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

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.

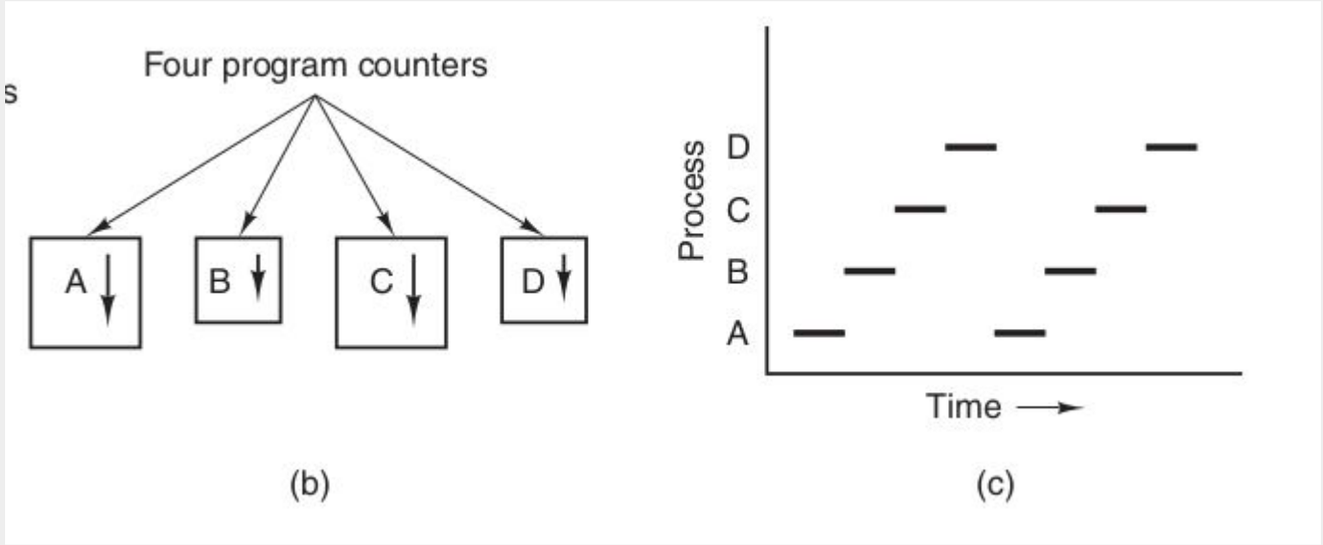
Cuando un programa está en ejecución, lo llamamos un proceso.

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

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):

```c
/* 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()
 */
```

```c
/* 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

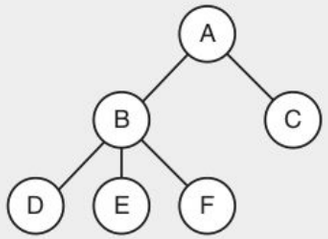


Figure 1-13. A process tree. Process *A* created two child processes, *B* and *C*. Process *B* created three child processes, *D*, *E*, and *F*.

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)

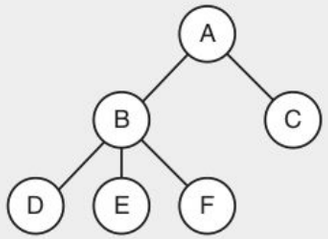


Figure 1-13. A process tree. Process *A* created two child processes, *B* and *C*. Process *B* created three child processes, *D*, *E*, and *F*.

Sistemas Operativos I - Procesos y Threads

Implementación de procesos

Que mantener? :

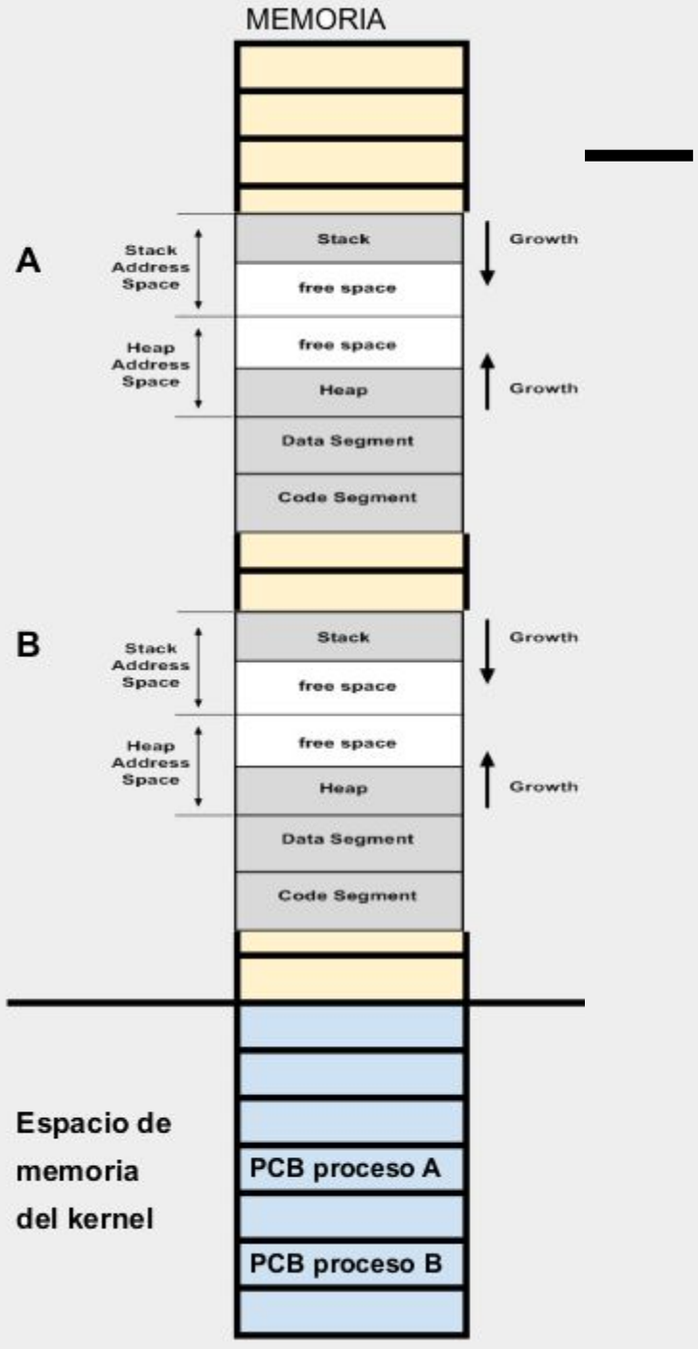
espacio de direcciones, registros del estado del proceso, lista de archivos abiertos, semáforos que espera, etc

Implementación

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

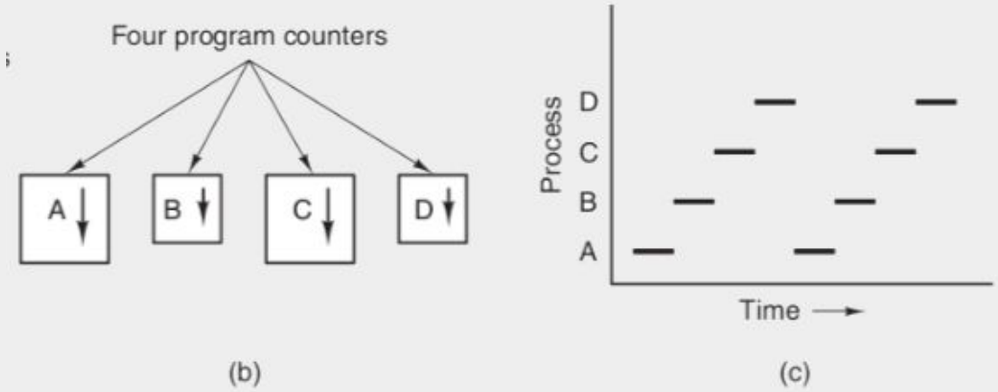
Cada PCB Contiene:

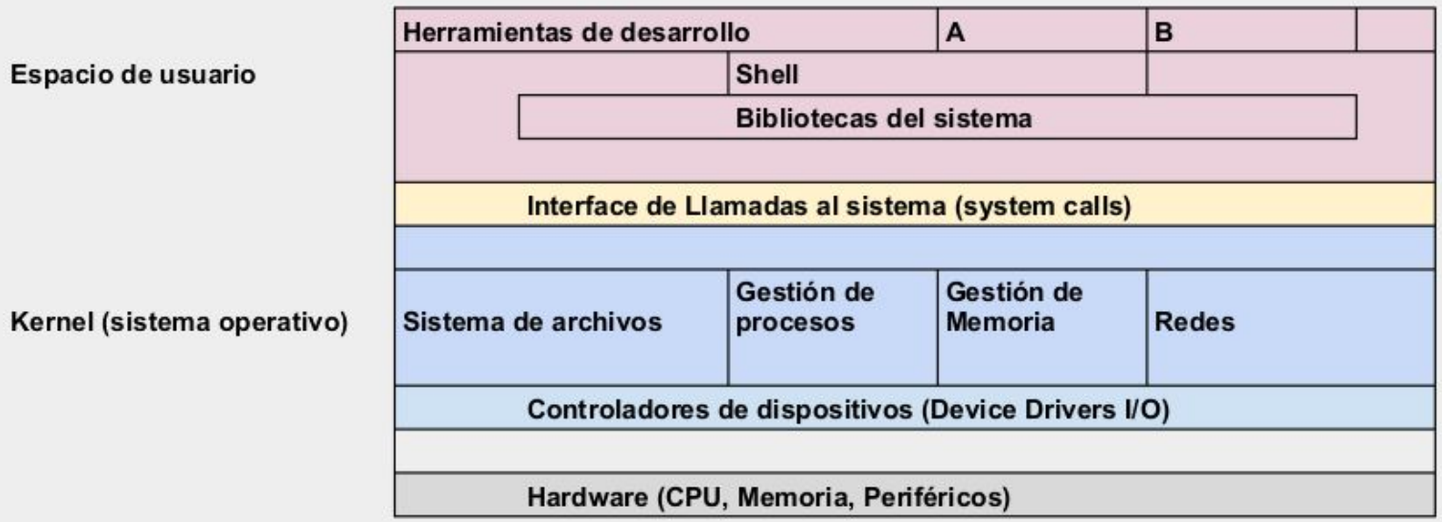
- PID
- Espacio para resguardar el contenido de los Registros de la CPU
  (pc, stack pointer, otros registros)
- datos sobre:
  - El espacio de direcciones de memoria del proceso
  - Archivos abiertos
  - Recursos en uso (semáforos, dispositivos E/S, etc)

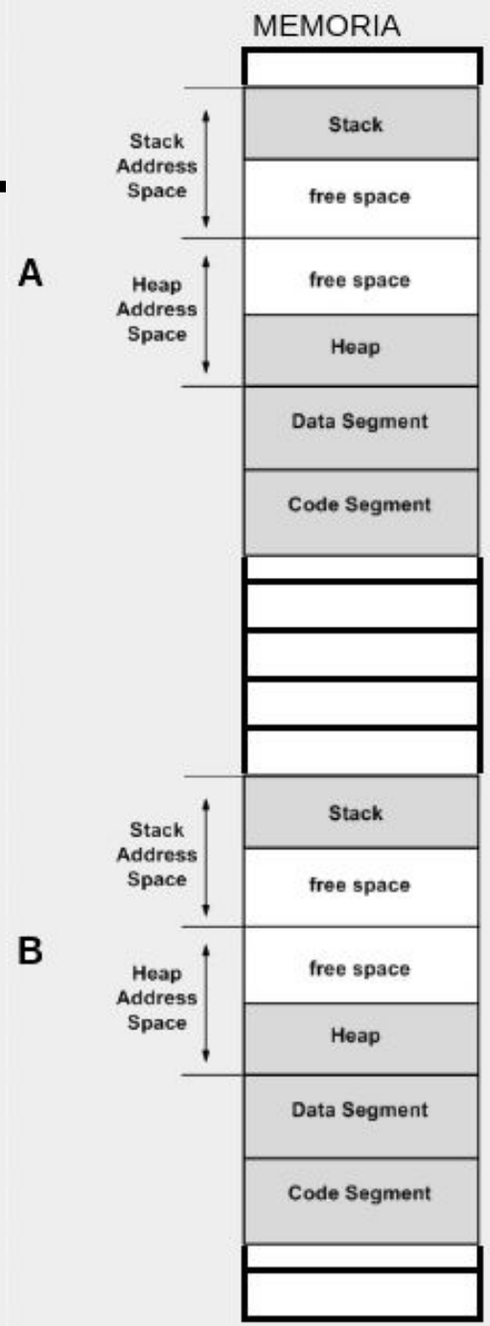


Sistemas Operativos I - Procesos y Threads

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)

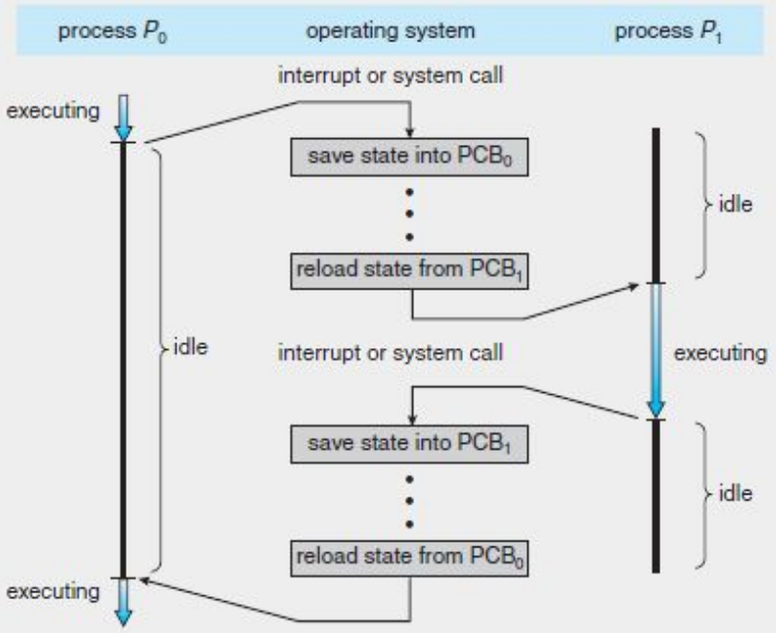
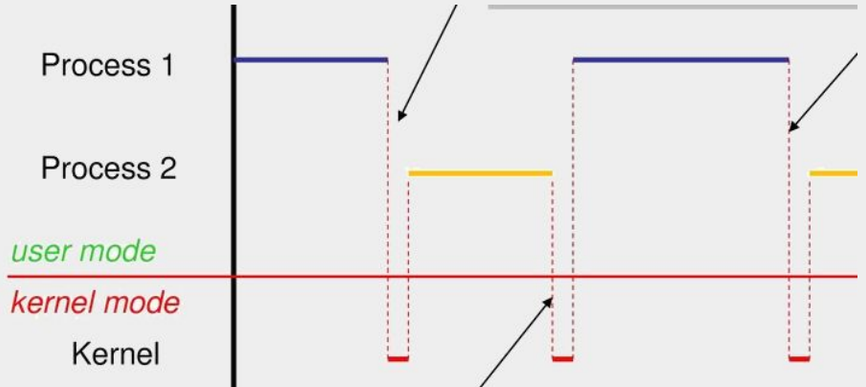
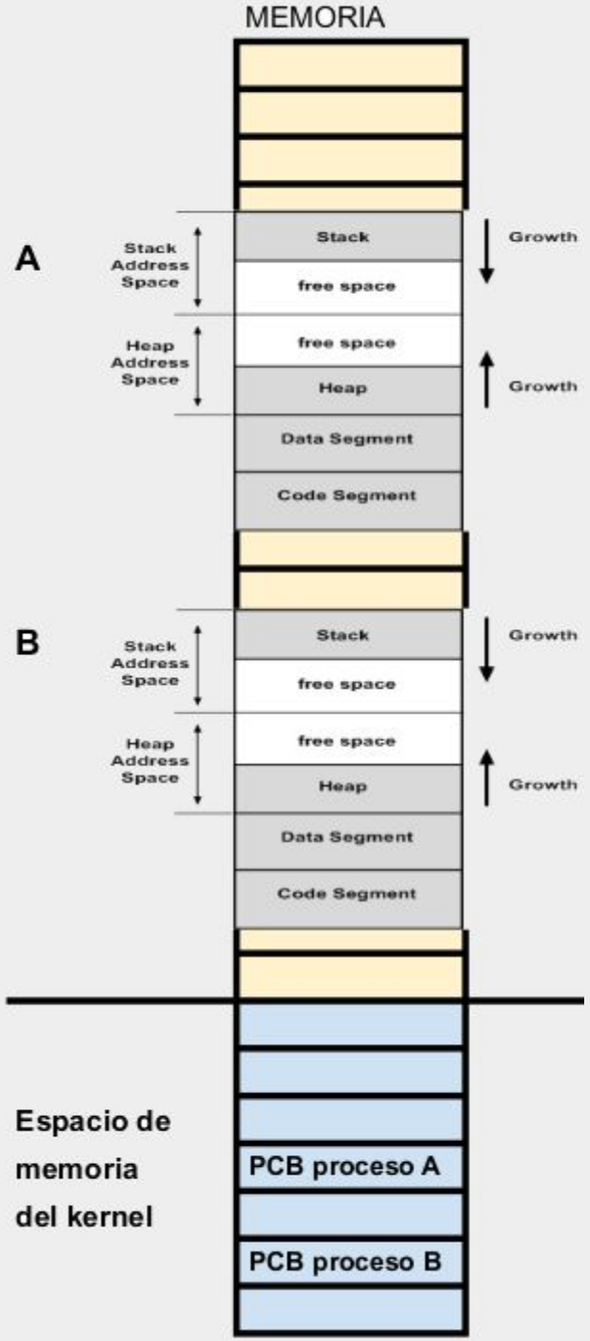


Figure 3.4 Diagram showing CPU switch from process to process.





Sistemas Operativos I - Procesos y Threads

5 de abril de 2022

Temario:

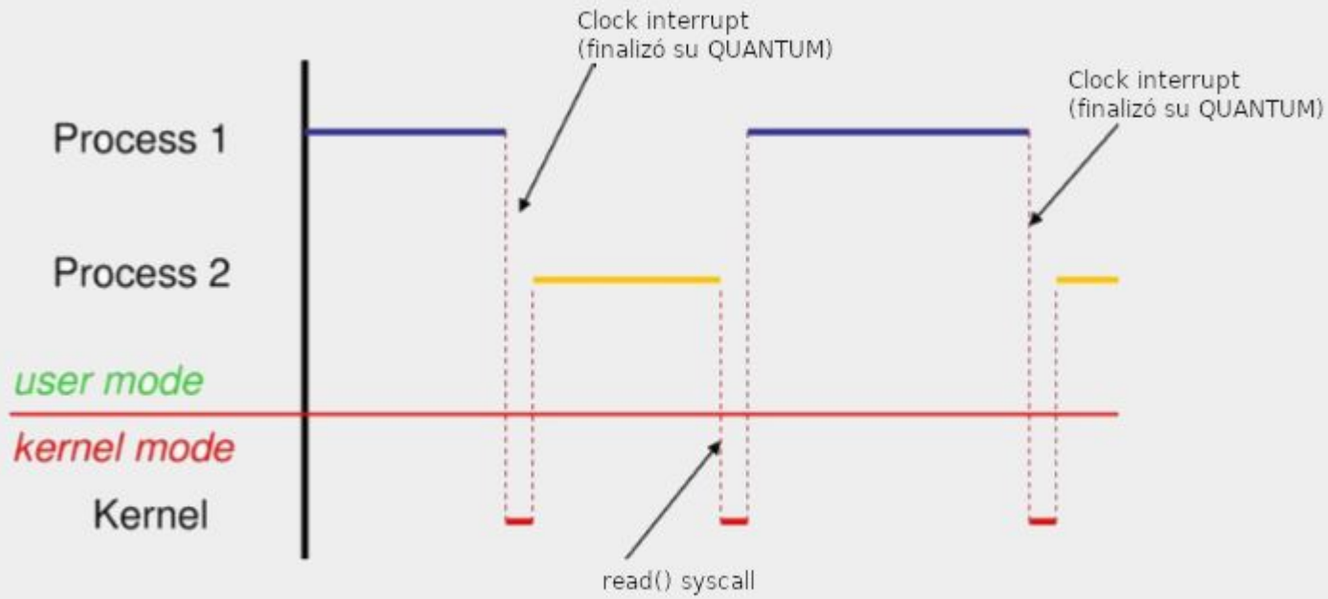
- Kernel monolítico vs microkernel
- Quantum y multiprogramación
- Threads
- Estados de un proceso en Xinu

Sistemas Operativos I - Procesos y Threads

Implementación de procesos concurrentes

Un proceso “libera” el uso de CPU cuando:

1. solicita un servicio al SO
2. finalizó su QUANTUM



1. multiprogramación
1. y 2. multiprogramación y tiempo compartido
(requiere reloj por hardware [clock/timer]) - interrupción

Los SO de tiempo compartido son “apropiativos” (preemptive)

Sistemas Operativos I - Procesos y Threads

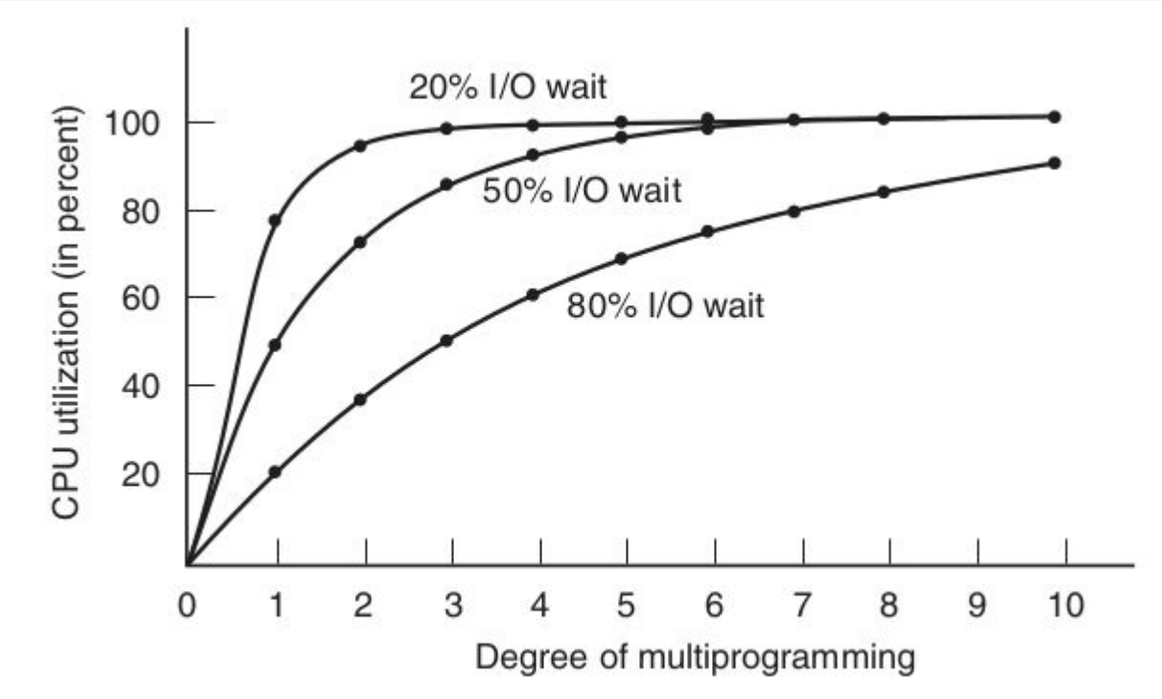
Grado de multiprogramación - Modelo sencillo

Utilización de CPU = $1 - p^n$

p: fracción de tiempo de un proceso esperando E/S
n: cantidad de procesos

Ejemplo: Sistema con 8GB de RAM.
2GB para el SO
2GB para cada proceso.
80% del tiempo esperando E/S.

¿Conviene tener mas RAM?



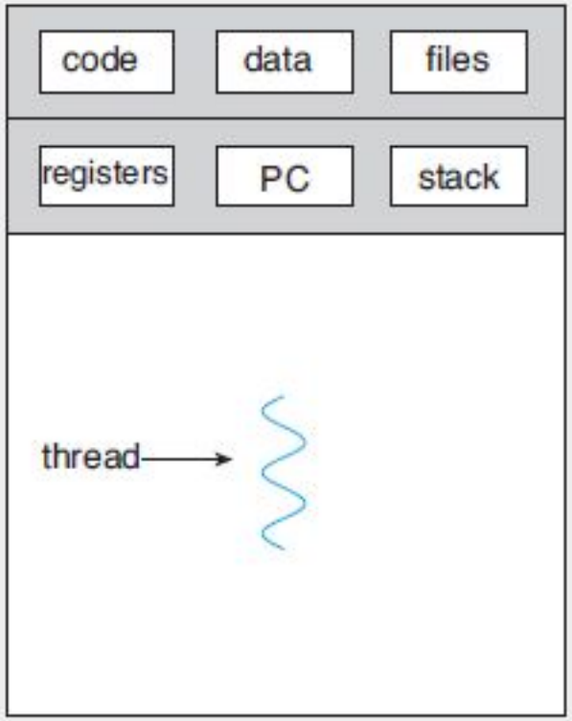
Threads

- Unidad básica de utilización de la CPU.
- Tiene su propio ID, PC, registros, y stack.
- Comparte la sección de código, datos y otros recursos (como archivos abiertos).

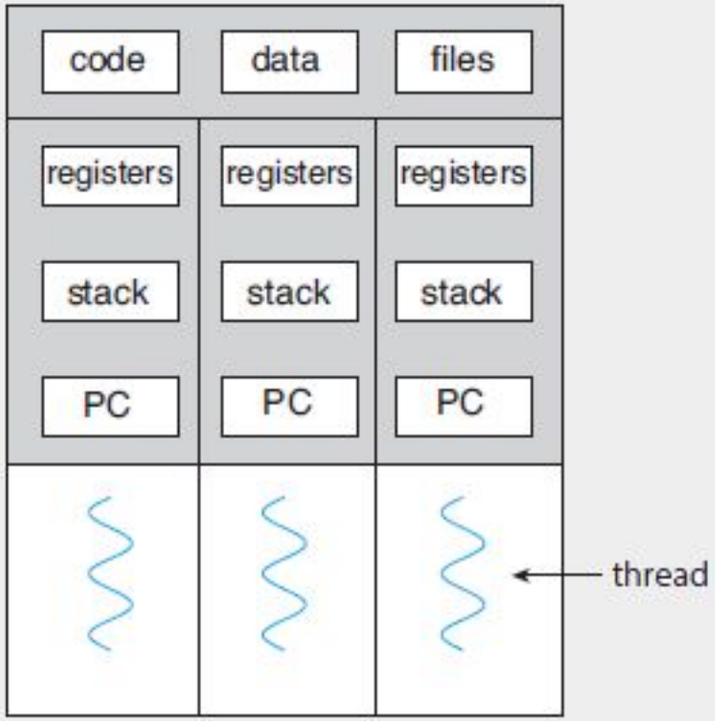
Otras denominaciones:

- Hilo de control
- Proceso de peso liviano (LWP)
- Thread de control

Un thread NO ES UN PROCESO HIJO



single-threaded process



multithreaded process

Threads

Si...

- n procesos en un sistema con m CPU son concurrentes/paralelos
  - comparten recursos del SO (dispositivos lógicos, semáforos, etc)

Si...

- n threads en un proceso con m CPU son concurrentes/paralelos
  - comparten recursos del proceso

Threads

Si...

- n procesos en un sistema con m CPU son concurrentes/paralelos
  - comparten recursos del SO (dispositivos lógicos, semáforos, etc)

Si...

- n threads en un proceso con m CPU son concurrentes/paralelos
  - comparten recursos del proceso

Entonces...¿tiene algún sentido tener THREADS?????

Threads

Si...

- n procesos en un sistema con
  - comparten recursos del S

Si...

- n threads en un proceso con m
  - comparten recursos del p

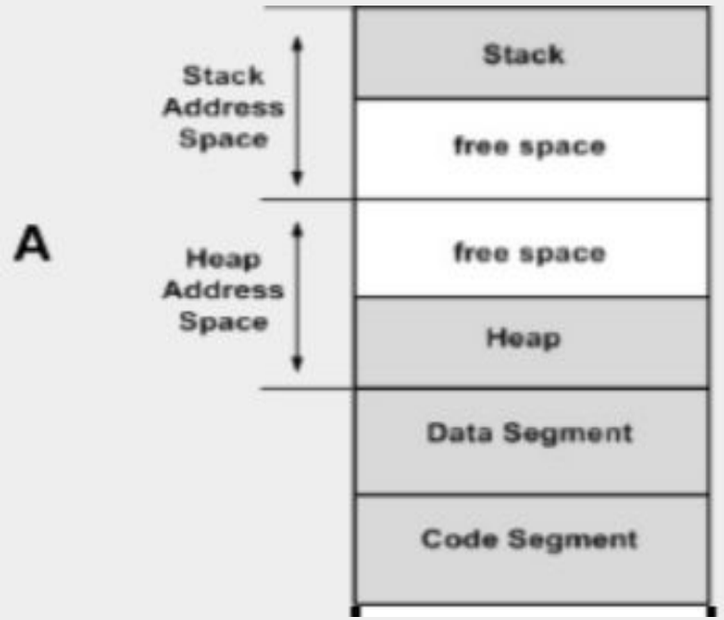
Entonces...¿tiene algún sentido te

Threads

- Es la unidad básica de utilización de la CPU.
- Tiene su propio ID, PC, registros, y stack.
- Comparte la sección de código, datos y otros recursos (como archivos abiertos).

Otras denominaciones:

- Hilo de control
- Proceso de peso liviano (LWP)
- Thread de control



Repaso: ¿Cómo implementa un SO el control de los procesos?

Threads

Algunos posibles beneficios:

- Capacidad de respuesta
- Compartir recursos
  - Menor overhead en:
    - la creación y finalización de threads
    - el cambio de contexto.
- Escalabilidad

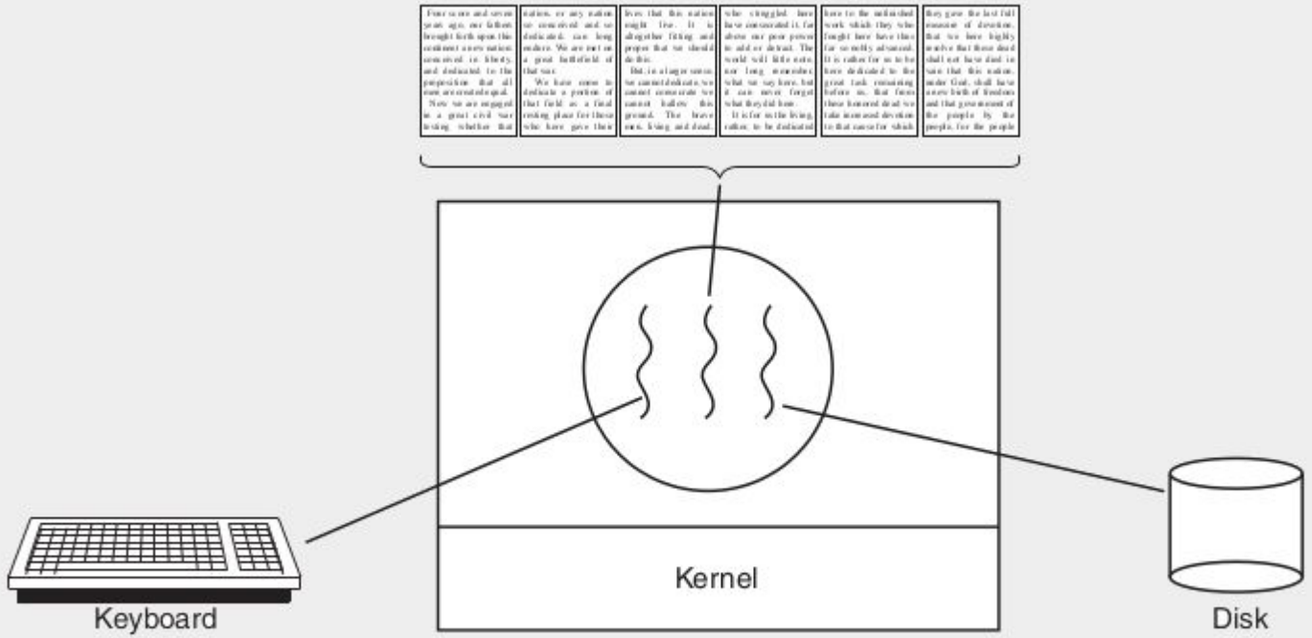


Figure 2-7. A word processor with three threads.

Threads

Algunos posibles beneficios:

- Capacidad de respuesta: un thread espera I/O, otro utiliza CPU, etc.
- Compartir recursos: compartir codigo y datos, archivos abiertos, etc.
  Menor overhead en:
  la creación y finalización de threads
  el cambio de contexto.
- Escalabilidad: aprovechar multiples CPU y tener paralelismo real.

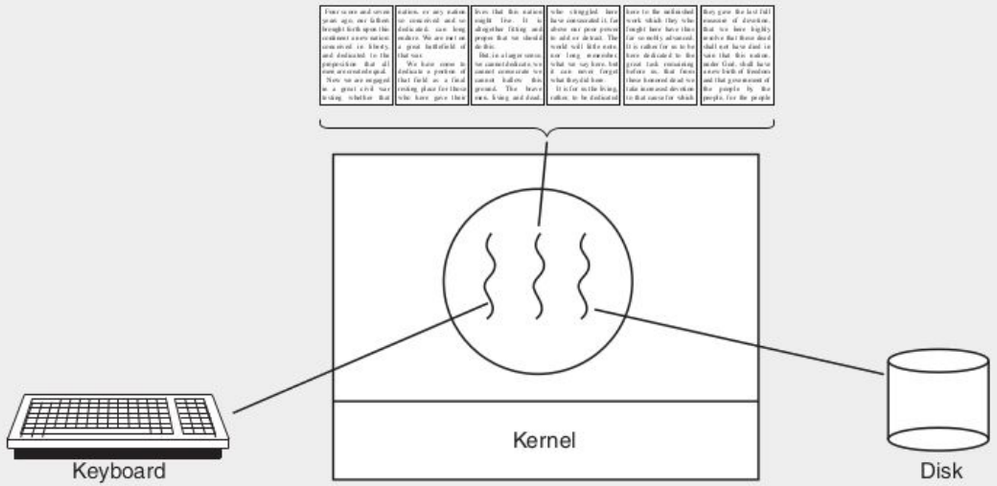
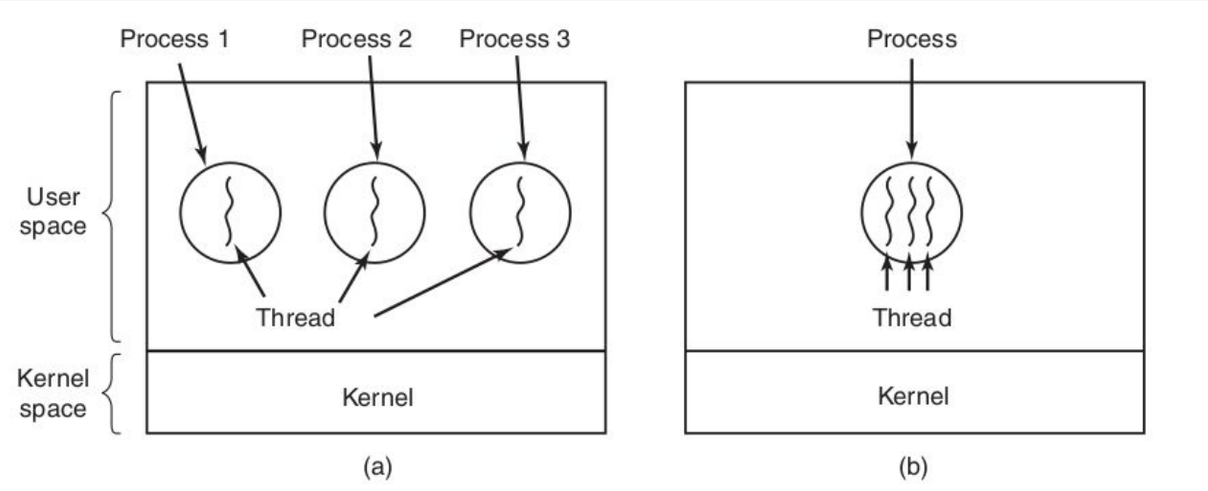


Figure 2-7. A word processor with three threads.

Sistemas Operativos I - Procesos y Threads

Threads - Implementación de threads



Con el fin de ser portables, la IEEE definió un estándar:

POSIX threads 1003.1c. Pthreads.
(Casi todos los UNIX soportan el estándar.)

Repaso: ¿Qué era POSIX y para qué es útil?

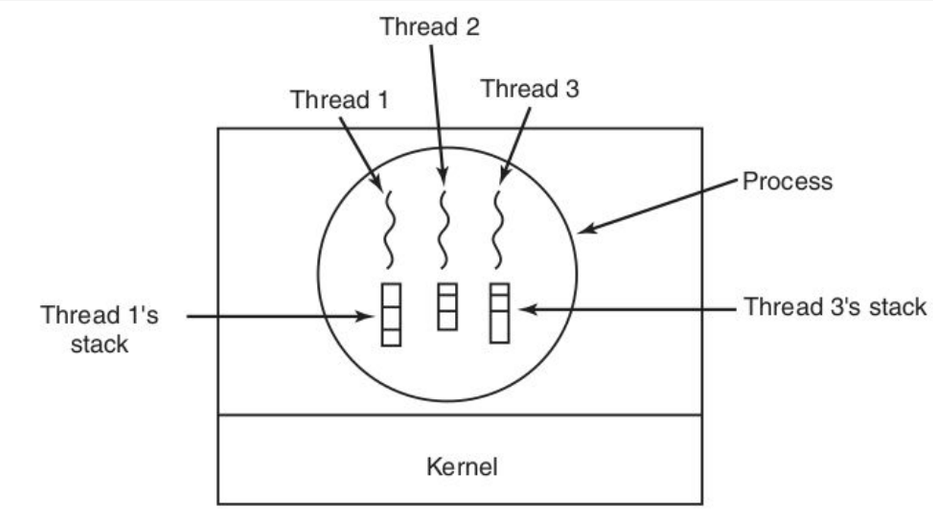


Figure 2-13. Each thread has its own stack.

Threads - Implementación de threads

- En espacio de usuario.
  En el ambiente de tiempo de ejecución (run-time system)
- En espacio del kernel

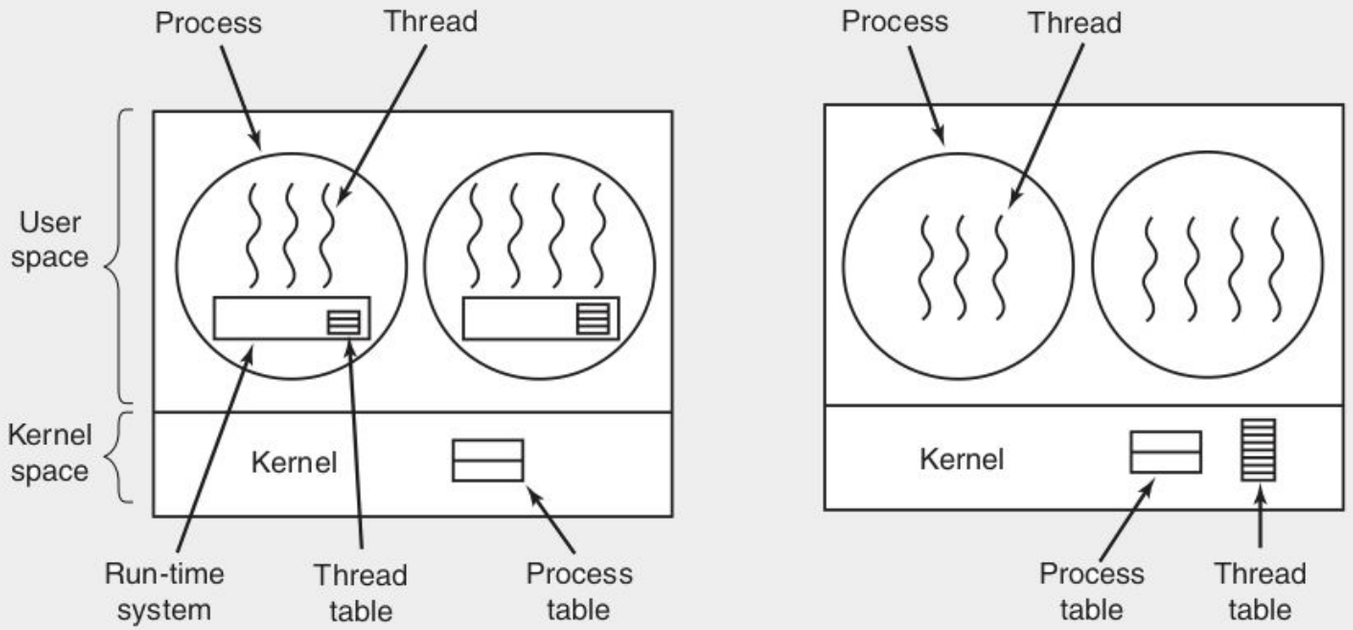


Figure 2-16. (a) A user-level threads package. (b) A threads package managed by the kernel.

Sistemas Operativos I - Procesos y Threads

Threads - Implementación de threads

Implementación en espacio de usuario

Ventajas:

- No es necesario pasar a modo kernel ni ejecutar código del SO. Mejor rendimiento que kernel threads.
- Aún mejor que planificar diferentes procesos (el planificador de threads tiene mejor rendimiento)
- Cada proceso puede tener un algoritmo de planificación diferente para sus threads

Desventajas

- ¿Cómo gestionar las llamadas al sistema bloqueantes?
  Posibilidad : cambiar la API de read, etc. O usar select en UNIX.
- Si existe un page fault el proceso es bloqueado por el SO.
- El thread toma la CPU y no la libera.
  Posibilidad: implementar una señal de clock periódica.
- Las aplicaciones más destinadas a ser resueltas con threads realizan muchas llamadas al sistema bloqueantes.

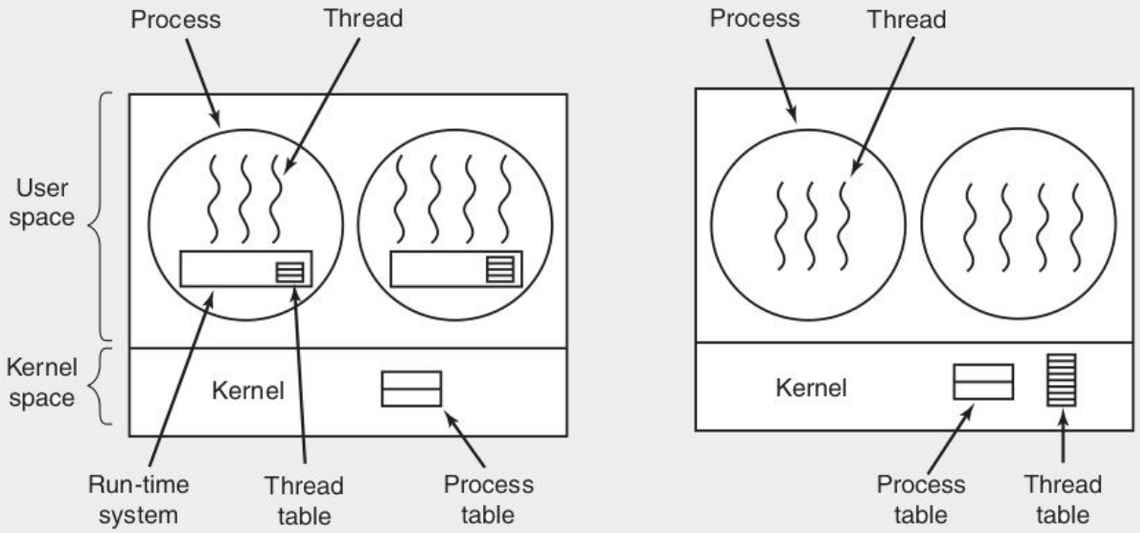


Figure 2-16. (a) A user-level threads package. (b) A threads package managed by the kernel.

Sistemas Operativos I - Procesos y Threads

Threads - Implementación de threads

Implementación en espacio del kernel

Ventajas:

- Soluciona casi todos los problemas presentes en la implementación de threads en espacio de usuario.

Desventajas

- La creación y finalización de threads es “mas costosa”.
  Posibilidad : al finalizar un thread no liberar su gestión y reusarla luego.
- Ante un system call desde un thread el SO podría ejecutar otro thread (bien), pero también podría switchear a otro proceso (ouch).
- Un system call sigue siendo complejo.
- ¿Cómo tratar las señales recibidas por el proceso?
- ¿Cómo tratar a un fork()?

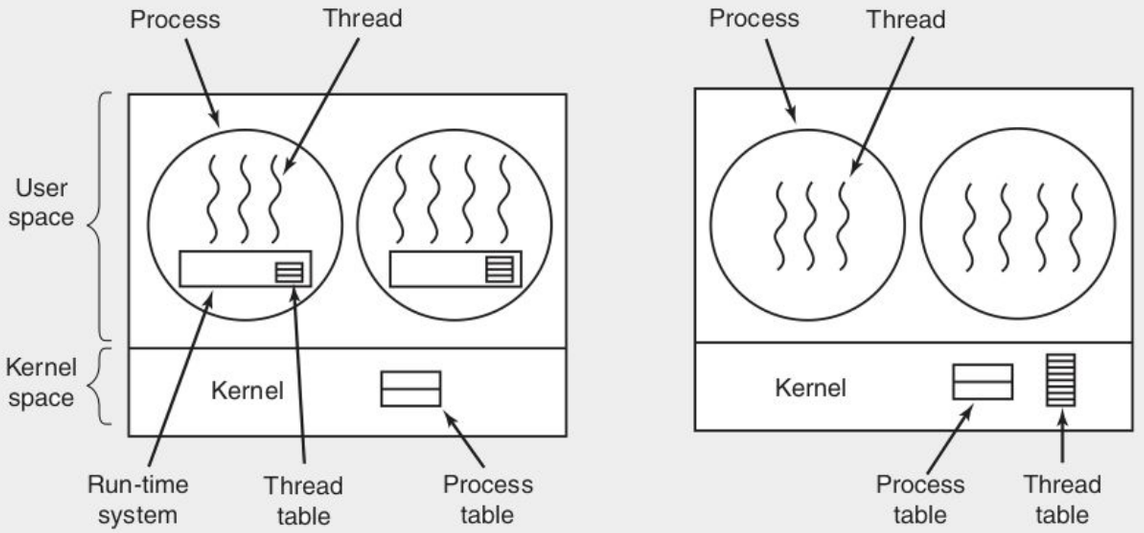


Figure 2-16. (a) A user-level threads package. (b) A threads package managed by the kernel.

Sistemas Operativos I - Procesos y Threads

Threads - Implementación de threads

Implementación en Sistemas Operativos

- Linux, Windows, Apple OS y otros sistemas operativos tradicionales de PC y servidores contienen implementación en espacio del kernel.
- También es posible usar threads implementados en espacio de usuario (Java, biblioteca de C posix threads, etc).
- Xinu y otros RTOS: el modelo de procesos es el de threads.
  Desventaja:
  Los procesos no tienen memoria protegida e independiente, todos comparten el segmento de código y de datos.
  Las pilas son independientes, pero podrían llegar a solaparse y corromper el sistema.

```c
/* Ejemplo Linux. Compilar con: gcc -o p p.c -lpthread */

#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>

void *p_msg( void *ptr );

main()
{
     pthread_t thread1, thread2;
     char *msg1 = "Thread 1";
     char *msg2 = "Thread 2";
     int  r1, r2;

    /* Create independent threads each of which will execute function */
     r1 = pthread_create( &thread1, NULL, p_msg, (void*) msg1);
     r2 = pthread_create( &thread2, NULL, p_msg, (void*) msg2);

     /* Wait till threads are complete before main continues.  */
     pthread_join( thread1, NULL);
     pthread_join( thread2, NULL);

     printf("Thread 1 returns: %d\n",r1);
     printf("Thread 2 returns: %d\n",r2);
     exit(0);
}

void *p_msg( void *ptr )
{
     char *m;
     m = (char *) ptr;
     printf("%s \n", m);
}
```

Sistemas Operativos I - Procesos : planificación

Planificación de procesos

Estados de un proceso en Xinu