# Concurrencia

Procesos

Threads

# Introducción a Procesos

- Todas las computadoras moderas realizan varias cosas al mismo tiempo.
- En cada instante la CPU ejecuta un único programa.
- Existen programas con un único hilo de control.
- Los sistemas de tiempo real son inherentemente concurrentes.

# El modelo de los procesos secuenciales

Un único procesador puede compartirse entre varios procesos utilizando un algoritmo de planificación que determine cuándo hay que detener el trabajo sobre un proceso y pasar a atender a otro diferente.

# El modelo de los procesos secuenciales

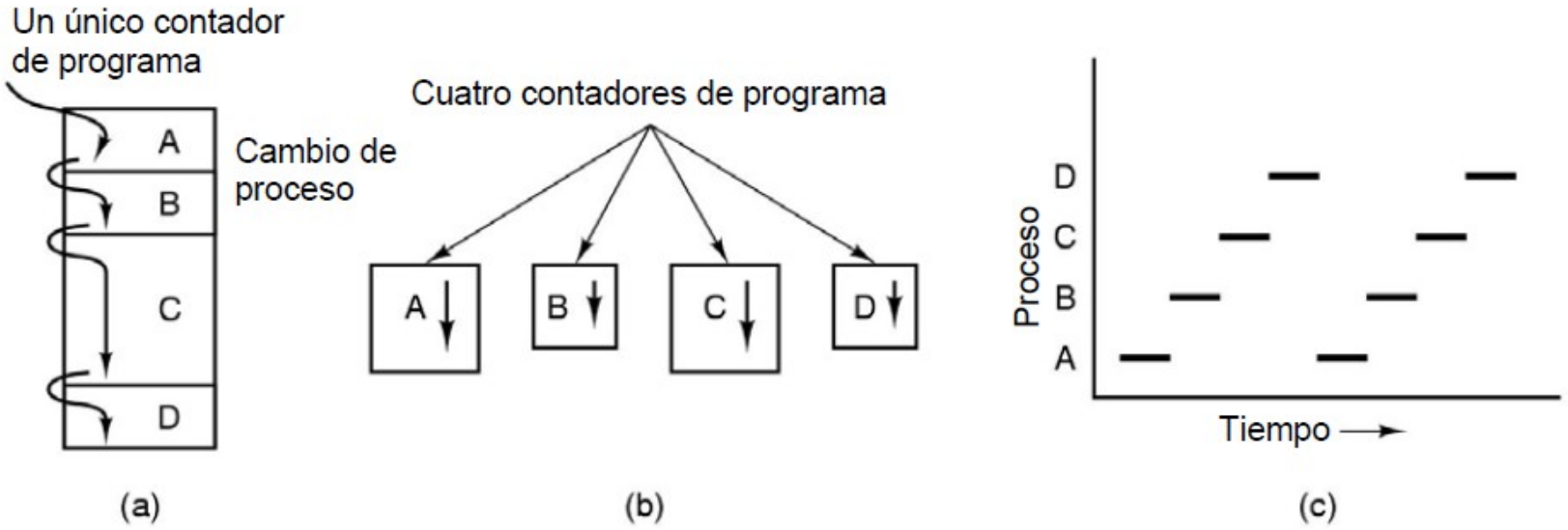



**Figura 2-1.** (a) Multiprogramación de cuatro programas. (b) Modelo conceptual de cuatro procesos secuenciales independientes. (c) Sólo un programa está activo en cada momento.

# El modelo de los procesos secuenciales

- La CPU conmuta de un proceso a otro.
- La velocidad a la cual un proceso realiza su cómputo no es uniforme y probablemente ni siquiera es reproducible.
- Los procesos no deben programarse bajo suposiciones preconcebidas sobre su velocidad de ejecución.
- Cuando un proceso tiene requerimientos de tiempo real críticos es necesario tomar medidas especiales para asegurar que efectivamente los sucesos ocurran dentro de ciertos límites de tiempo.

# Procesos concurrentes

El trabajar con procesos concurrentes añade complejidad a la tarea de programar.

¿Cuáles son entonces los beneficios que aporta la programación concurrente?

# Beneficios de la programación concurrente

- Mejor aprovechamiento de la CPU.
- Velocidad de ejecución.
- Solución de problemas de naturaleza concurrente:
  - Sistemas de control
  - Tecnologías web
  - Aplicaciones basadas en interfaces de usuarios
  - Simulación
  - SGDB

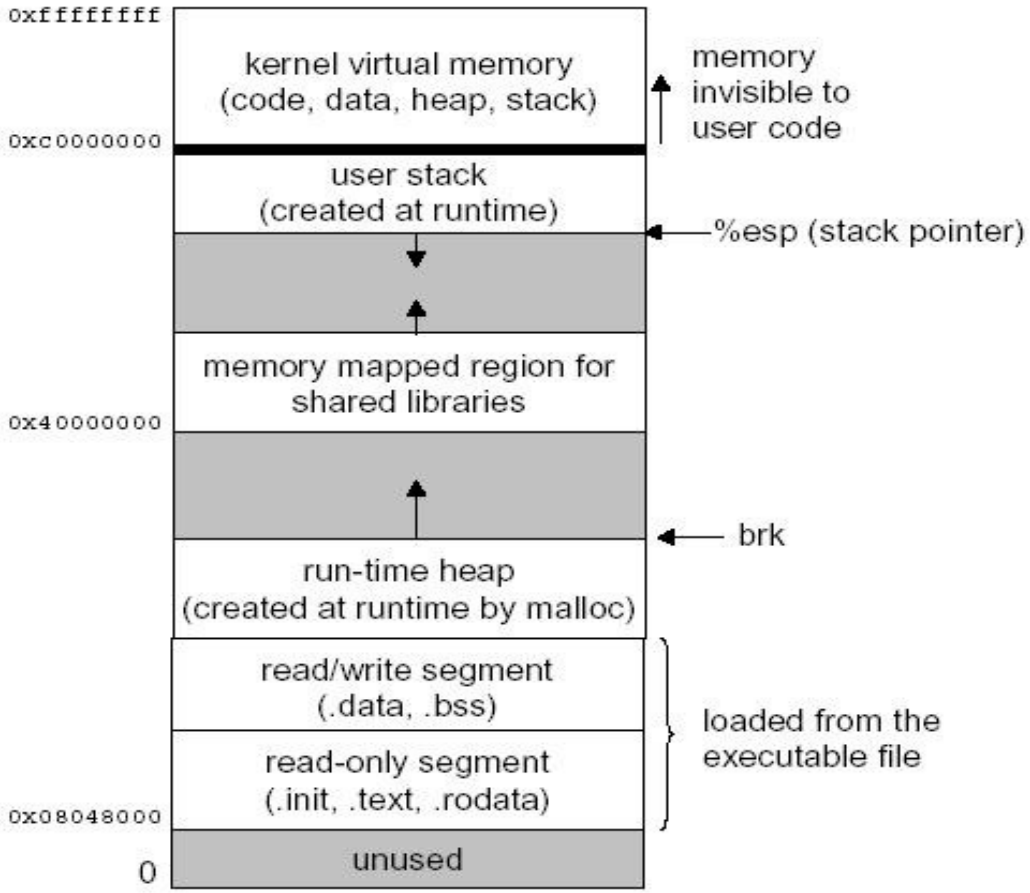
# Procesos

- Es una instancia de ejecución de un programa.
- Además del código fuente (texto) incluye:
  - un contador de programa
  - contenido de registros del procesador
  - stack (pila): con datos temporales como parámetros de funciones, variables locales, etc.
  - sección de datos: variables globales.
  - heap: memoria alocada dinámicamente.
- Tiene un ciclo de vida, es decir pasa por distintos estados.

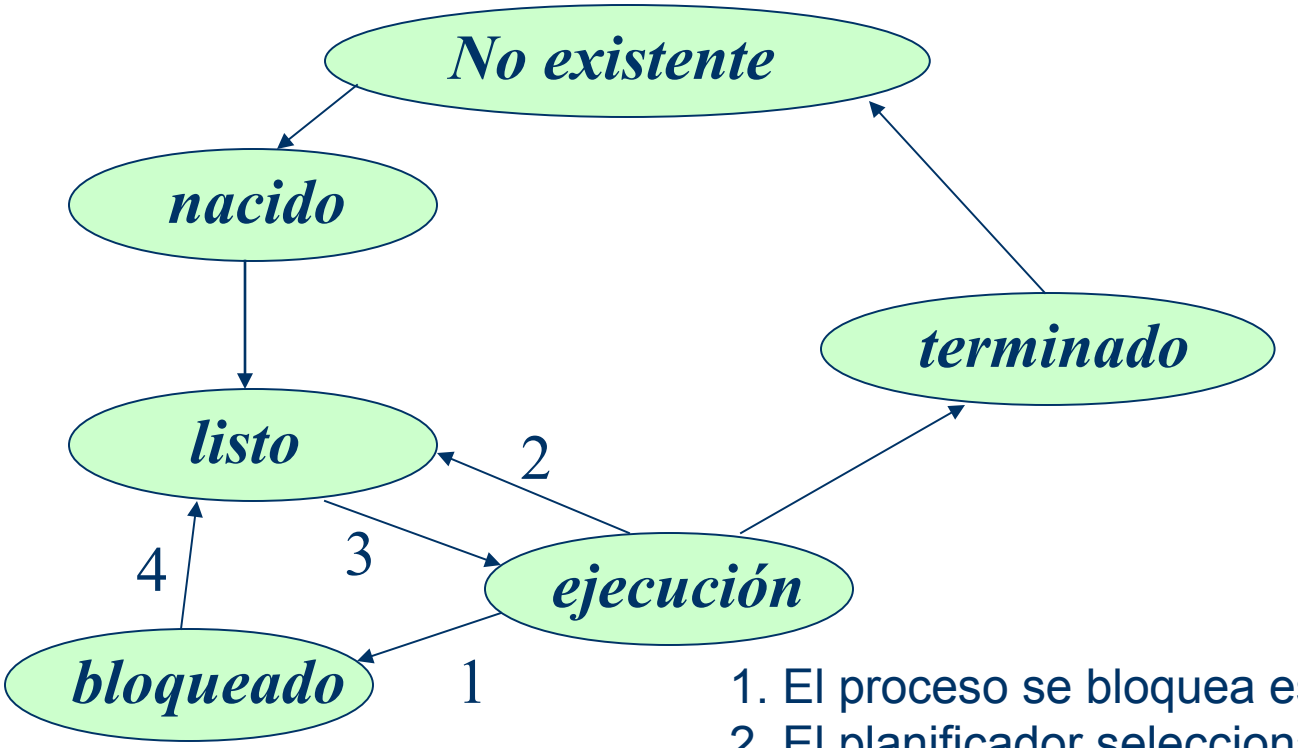
# Procesos

- A todo proceso se le asigna un espacio de direccionamiento que representa las zonas de memoria asignadas al proceso.
- Este espacio incluye:
  - El código del proceso.
  - Los datos del proceso. Se divide en variables inicializadas y no inicializadas.
  - Código y datos de bibliotecas.
  - La pila.

# Procesos


0xffffffff
kernel virtual memory
(code, data, heap, stack)
memory
invisible to
user code
0xc0000000
user stack
(created at runtime)
%esp (stack pointer)
memory mapped region for
shared libraries
0x40000000
brk
run-time heap
(created at runtime by malloc)
read/write segment
(.data, .bss)
loaded from the
executable file
read-only segment
(.init, .text, .rodata)
0x08048000
0
unused

# Ciclo de vida de un proceso

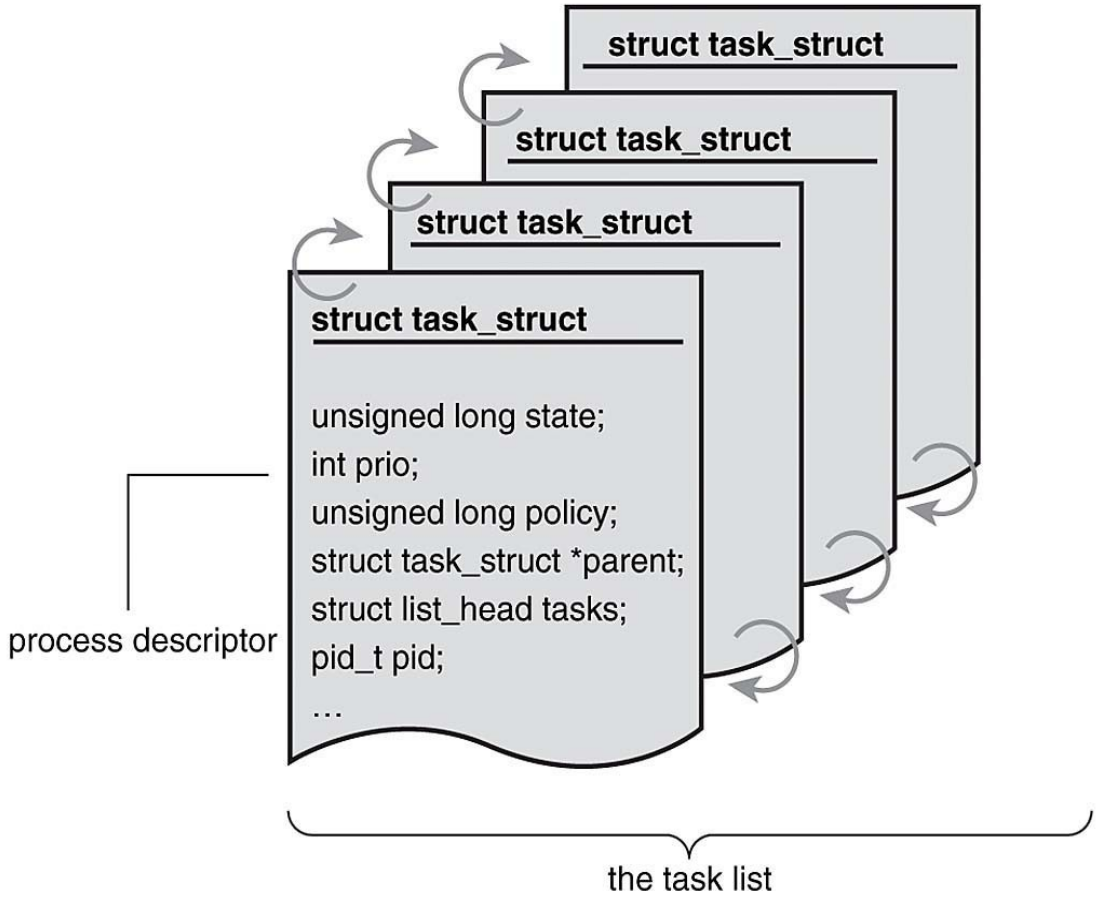



1. El proceso se bloquea esperando un dato
2. El planificador selecciona otro proceso
3. El planificador selecciona este proceso
4. El dato está disponible

# Atributos de un proceso

- Estado.
- Identificadores (proceso, usuario, grupo, etc.)
- Valor de registros, incluyendo PC.
- Identidad del usuario.
- Prioridad (política y mecanismos de planificación).
- Espacio de almacenamiento.
- Descriptores de archivos.

# Descriptor de un proceso



# Creación de un proceso

1. La inicialización del sistema
2. La ejecución por parte de un proceso (en ejecución) de una llamada al sistema de creación de un nuevo proceso.
3. La petición por parte del usuario de la creación de un nuevo proceso.
4. El inicio de un trabajo en batch.

# Finalización de un proceso

1. Finaliza la ejecución de su cuerpo.
2. Ejecución de alguna sentencia de auto finalización.
3. Condición de error sin tratar.
4. Aborto por medio de la intervención de otro proceso.
5. Nunca: procesos que se ejecutan en bloques que no terminan.
6. Cuando ya no son necesarios.

# Cambio de contexto de un proceso

- Guardar el contexto del proceso (registros, contador, direccionamiento, etc)
- Cambiar el estado y guardar información acerca de la interrupción.
- Seleccionar un programa para su ejecución.
- Restaurar el contexto del nuevo programa.
- Ejecutar el programa seleccionado.

# Ejecución de los procesos

- Todos los SO tienen formas de crear procesos.
- Cada proceso se ejecuta en su propia máquina virtual.
- **`fork()`** crea un nuevo proceso duplicando al proceso que lo invoca.

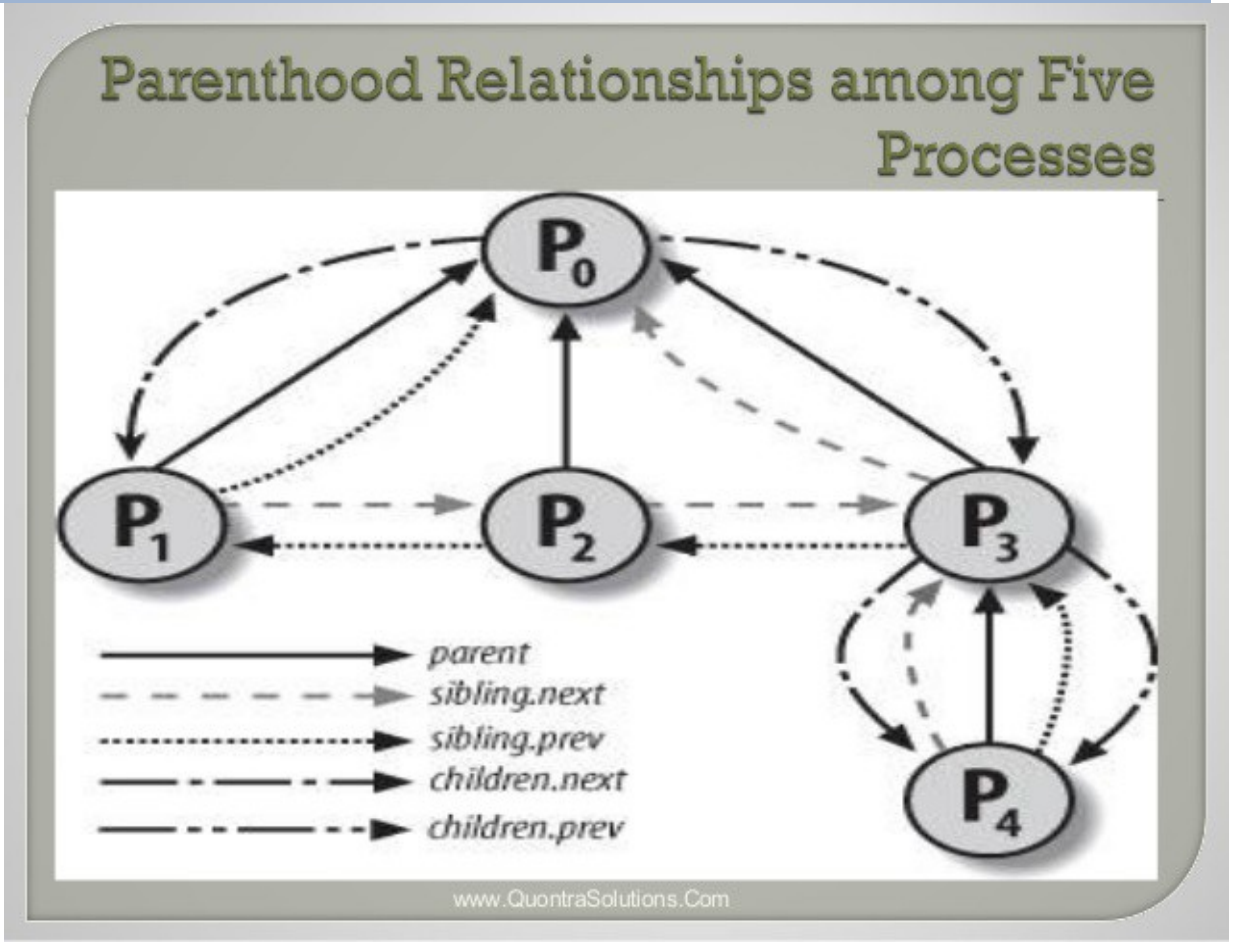
# Creación de procesos Linux

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

int value = 5;

int main()
{
pid_t pid;

   pid = fork();

   if (pid == 0) {/* child process */
      value += 15;
   }
   else if (pid > 0) {/* parent process */
      wait(NULL);
      printf("PARENT: value = %d",value); /* LINE A */
      exit(0);
   }
}
```

# Creación de procesos Linux

- El nuevo proceso es llamado proceso *hijo* y es una copia exacta del proceso *padre*.
- `fork()` retorna 0 al hijo y el PID del hijo al padre.
- La utilización de recursos es reseteada a 0 en el hijo.
- El espacio de direcciones es replicado (se copian todas las variables y archivos abiertos).
- Ejemplo: intérpretes de comandos.

# Jerarquía de procesos



# Comunicación entre procesos: IPC

- Memoria compartida
- Tuberías (pipes)
- Paso de mensajes
- Semáforos

**Esta comunicación es costosa.**

# Clonado de procesos

- Existe en linux la primitiva **`clone()`** que crea un proceso por duplicación de su padre.
- Puede compartir sólo una parte de su contexto con su padre:
- Espacio de direccionamiento (segmentos de código y datos).
- Información de control del sistema de archivos (directorios raíz y actual).

# Clonado de procesos

- Descriptores de archivos abiertos.
- Gestores de señales.
- Identificador de procesos (ambos procesos comparten el mismo número).
- Esto permite ejecutar varias actividades sin necesidad de IPC, compartiendo simplemente datos
- Los SO modernos permiten crear hilos o threads (procesos ligeros) dentro de la misma máquina virtual.

# Threads

- Hay situaciones en las que es deseable contar con múltiples hilos de control (threads) en el mismo espacio de direcciones ejecutándose quasi-paralelamente, como si fueran procesos separados.
- Tener múltiples threads ejecutándose en paralelo dentro de un proceso es análogo a tener múltiples procesos ejecutándose en paralelo dentro de un ordenador.

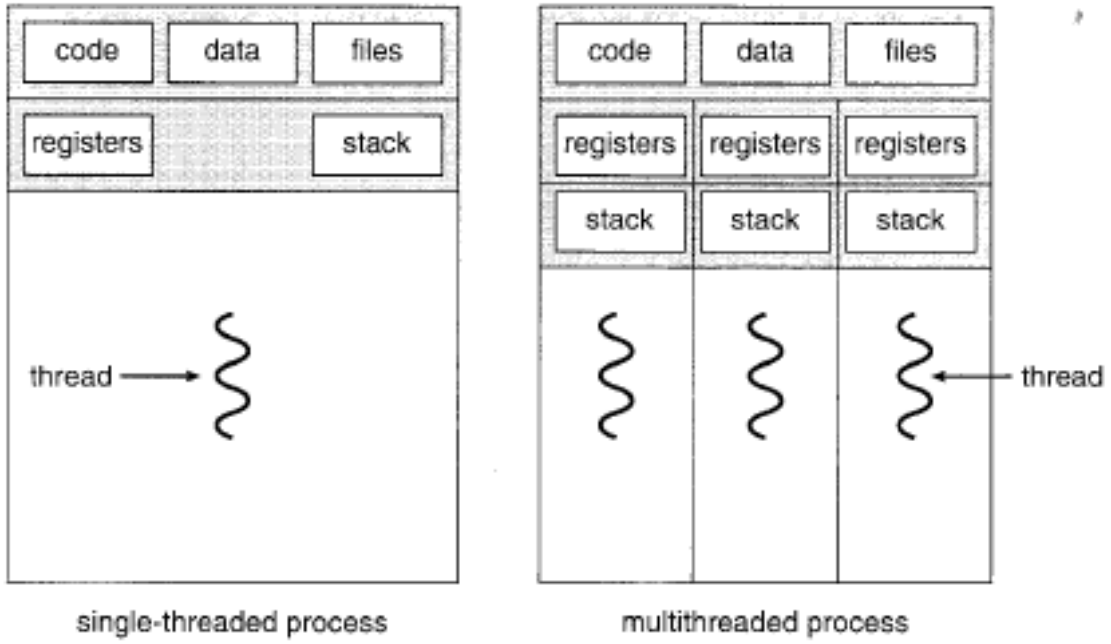
# Threads

- Comprenden un ID, un contador de programa, un conjunto de registros y una pila.
- Comparten con otros threads pertenecientes al mismo proceso: la sección de código, la sección de datos y otros recursos del SO tales como archivos abiertos y señales.
- No son jerárquicos.

# Threads

- Se los suele llamar **procesos ligeros** (*lightweight process*).
- También se utiliza el término de **multihilo** (*multithreaded*) para describir la situación en la cual se permite que haya múltiples threads en el mismo proceso.

# Threads




single-threaded process

multithreaded process

# Threads

- Un thread puede estar en cualquiera de los estados de un proceso tradicional.
  - Un thread en ejecución tiene actualmente la CPU y está activo.
  - Un thread bloqueado está esperando a que algún suceso lo desbloquee. Un thread puede bloquearse esperando a que tenga lugar algún suceso externo o a que algún otro thread lo desbloquee.
  - Un thread listo está planificado para ejecutarse y lo hace tan pronto como le llega su turno.

# Threads

- Las transiciones entre los estados de un thread son las mismas que las transiciones entre los estados de un proceso.
- Los cambios de contexto son mucho más rápidos.

código

# Utilización de Threads

- Mejora el tiempo de respuesta.
  - Por ejemplo un browser multihilo puede permitir interacción con un usuario mientras se carga una imagen.
- Son más fáciles de crear y destruir que los procesos.
  - En numerosos sistemas, la creación de un thread puede realizarse 100 veces más rápido que la creación de un proceso.
- Memoria compartida => Comunicación.

# Ejemplo

- Un procesador de texto.
- La mayoría de los procesadores de texto visualizan en la pantalla el documento que se está creando formateado exactamente como aparecería una vez impreso.
- Todos los saltos de línea y de página aparecen en su posición correcta final.

# Ejemplo

- Supongamos que el usuario está escribiendo un libro.
  - Desde el punto de vista del autor es más cómodo meter el libro entero en un único archivo (búsqueda por temas, sustituciones globales, etc)
  - Alternativamente, puede ponerse cada capítulo en un archivo separado.
- Se borra una frase de la página 1 de un documento de 800 páginas.
- Se busca la página 600. El procesador de texto se ve forzado a reformatear inmediatamente todo el libro hasta la página 600.
- Espera considerable.

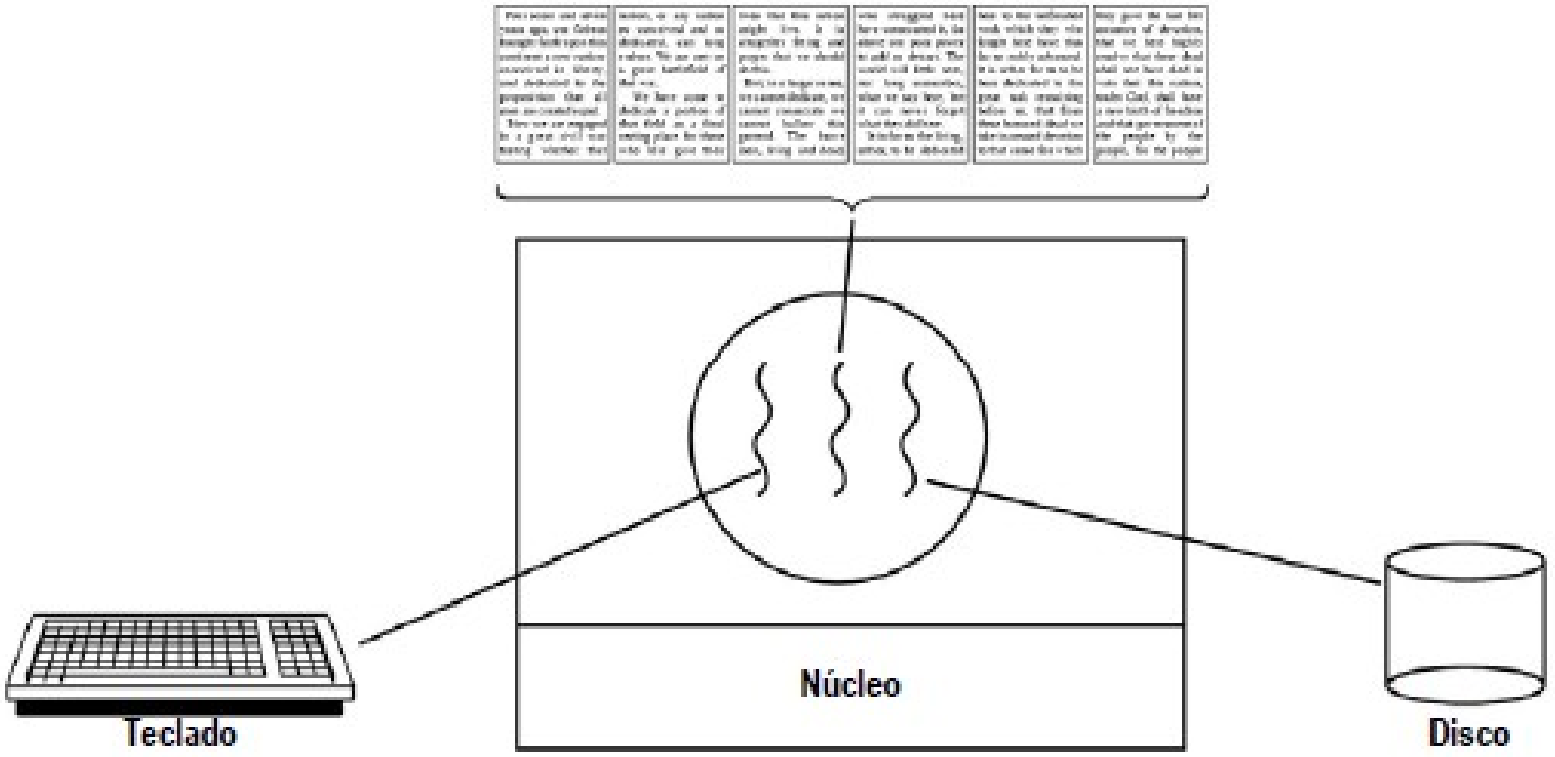
# Ejemplo

- Los threads pueden ayudarnos.
- Un thread interactúa con el usuario (atiende teclado, mouse, scrolls).
- Otro realiza el reformateo como una actividad de fondo (tan pronto como se borra la frase de la página 1).
- Con un poco de suerte, el reformateo se completa antes de que el usuario pida ver la página 600, de forma que en ese momento puede visualizarse instantáneamente.

# Ejemplo

- Muchos procesadores de texto ofrecen la posibilidad de guardar automáticamente todo el archivo en el disco cada pocos minutos.
- El tercer thread puede ocuparse de los backups en el disco sin interferir con los otros dos.

# Ejemplo




Un procesador de texto con tres threads.

# Threads

- Desventajas de trabajar con hilos:
  - Protección de variables compartidas (concurrencia).
  - Deadlocks, inversión de prioridades.
  - Dificultad para depurar.
- Se recomienda la utilización cuando:
  - Muchos cómputos pueden correr en paralelo.
  - Existen actividades asíncronas.
  - Sistemas de tiempo real.
  - Sistemas distribuidos.