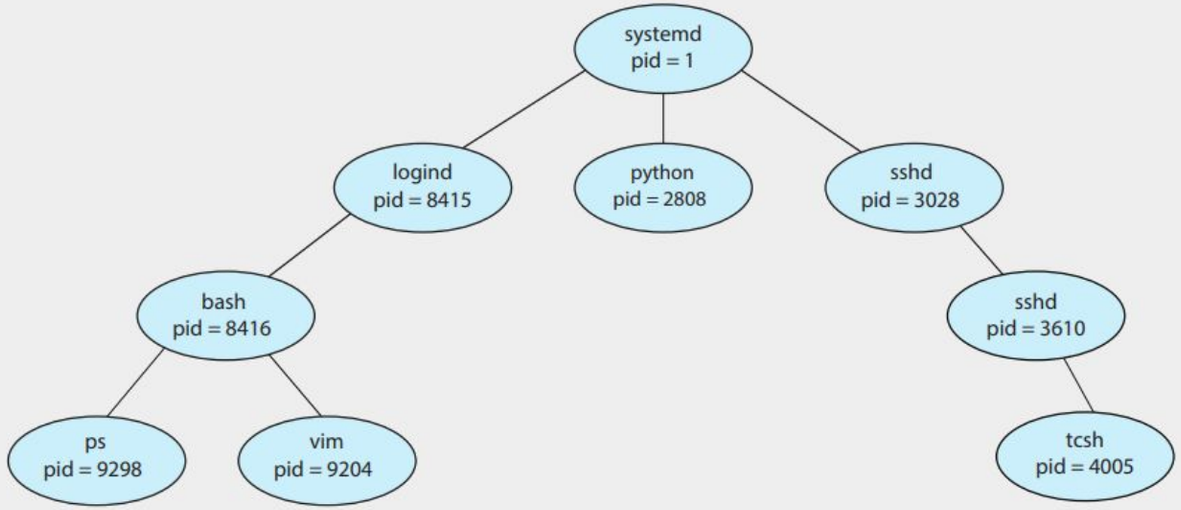


Figure 3.7 A tree of processes on a typical Linux system.

Finalización de procesos (cuando)

- **Finalización voluntaria (normal o con error)**
  exit()
- **Finalización involuntaria (instruccion ilegal, división por cero, etc)**
  - **Error detectado por el OS**
  - **Finalizado por otro proceso**
    kill()

```c
/* Creación y finalización de proceso en Linux */

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


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

     pid = fork();

     if (pid == 0)
        for(;;)
                printf("Proceso hijo %d\n", x++);

     else {
        sleep(5);
        kill(pid, SIGKILL);
        printf("Maté a mi propio hijo (suena horrible)\n");

     }
}
```

```c
/* Creación y finalización de proceso en XINU */

#include <xinu.h>

void    sndA(void);

void    main(void)
{
     int pid;

     pid = create(sndA, 2048, 20, "process 1", 0) );
     resume(pid);
     sleep(5);
     kill(pid);
     printf("Maté a mi propio hijo (suena horrible)\n");
}


/* proceso sndA */
void    sndA(void)
{
        while( 1 )
                putc(CONSOLE, 'A');
}
```

Finalización de procesos (code)

Multiprogramación apropiativa (tiempo compartido)

- Slice o QUANTUM
- Soporte del hardware: timer
- Cambio de contexto