Universidad Nacional Mayor de San Marcos
Universidad del Perú. Decana de América

Programación concurrente y paralela

Semana 2

28/08/2025 – Escuela Profesional de Ingeniería de Software

Proyecto fin de curso

Cada grupo seleccionara uno de los siguientes proyectos, en el cual, buscarán una solución paralela (que podrán realizar o encontrar pero entender y mejorar) con cualquiera de los paradigmas dados (memoria compartida con OpenMP, memoria distribuida con MPI o memoria híbrida con CUDA), en una solución en C o C++ exclusivamente (no se aceptan otros lenguajes). Tengan en cuenta que deben ser problemas con muchos datos (bastante grandes que un pc no pueda realizarlo). En cada propuesta, se les da un enlace de base con ejemplos de códigos en C para cada una de las propuestas. Estos ejemplos son eso, ejemplos y no necesariamente la base de su proyecto, pero si una guia.

1) Clasificación paralela: implemente un algoritmo de clasificación paralela que pueda ordenar de manera eficiente grandes conjuntos de datos mediante el uso de múltiples procesadores o subprocesos. (Ver: https://www.geeksforgeeks.org/sorting-algorithms/)

2) Multiplicación de matrices paralelas: desarrolle un algoritmo paralelo para multiplicar matrices, lo que puede acelerar significativamente el cálculo de matrices grandes. (Ver: https://www.geeksforgeeks.org/sorting-algorithms/)

3) Algoritmos genéticos paralelos: implemente algoritmos genéticos paralelos para resolver problemas de optimización, como encontrar la solución óptima para un problema determinado mediante la evolución de una población de soluciones candidatas. (Ver: https://www.geeksforgeeks.org/genetic-algorithms/)

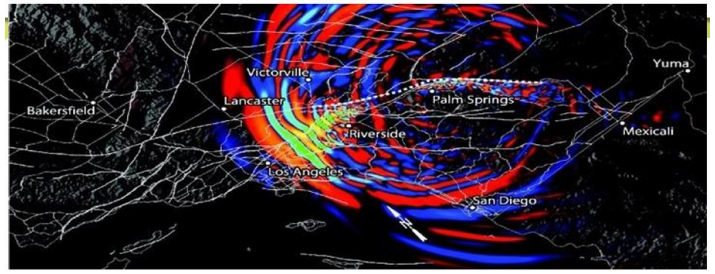
4) Procesamiento de imágenes en paralelo: cree una aplicación de procesamiento de imágenes en paralelo que pueda realizar tareas como filtrado de imágenes, detección de bordes o reconocimiento de imágenes utilizando técnicas de computación en paralelo. (Ver: https://www.geeksforgeeks.org/histogram-equalisation-in-c-image-processing/)

5) Simulación paralela: cree un programa de simulación paralela que pueda simular sistemas complejos, como el flujo de tráfico, la dinámica de la población o los patrones climáticos, automatas, distribuyendo la carga computacional entre múltiples procesadores o nodos. (Ver: https://www.geeksforgeeks.org/c-program-to-simulate-nondeterministic-finite-automata-nfa/)

6) Algoritmos de gráficos paralelos: desarrolle algoritmos paralelos para problemas relacionados con gráficos, como el recorrido de gráficos, la búsqueda de la ruta más corta o la agrupación de gráficos, para procesar de manera eficiente gráficos a gran escala. (Ver: https://www.geeksforgeeks.org/computer-graphics-2/)

7) Aprendizaje automático paralelo: implemente algoritmos paralelos para tareas de aprendizaje automático, como paralelizar procesos de entrenamiento para redes neuronales profundas o paralelizar algoritmos de selección de características. (Ver: https://www.geeksforgeeks.org/machine-learning-algorithms/)

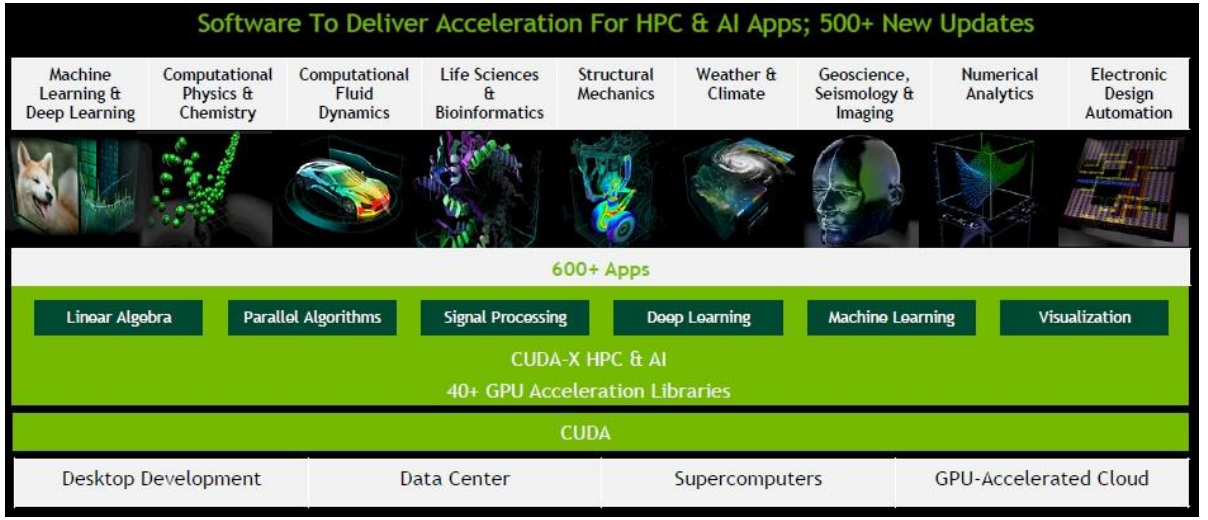
8) Simulación paralela de Monte Carlo: cree un programa de simulación paralelo de Monte Carlo para estimar el valor de problemas matemáticos complejos, como la fijación de precios de opciones o el análisis de riesgos. (Ver: https://www.geeksforgeeks.org/ml-monte-carlo-tree-search-mcts/)



Simulación de Terremoto

Rúbrica

Criterios	Logrado	En proceso	En Inicio
Utilización de las librerías de programación paralela vistas en el curso	Utiliza todas las librerías vistas en clase	Utiliza al menos dos librerías o APIs	Utiliza sólo una librería
Resolución eficiente de problemas científicos y de ingeniería	Demuestra la eficiencia lograda con la programación paralela, en comparación con la programación serial	Se logra una eficiencia mayor con la programación paralela que la programación serial pero no es de impacto	La eficiencia con la programación paralela es menor
Estudia en profundidad un esquema de programación paralela híbrida	Utiliza al menos dos esquemas de programación paralela híbrida (OpenMP+MPI, OpenMP+GPU, MPI+OpenMP+GPU)	Utiliza al menos un esquema de programación paralela híbrida (OpenMP+MPI, OpenMP+GPU, MPI+OpenMP+GPU)	No utiliza un esquema de programación paralela híbrida
Estudia algún esquema algorítmico paralelo (divide y vencerás, maestro-esclavo, pipeline, ramificación y poda)	Demuestra la optimización lograda en al menos tres esquemas algoritmicos con programación paralela	Demuestra la optimización lograda en al menos dos esquemas algoritmicos con programación paralela	Demuestra la optimización lograda en sólo un esquema algoritmico con programación paralela
Configura un entorno de hardware y software para la computación paralela	Utiliza CPU y GPU en una red de computadoras tipo cluster	Utiliza una plataforma en la nube para lograr la computación paralela del proyecto	Utiliza sólo un computador personal tipo gamer



UNMSM

Universidad Nacional Mayor de San Marcos
Universidad del Perú. Decana de América

Logro de la sesión

Al finalizar la sesión, el estudiante:

- **Identificará los componentes básicos de un hilo y contrastará hilos y procesos**
- **Describirá los beneficios y desafíos de diseñar aplicaciones multihilos.**
- **Utilizará la API de hilos Pthreads para construir una aplicación simple.**

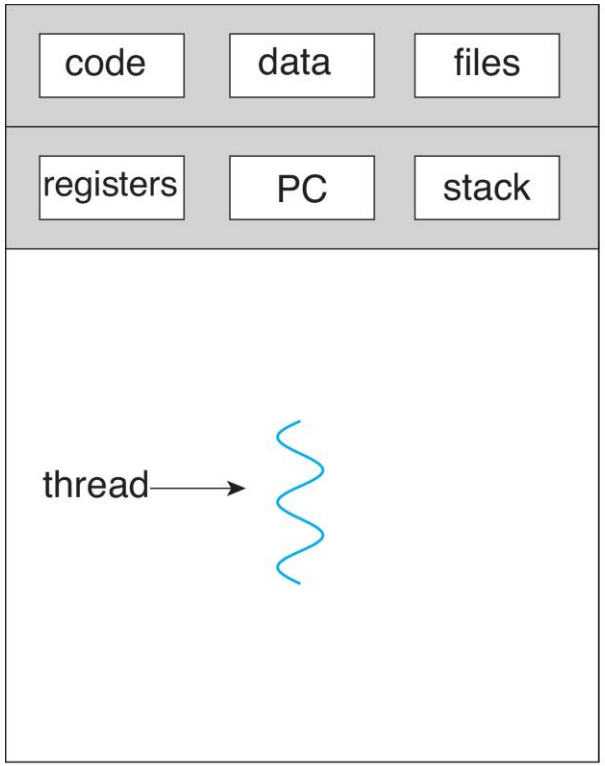
Hilos y Concurrencia

Importancia

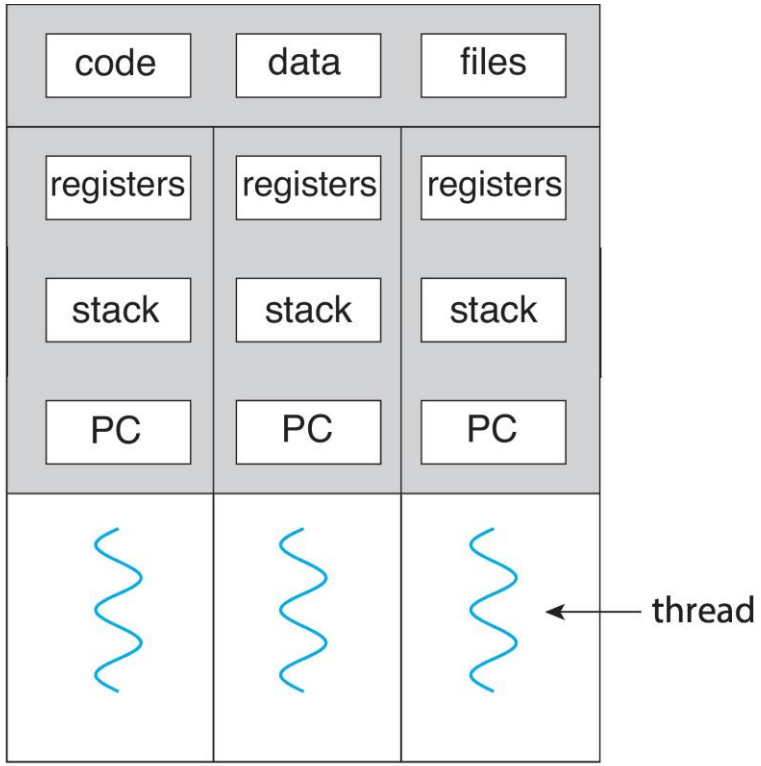
- La mayoría de las aplicaciones modernas son multihilos.
- Los hilos se ejecutan dentro de la aplicación.
- Se pueden implementar múltiples tareas con la aplicación mediante hilos separados:
  - Actualizar pantalla
  - Obtener datos
  - revisión ortográfica
  - Responder una solicitud de red
- La creación de procesos es pesada mientras que la creación de hilos es liviana
- Puede simplificar el código y aumentar la eficiencia.
- Los kernels son generalmente multihilos.

UTILIDAD

Proceso de un hilo y multihilos



single-threaded process



multithreaded process

- **¿Cómo puede este código aprovechar 2 hilos?**

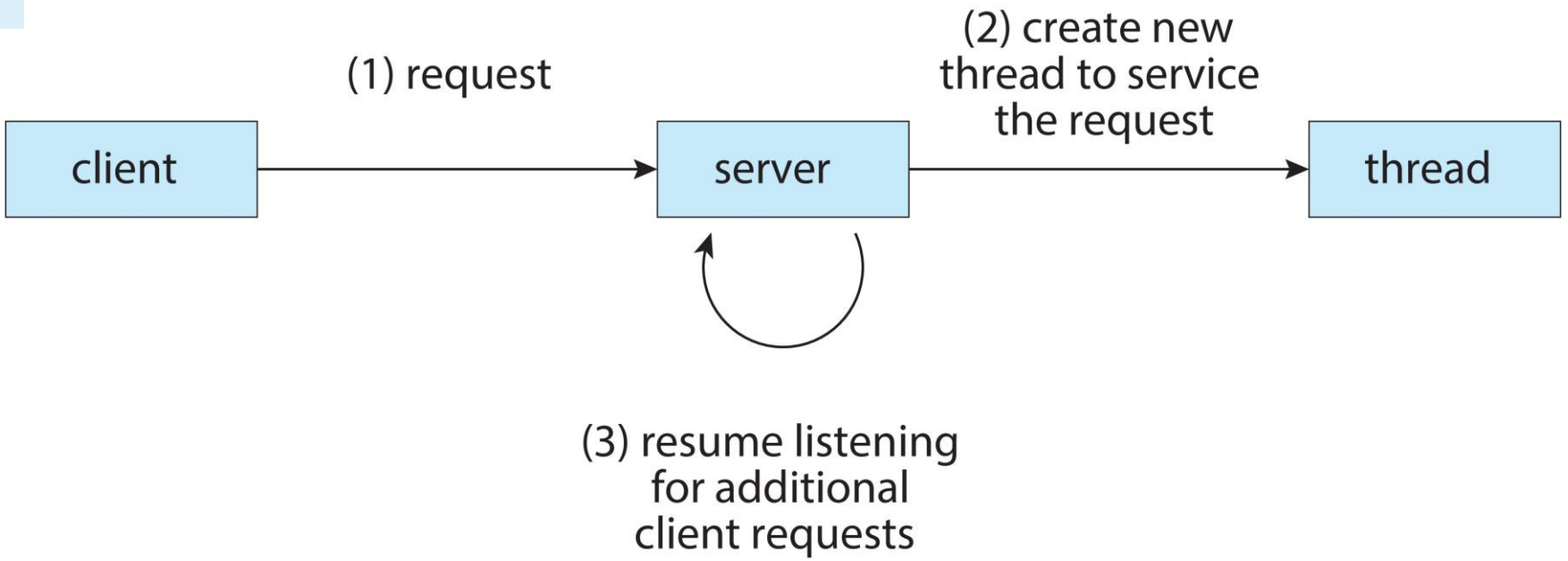
```
for(k = 0; k < n; k++)
    a[k] = b[k] * c[k] + d[k] * e[k];
```

- **Reescribe este fragmento de código como:**

```
do_mult(l, m) {
    for(k = l; k < m; k++)
        a[k] = b[k] * c[k] + d[k] * e[k];
}
main() {
  CreateThread(do_mult, 0, n/2);
  CreateThread(do_mult, n/2, n);
}
```

- ¿Qué ganamos?

Arquitectura de un Servidor multihilo

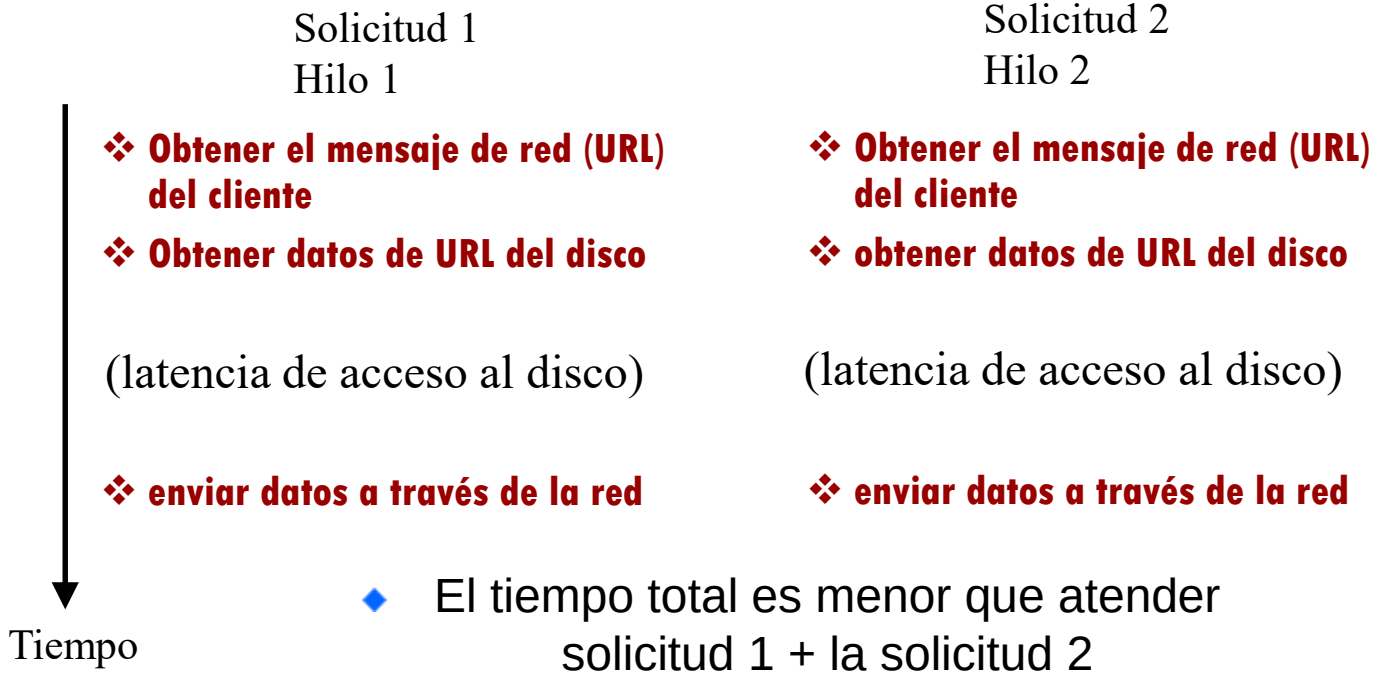


- **Considere un servidor web**

  Cree una serie de hilos, y para cada hilo haga

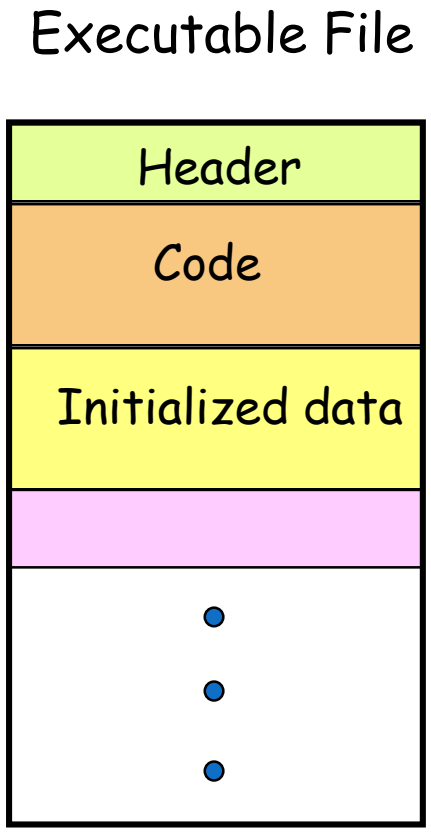
  - **obtener el mensaje de red del cliente**
  - **obtener datos de URL del disco**
  - **enviar datos a través de la red**

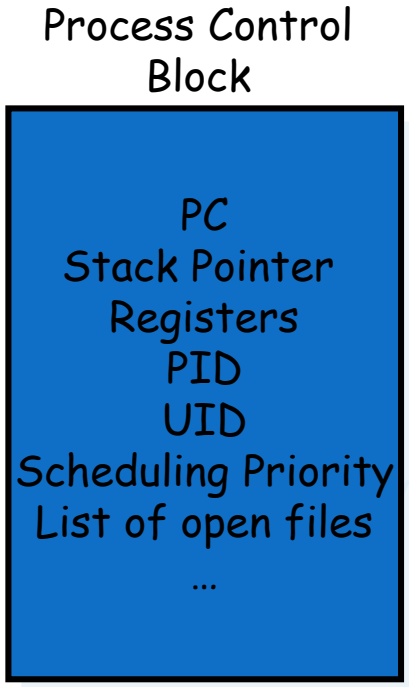
- **¿Qué ganamos?**

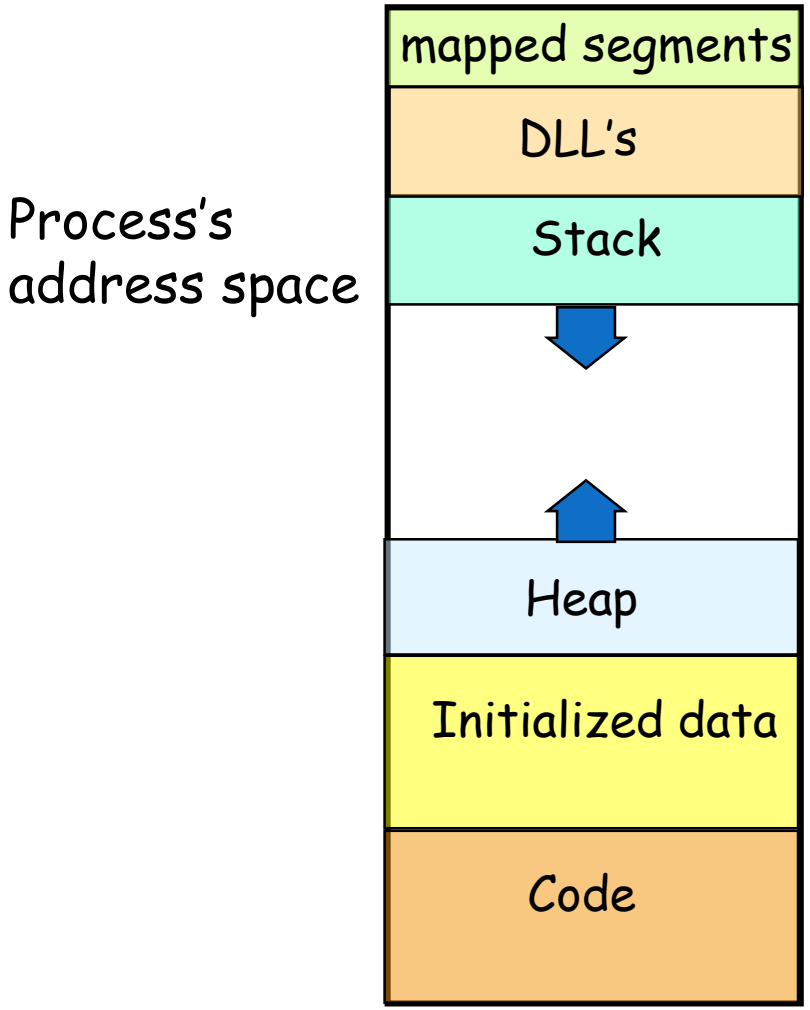


- El tiempo total es menor que atender solicitud 1 + la solicitud 2

Anatomía de un proceso







Process Control Block (PCB)

El Bloque de control de proceso contiene elementos de información asociados con un proceso específico (también llamado bloque de control de tarea - **task control block**)

Estado del proceso: En ejecución, en espera, etc.

Número o identificador del proceso (PID)

Contador de programa: Dirección de la sgte instrucción que será ejecutada

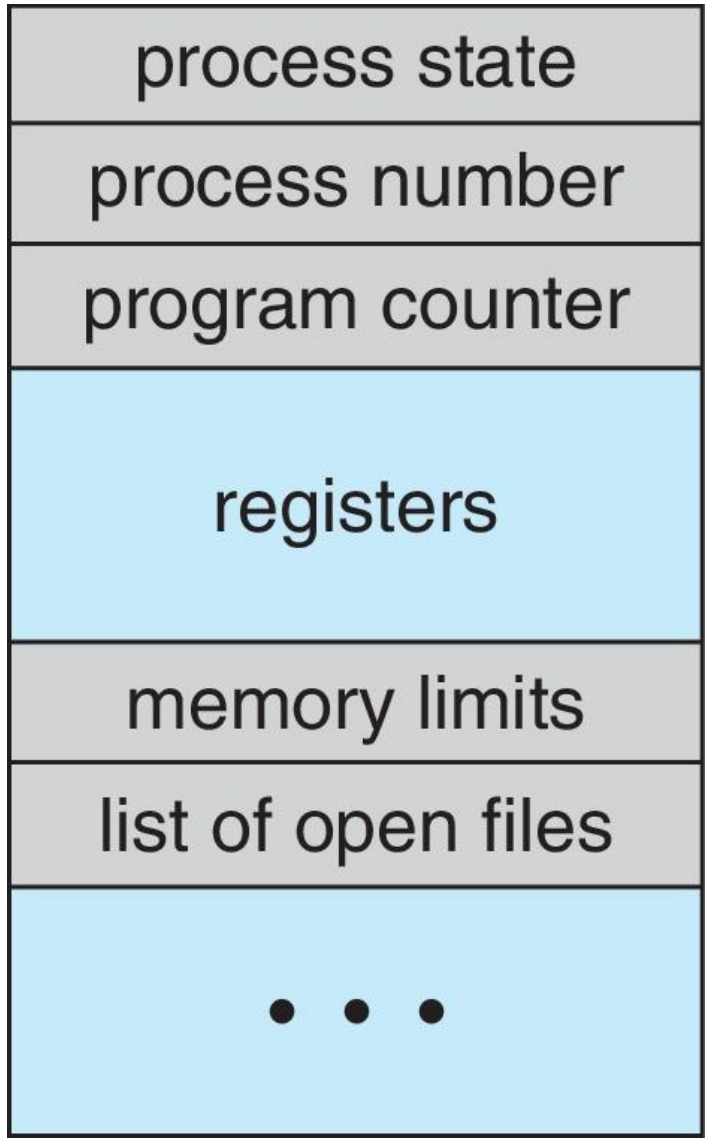
Registros del CPU: Contenido de todos los registros centrados en procesos

Información de planificación de la CPU: Prioridad, puntero a las colas de planificación

Información de gestion de memoria: Memoria asignada al proceso

Información contable – CPU utilizada, tiempo transcurrido desde el inicio, limites de tiempo asignado

Información de estado de E/S: dispositivos de E/S asignados al proceso, lista de archivos abiertos



Hilos (Threads)

Aunque normalmente pensamos que un proceso tiene un único flujo de control, en los sistemas modernos un proceso puede constar de varias unidades de ejecución, llamadas hilos, cada una de las cuales se ejecuta en el contexto del proceso y comparte el mismo código y datos globales. Los hilos son un modelo de programación cada vez más importante debido al requisito de concurrencia en los servidores de red, porque es más fácil compartir datos entre varios hilos que entre varios procesos, y porque los hilos suelen ser más eficientes que los procesos. Los hilos múltiples (Multi-threading) también son una forma de hacer que los programas se ejecuten más rápido cuando hay varios procesadores disponibles

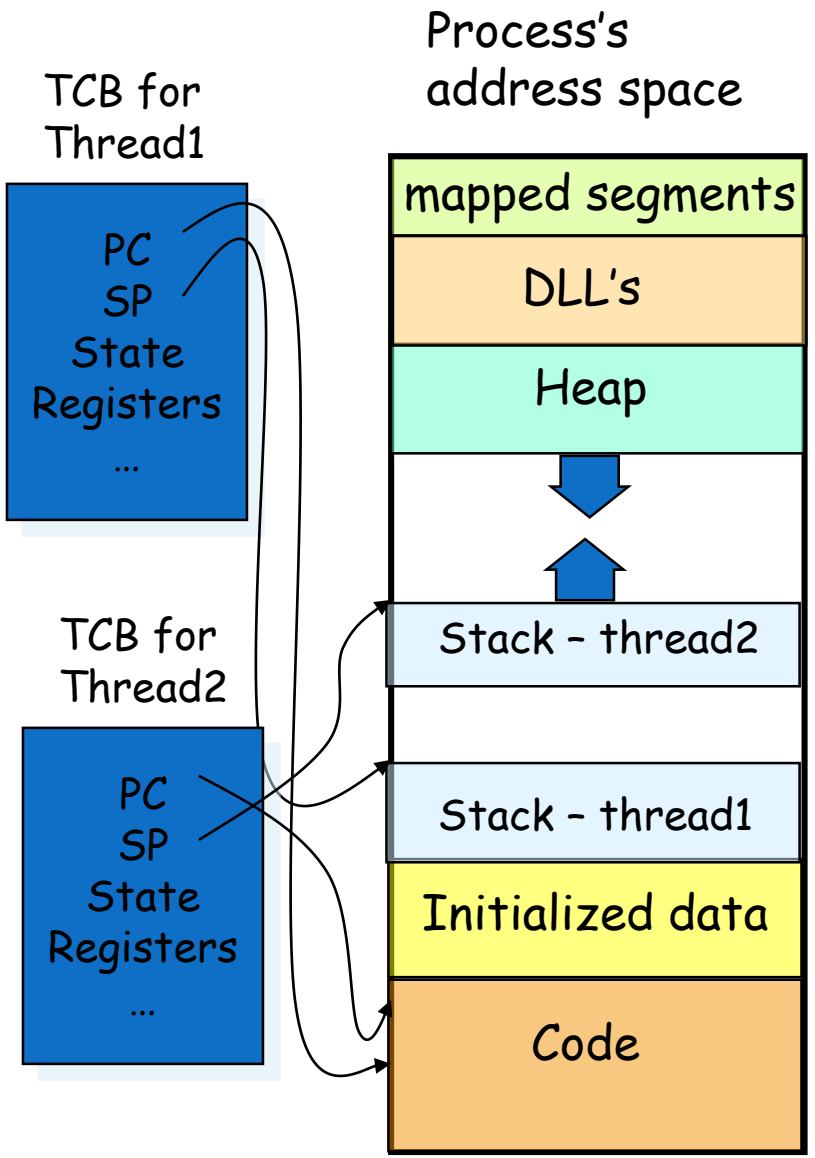
Los procesos definen un espacio de direcciones; los hilos comparten el espacio de direcciones.

El bloque de control de proceso (PCB) contiene información específica del proceso

Propietario, PID, puntero al heap (monton), prioridad, hilo activo, y punteros a la información del hilo

El bloque de control del hilo (TCB-Thread Control Block) contiene información específica del hilo

Puntero a la pila, PC, estado del hilo (ejecutando, …), valores del registro, un puntero al PCB, …



¿Porqué usar hilos?

Algunas razones simples

Evite el bloqueo innecesario

- Un proceso de un solo hilo se **bloqueará** al hacer E/S; en un proceso de múltiples hilos, el sistema operativo puede cambiar la CPU a otro hilo en ese proceso.

Aprovechar el paralelismo

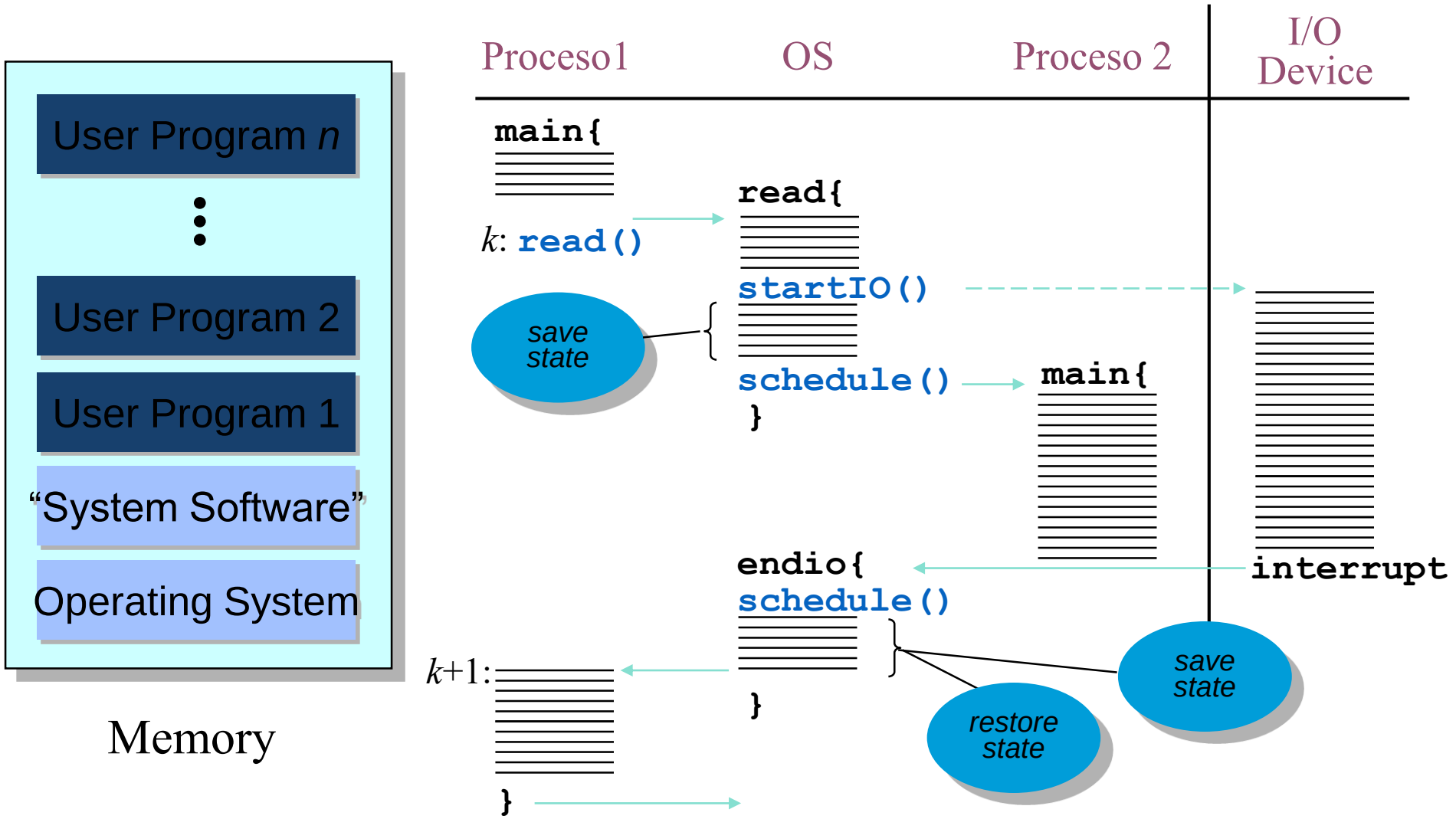
- Los hilos de un proceso de múltiples hilos pueden ser planificados para ejecutarse en paralelo en un procesador multiprocesador o multinúcleo.

Evitar el cambio de proceso (cambio de contexto)

- Estructurar aplicaciones grandes no como una colección de procesos, sino a través de múltiples hilos.

Cambio de contexto de un proceso

Un cambio de contexto ocurre cuando la CPU cambia de un proceso a otro.

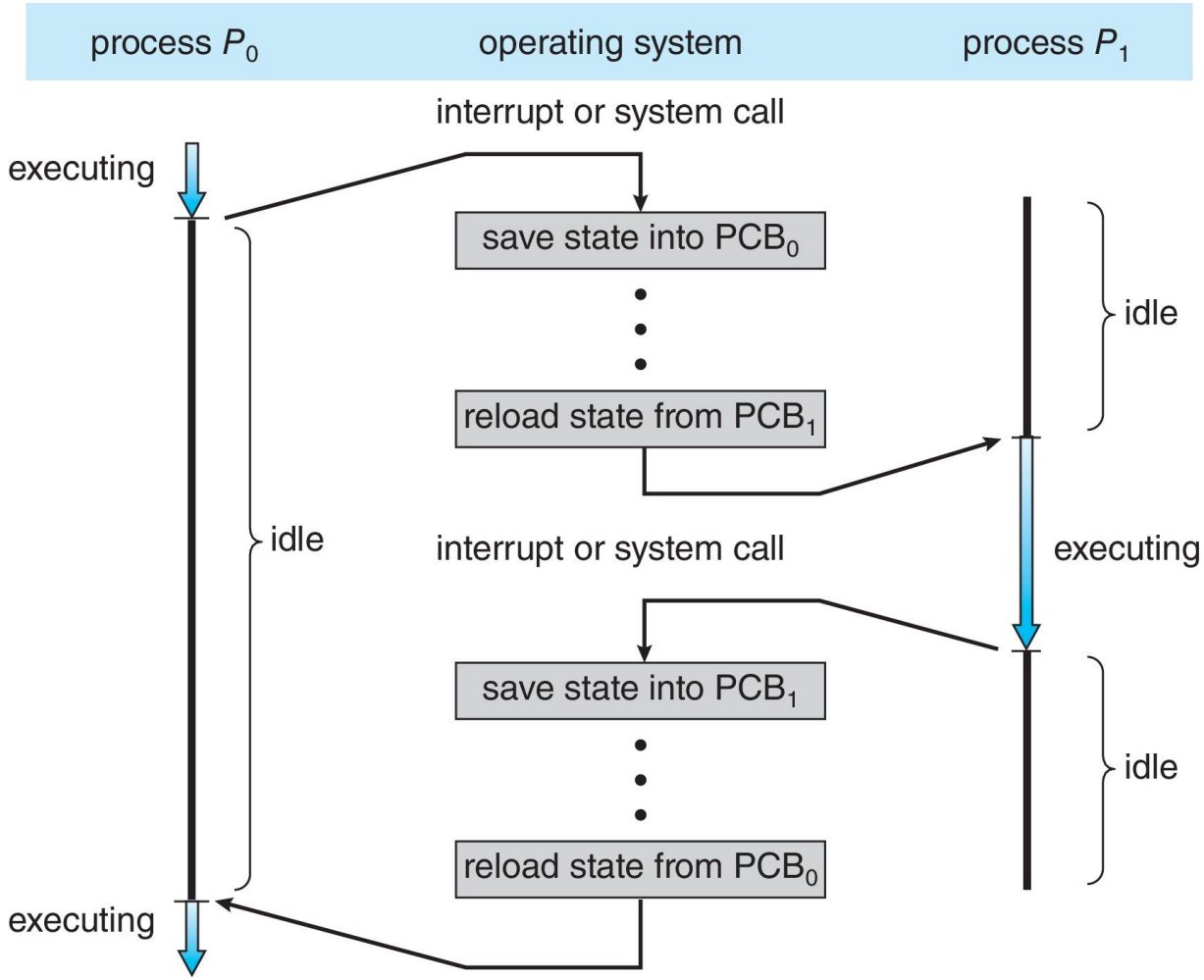


Un proceso es la abstracción del SO para un programa en ejecución. Múltiples procesos pueden ejecutarse concurrentemente en el mismo sistema, y cada proceso parece tener un uso exclusivo del hardware. Por concurrente, queremos decir que las instrucciones de un proceso están intercaladas con las instrucciones de otro proceso. En la mayoría de los sistemas, hay más procesos para ejecutar que CPU para ejecutarlos.

Cambio de contexto de un proceso

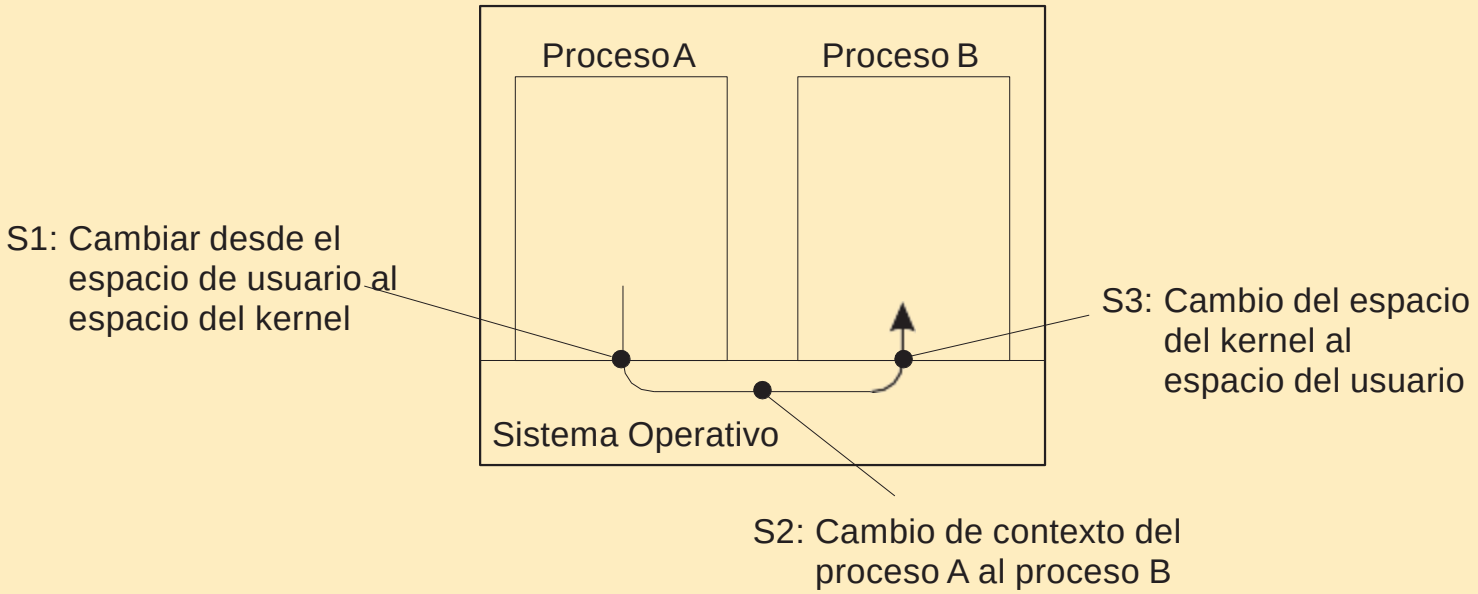
- Cuando la CPU conmuta a otro proceso, el sistema debe **guardar el estado** del proceso anterior y cargar el **estado guardado** para el nuevo proceso mediante un **cambio de contexto**.

- **El Contexto** de un proceso está representado en la PCB
- El tiempo dedicado al cambio de contexto es tiempo desperdiciado, dado que el Sistema no realiza ningún trabajo útil durante la conmutación
  - Cuanto más complejo sea el OS y la PCB → más largo será el cambio de contexto
- El tiempo empleado depende del soporte hardware
  - Algunos sistemas porporcionan multiples conjuntos de registros por CPU → multiples contextos cargados a la vez



Evite el cambio de proceso

Evitar los costosos cambios de contexto



Observaciones

- Los hilos comparten el mismo espacio de direcciones. El cambio de contexto de un hilo puede ser realizado completamente independiente del sistema operativo y puede ser más rápido que el cambio de contexto de un proceso.
- El cambio de contexto de un proceso es generalmente (algo) más costoso ya que implica poner el sistema operativo en el bucle, es decir, atrapar al kernel.
- Crear y destruir hilos es mucho más económico que hacerlo por procesos.

Beneficios del uso de hilos

Capacidad de respuesta – Puede permitir la ejecución continua si se bloquea parte del proceso, lo que es especialmente importante para las interfaces de usuario

Compartición de recursos – Los hilos comparten recursos del proceso, más fácil que la memoria compartida o el paso de mensajes.

Economía – Dado que los hilos comparten recursos del proceso al que pertenecen , es más económico realizar cambios de contexto

Escalibilidad – Las ventajas de configuraciones multihilos pueden ser mayores en aquitecturas multiprocesador.

Hilos de usuario e hilos del kernel

Hilos de usuario - gestión realizada por la biblioteca de hilos a nivel de usuario

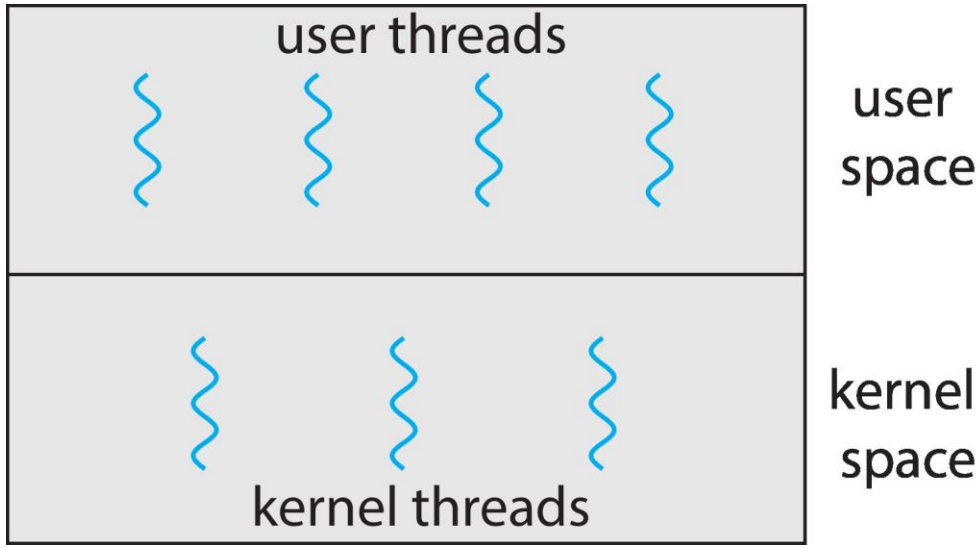
Tres bibliotecas de hilos principales:

- POSIX **Pthreads**
- Windows threads
- Java threads

Hilos del Kernel – Soportados por el kernel

Ejemplos – prácticamente todos los sistemas operativos de propósito general, incluidos:

- Windows
- Linux
- Mac OS X
- iOS
- Android



Modelo multihilos: Muchos-a-Uno

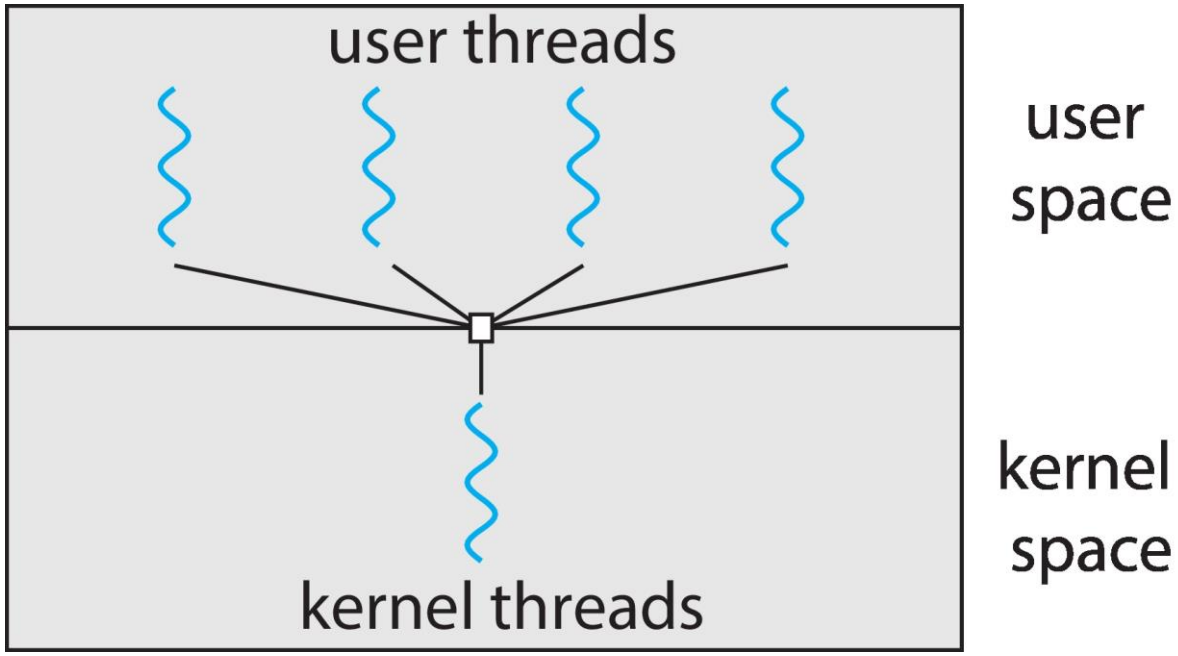
Muchos hilos a nivel de usuario asignados a un solo hilo del kernel

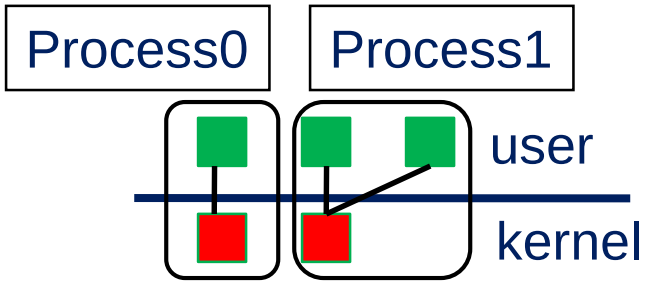
El bloqueo de un hilo hace que todos se bloqueen

Es posible que varios hilos no se ejecuten en paralelo en un sistema de varios núcleos porque solo uno puede estar en el kernel a la vez

Actualmente, pocos sistemas utilizan este modelo. Ejemplos:

- **Solaris Green Threads**
- **GNU Portable Threads**





Modelo multihilos: Uno-a-Uno

Cada hilo de nivel de usuario se asigna al hilo del kernel

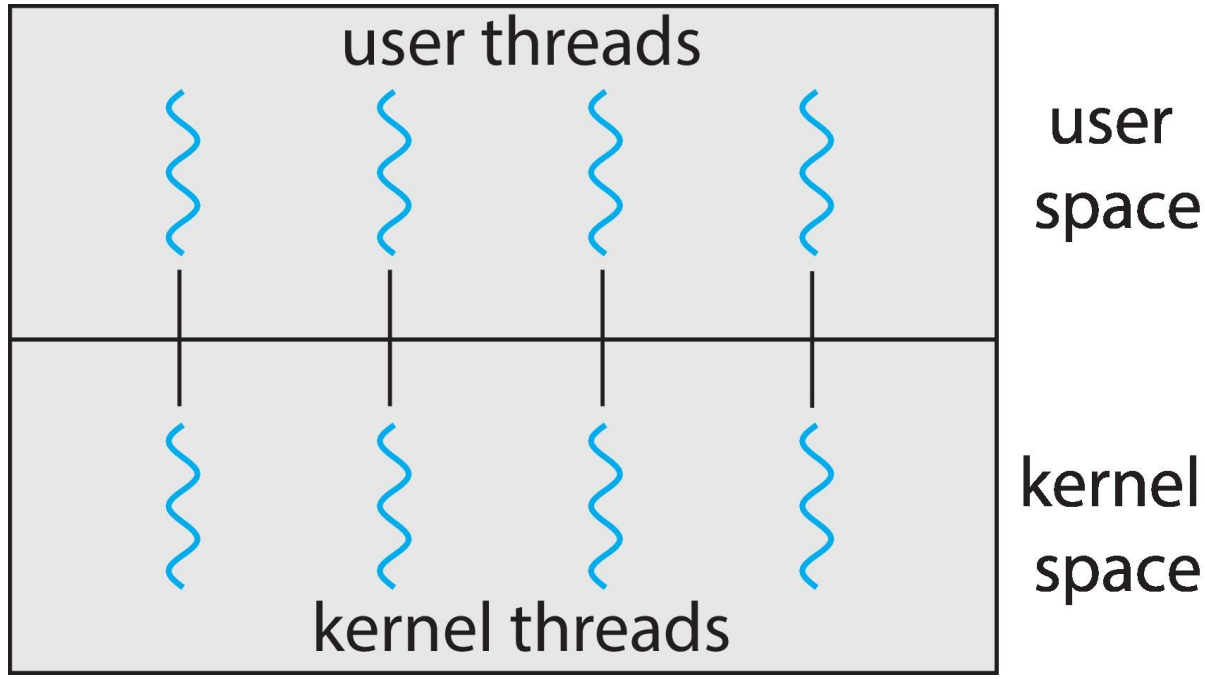
La creación de un hilo a nivel de usuario crea un hilo del kernel

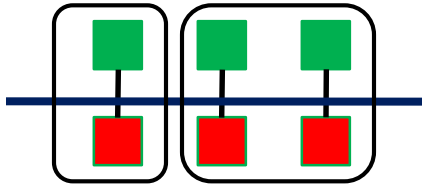
Mayor concurrencia que muchos-a-uno

Número de hilos por proceso a veces restringido por rendimiento

Ejemplos

- Windows
- Linux





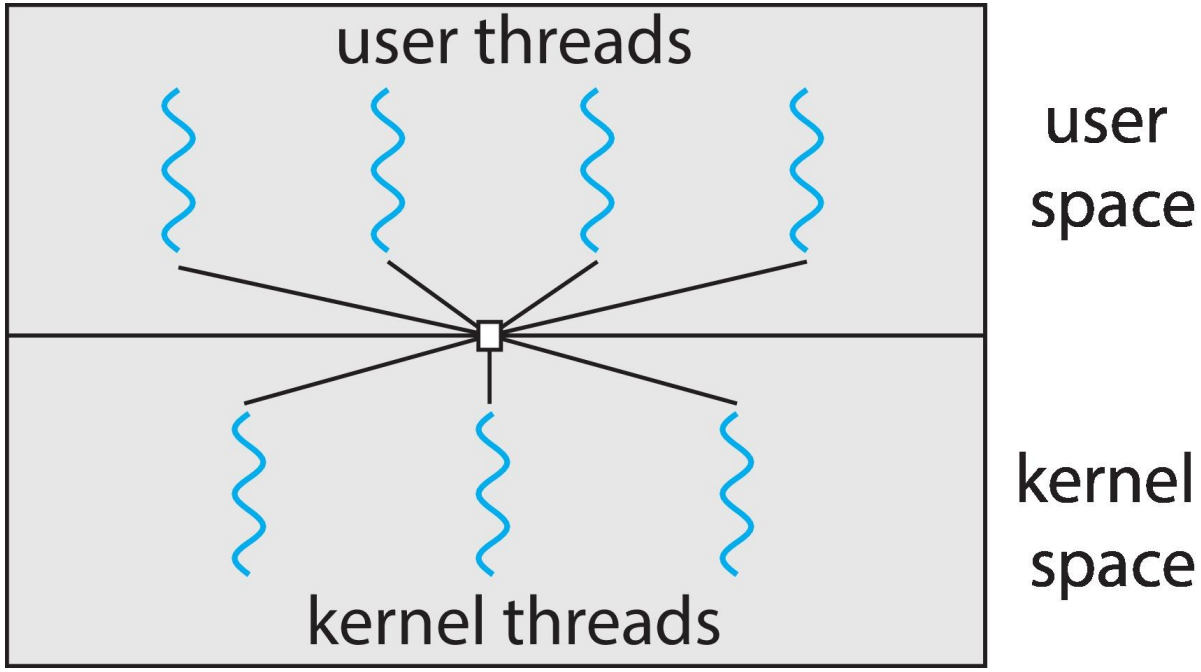
Modelo multihilos: Muchos-a-Muchos

Permite que muchos hilos de nivel de usuario se asignen a muchos hilos del kernel.

Permite que el sistema operativo cree una cantidad suficiente de hilos del kernel

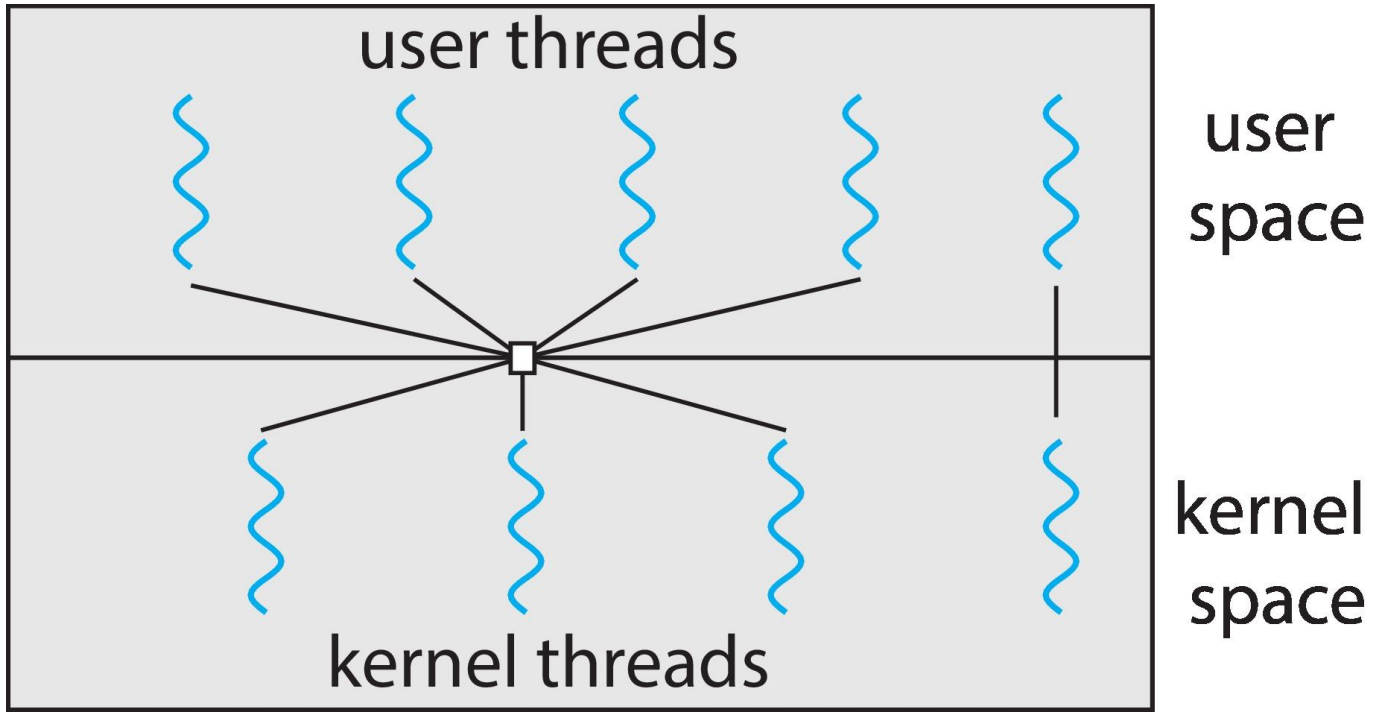
Windows con el paquete ThreadFiber

De lo contrario, no es muy común



Modelo de dos niveles

Similar a M:M, excepto que permite **acoplar** un hilo de usuario a un hilo del kernel.



Ley de Amdahl

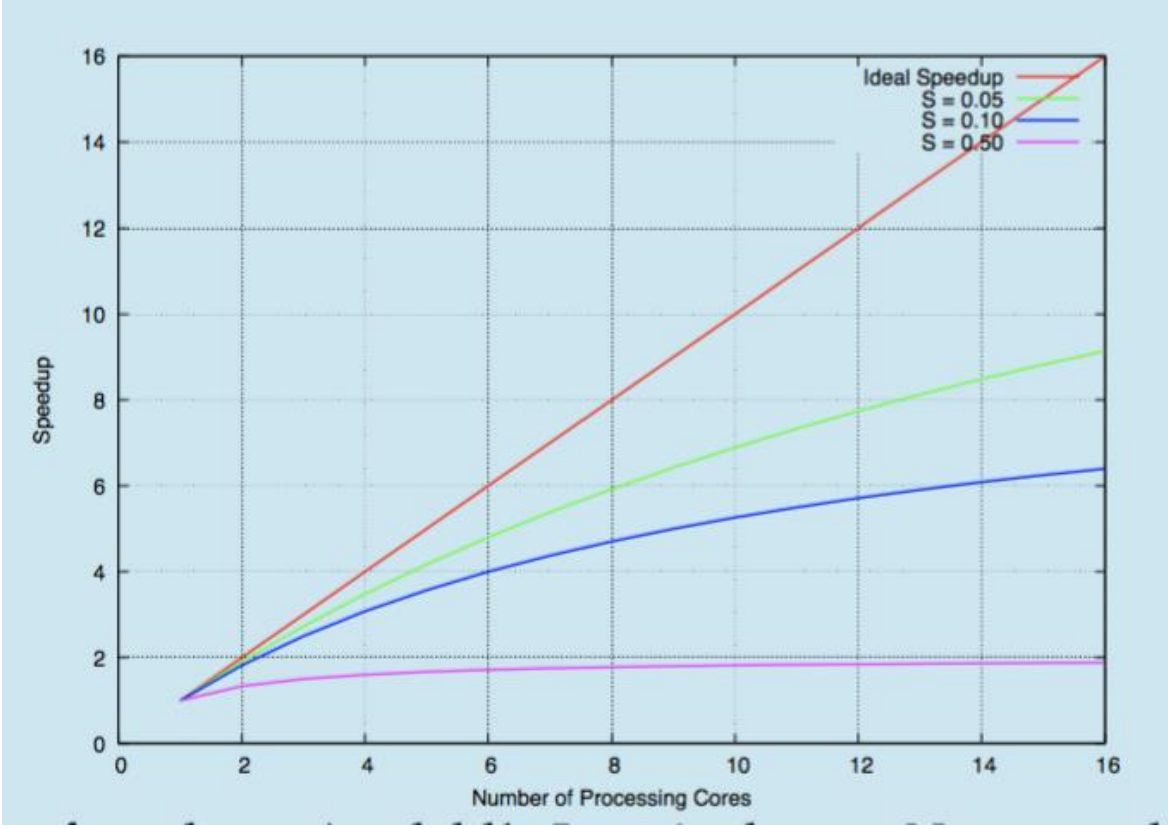
Identifica las ganancias de rendimiento al agregar núcleos adicionales a una aplicación que tiene componentes en serie y en paralelo

S es la parte serial

N núcleos de procesamiento

$$speedup \leq \frac{1}{S + \frac{(1-S)}{N}}$$

- **Es decir, si la aplicación es 75% paralela/ 25% serial, pasar de 1 a 2 núcleos da como resultado una aceleración de 1,6 veces**
- **A medida que N se acerca al infinito, la aceleración se acerca a 1/S**
  La parte en serie de una aplicación tiene un efecto desproporcionado en el rendimiento obtenido al agregar núcleos adicionales
- **Pero, ¿la ley tiene en cuenta los sistemas multinúcleo contemporáneos?**



Biblioteca de hilos

Una biblioteca de hilos proporciona al programador una API para crear y administrar hilos

Dos formas principales de implementar:

- Biblioteca completamente en el espacio del usuario
- Biblioteca de nivel de kernel soportada por el sistema operativo

Pthreads

Puede proporcionarse a nivel de usuario o de kernel

Una API estándar POSIX (IEEE 1003.1c) para la creación y sincronización de hilos

Especificación, no ***implementación***

La API especifica el comportamiento de la biblioteca de hilos, la implementación depende del desarrollo de la biblioteca

Común en los sistemas operativos UNIX (Linux & Mac OS X)

Ejemplo Pthreads

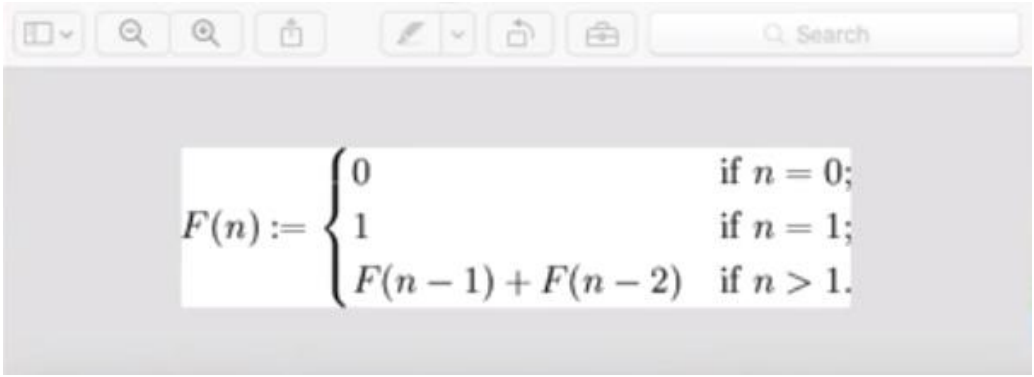
```c
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
int sum; /* los hilos comparten esta variable */
void *runner(void *param); /* los hilos llaman a esta fencion */
int main(int argc, char *argv[])
{
pthread_t tid; /* el identificador del hilo */
pthread_attr_t attr; /* conjunto de atributos del hilo */
/* configura los atributos del hilo por defecto */
pthread_attr_init(&attr);
/* crea el hilo */
pthread_create(&tid, &attr, runner, argv[1]);
/* esperar que el hilo termine */
pthread_join(tid,NULL);
printf("sum = %d",sum);
}
/* El hilo inicia su ejecución en esta funcion */
void *runner(void *param)
{
int i, upper = atoi(param);
sum = 0;
for (i = 1; i <= upper; i++)
sum += i;
pthread_exit(0);
}
```

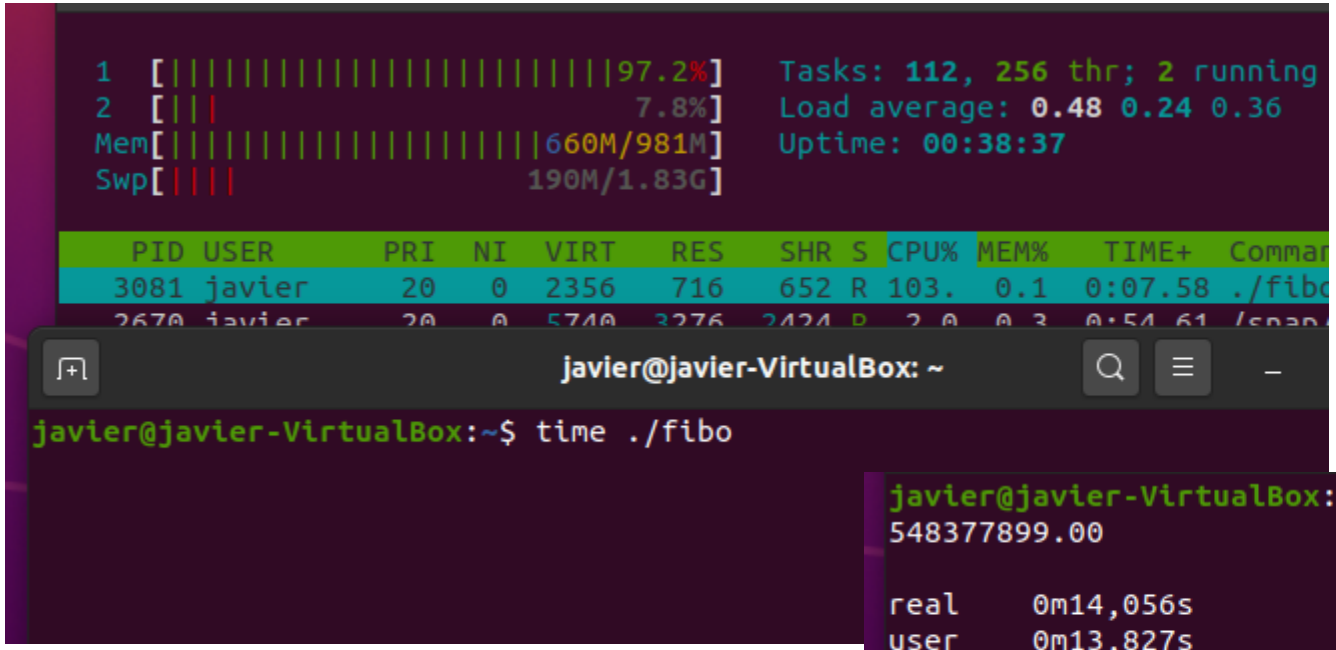
```c
#define NUM_THREADS 10

/* an array of threads to be joined upon */
pthread_t workers[NUM_THREADS];

for (int i = 0; i < NUM_THREADS; i++)
   pthread_join(workers[i], NULL);
```

```
javier@javier-VirtualBox:~$ gcc -pthread prueba1.c -o prueba1
javier@javier-VirtualBox:~$ time ./prueba1 1000
sum = 500500
real    0m0,004s
user    0m0,000s
sys     0m0,004s
javier@javier-VirtualBox:~$
```

Ejemplo Fibonacci (Versión secuencial)

```c
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <time.h>

#define initrandomico() srand(time(NULL))
#define randomico(n) rand() % n

#define MAXITER 1000000

int fibonacci(int n) {
if (n==0)
return 0;
else if (n==1)
return 1;
else
return fibonacci(n-1) + fibonacci(n-2);
}
int main (int argc, char** argv){
int i;
double sum;

sum=0;
initrandomico();
for (i=0;i< MAXITER; i++){
sum+=fibonacci(randomico(20));
}
printf("%.2lf\n",sum);
return 0;
}
```






```
javier@javier-VirtualBox:~$ time ./fibo
548377899.00

real    0m14,056s
user    0m13,827s
sys     0m0,079s
javier@javier-VirtualBox:~$
```

Ejemplo Fibonacci con Pthreads

```c
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <time.h>
#include <pthread.h>
#define initrandomico() srand(time(NULL))
#define randomico(n) rand() % n
#define MAXITER 1000000
#define MAXTHREADS 8
double sumparcial[MAXTHREADS];
int fibonacci(int n) {
if (n==0) return 0;
else if (n==1) return 1;
else return fibonacci(n-1) + fibonacci(n-2);
}
void* myfunction(void *data){
int id = (int) data;
int iterxthread = MAXITER/MAXTHREADS;
int i;
printf("Hilo %d interaciones %d\n",id,iterxthread);
sumparcial[id]=0;
for(i=0; i< iterxthread; i++) sumparcial[id] += (double)fibonacci(randomico(20));
return NULL;
}
int main (int argc, char** argv){
int i;
double sum;
pthread_t threadid[MAXTHREADS];
sum=0;
initrandomico();
for (i=0;i< MAXTHREADS; i++) pthread_create(&threadid[i],NULL, myfunction, (void*)i);
for (i=0;i< MAXTHREADS; i++){
pthread_join(threadid[i],NULL);
sum+=sumparcial[i];
}
printf("%.2lf\n",sum);
return 0;
}
```

$$F(n) := \begin{cases} 0 & \text{if } n = 0; \\ 1 & \text{if } n = 1; \\ F(n-1) + F(n-2) & \text{if } n > 1. \end{cases}$$

```
javier@javier-VirtualBox: ~
javier@javier-VirtualBox:~$ time ./fibopthread
Hilo 0 interaciones 125000
Hilo 5 interaciones 125000
Hilo 1 interaciones 125000
Hilo 2 interaciones 125000
Hilo 4 interaciones 125000
Hilo 3 interaciones 125000
Hilo 7 interaciones 125000
Hilo 6 interaciones 125000
```

```
javier@javier-VirtualBox:~$ time ./fibopthread
Hilo 0 interaciones 125000
Hilo 5 interaciones 125000
Hilo 1 interaciones 125000
Hilo 2 interaciones 125000
Hilo 4 interaciones 125000
Hilo 3 interaciones 125000
Hilo 7 interaciones 125000
Hilo 6 interaciones 125000
548082535.00

real    0m9,521s
user    0m17,919s
sys     0m0,263s
```

Hilos de Java

- Los hilos de Java son gestionados por la JVM.
- Normalmente se implementa utilizando el modelo de hilos proporcionado por el sistema operativo subyacente.
- Los hilos de Java pueden ser creados por:
  - Extendiendo la Clase Threads
  - Implementando la interfaz Runnable

```
public interface Runnable
{
    public abstract void run();
}
```

  - La práctica estándar es implementar la interfaz Runnable

Hilos de Java

Implementación de la interfaz Runnable:

```
class Task implements Runnable
{
   public void run() {
      System.out.println("I am a thread.");
   }
}
```

Creando un hilo:

```
Thread worker = new Thread(new Task());
worker.start();
```

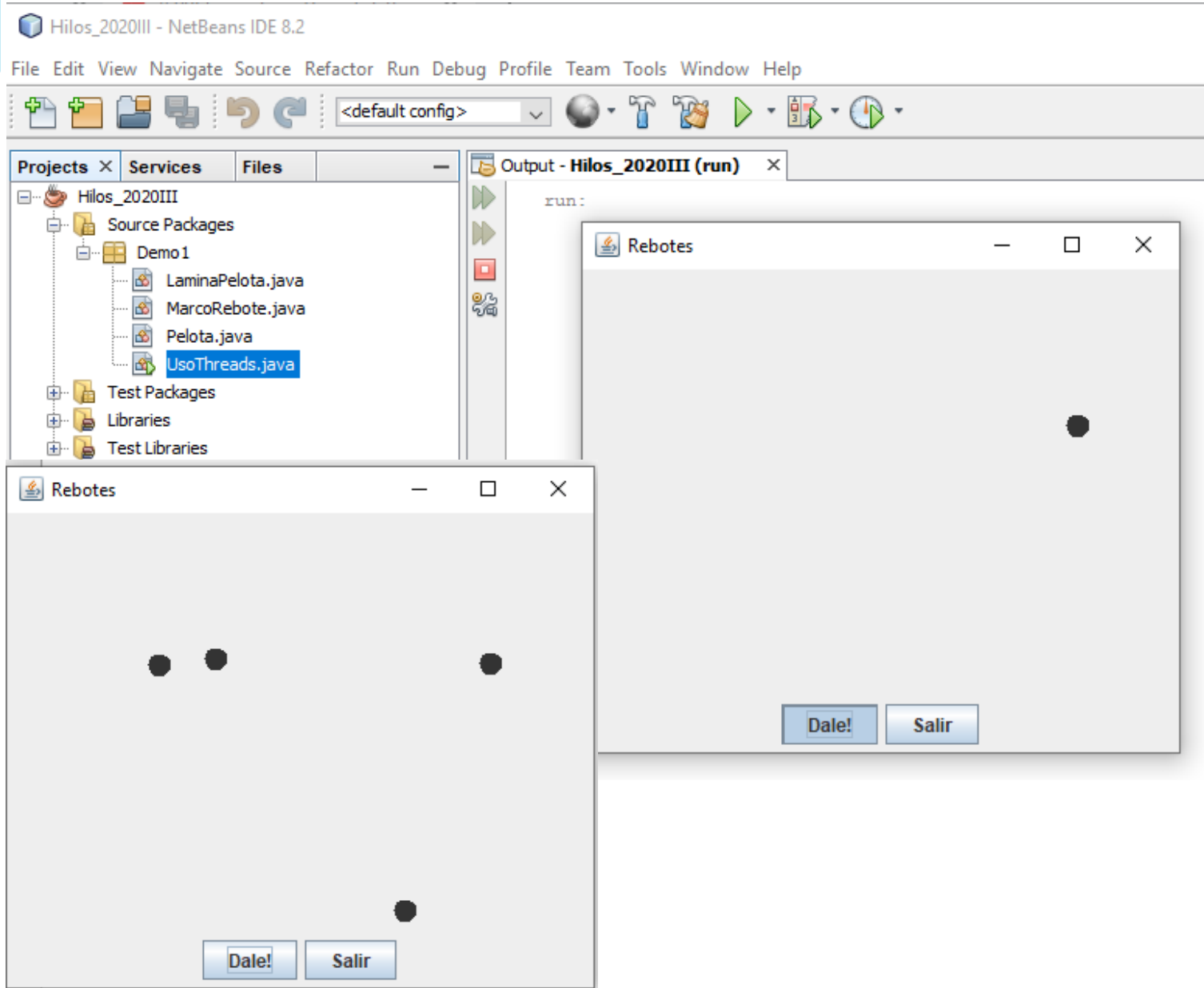
Esperando un hilo:

```
try {
   worker.join();
}
catch (InterruptedException ie) { }
```

Programación de Hilos en Java

Crear hilos de ejecución

1. Crear clase que implemente la interfaz runnable (método run())
2. Escribir el código de la tarea dentro del método run
3. Instanciar la clase creada y almacenar la instancia en variable de tipo Runnable
4. Crear instancia de la clase Thread pasando como parámetro al constructor de Thread el objeto Runnable anterior
5. Poner en marcha el hilo de ejecución con el método start() de la clase Thread



OpenMP

Conjunto de directivas del compilador y una API para C, C ++, FORTRAN

Proporciona soporte para programación paralela en entornos de memoria compartida

Identifica **regiones paralelas** – bloques de código que pueden ejecutarse en paralelo

`#pragma omp parallel`

Crea tantos hilos como núcleos

```
#include <omp.h>
#include <stdio.h>

int main(int argc, char *argv[])
{
  /* sequential code */

  #pragma omp parallel
  {
    printf("I am a parallel region.");
  }

  /* sequential code */

  return 0;
}
```

OpenMP – Ejemplos

- **Ejecutar el siguiente programa hello1.c**

```c
#include <stdio.h>
#include <omp.h>

int main(int argc, char** argv){
#pragma omp parallel
{
int id=omp_get_thread_num();
printf("hello %d",id);
printf("world %d\n",id);
}
return 0;
}
```

```
javier@javier-VirtualBox:~$ time ./hello1
hello 0world 0
hello 1world 1

real    0m0,035s
user    0m0,006s
sys     0m0,000s
javier@javier-VirtualBox:~$
```

- **Ejecutar el siguiente programa hello2.c**

```c
#include <stdio.h>
#include <omp.h>

int main(int argc, char** argv){
#pragma omp parallel num_threads(8)
{
int id=omp_get_thread_num();
printf("hello %d",id);
printf("world %d\n",id);
}
return 0;
}
```

```
javier@javier-VirtualBox:~$ gcc -fopenmp hello2.c -o hello2
javier@javier-VirtualBox:~$ time ./hello2
hello 1hello 0hello 4world 4
hello 2world 2
hello 3world 3
world 0
world 1
hello 5world 5
hello 6world 6
hello 7world 7

real    0m0,009s
user    0m0,008s
sys     0m0,001s
javier@javier-VirtualBox:~$
```

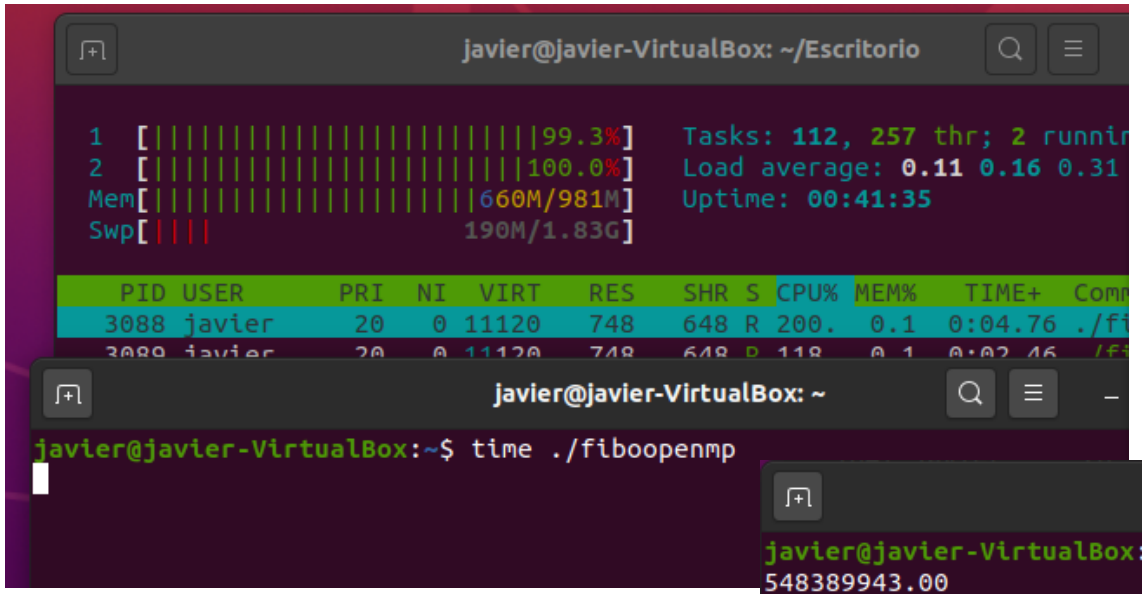
Ejecuta el bucle for en paralelo

```c
#pragma omp parallel for
for (i = 0; i < N; i++) {
   c[i] = a[i] + b[i];
}
```

Ejemplo Fibonacci con OpenMP

```c
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <time.h>

#define initrandomico() srand(time(NULL))
#define randomico(n) rand() % n

#define MAXITER 1000000

int fibonacci(int n) {
if (n==0)
return 0;
else if (n==1)
return 1;
else
return fibonacci(n-1) + fibonacci(n-2);
}
int main (int argc, char** argv){
int i;
double sum;

sum=0;
initrandomico();
#pragma omp parallel for private(i) reduction(+:sum)
for (i=0;i< MAXITER; i++){
sum+=fibonacci(randomico(20));
}
printf("%.2lf\n",sum);
return 0;
}
```

$$F(n) := \begin{cases} 0 & \text{if } n = 0; \\ 1 & \text{if } n = 1; \\ F(n-1) + F(n-2) & \text{if } n > 1. \end{cases}$$



Universidad Nacional Mayor de San Marcos
Universidad del Perú. Decana de América

Guía de Laboratorio

28/08/2025 – Escuela Profesional de Ingeniería de Software

Universidad Nacional Mayor de San Marcos

Universidad del Perú. Decana de América

Resumen

- Un hilo representa una unidad básica de utilización de la CPU, y los hilos que pertenecen al mismo proceso comparten muchos de los recursos del proceso, incluidos el código y los datos.

- Las aplicaciones multihilos tienen cuatro ventajas principales: (1) capacidad de respuesta, (2) uso compartido de recursos, (3) economía y (4) escalabilidad.

- Existe concurrencia cuando varios hilos están progresando, mientras que existe paralelismo cuando varios hilos están progresando simultáneamente. En un sistema con una sola CPU, solo es posible la concurrencia; el paralelismo requiere un sistema multinúcleo que proporcione varias CPU.

Resumen

- Existen varios desafíos en el diseño de aplicaciones multihilos. Entre ellos se incluyen la división y el equilibrio del trabajo, la división de los datos entre los diferentes hilos y la identificación de las dependencias de los datos. Por último, los programas multihilos son especialmente difíciles de probar y depurar.

- El paralelismo de datos distribuye subconjuntos de los mismos datos entre diferentes núcleos informáticos y realiza la misma operación en cada núcleo. El paralelismo de tareas no distribuye datos sino tareas entre varios núcleos. Cada tarea ejecuta una operación única.

- Las aplicaciones de usuario crean hilos a nivel de usuario, que en última instancia deben asignarse a hilos del núcleo para ejecutarse en una CPU. El modelo de varios a uno asigna muchos hilos a nivel de usuario a un hilo del núcleo. Otros enfoques incluyen los modelos de uno a uno y de varios a varios.

Resumen

- Una biblioteca de hilos proporciona una API para crear y administrar hilos. Tres bibliotecas de hilos comunes incluyen Pthreads y los hilos de Java. Pthreads está disponible para sistemas compatibles con POSIX, como UNIX, Linux y macOS. Los hilos de Java se ejecutarán en cualquier sistema que admita una máquina virtual Java.

Universidad Nacional Mayor de San Marcos

Universidad del Perú. Decana de América