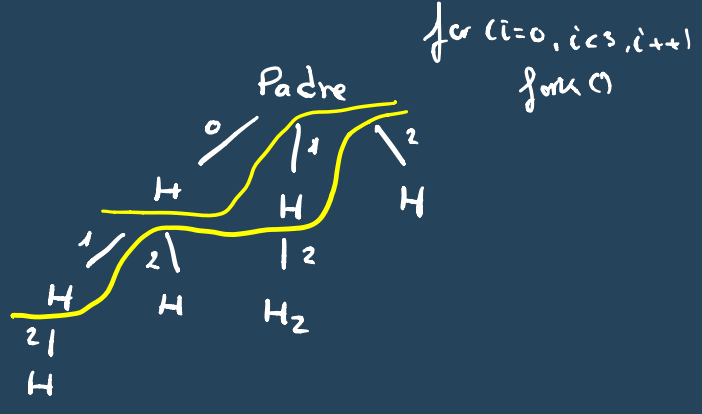


SISTEMAS OPERATIVOS: PROCESOS

Hilos y Procesos

ADVERTENCIA

- Este material es un simple guión de la clase: no son los apuntes de la asignatura.
- El conocimiento exclusivo de este material no garantiza que el alumno pueda alcanzar los objetivos de la asignatura.
- Se recomienda que el alumno utilice los materiales complementarios propuestos.

Contenido

- **Concepto de hilo.**
- Modelos de hilos.
- Aspectos de diseño.
- Hilos en pthreads.
- Planificación de hilos

Aplicaciones con tareas concurrente

Hilos de ejecución paralelos en un proceso.

- Un proceso incluye un único hilo de ejecución.

Crear hilos permite que compartan memoria
Aprovecha mejor la rodaja de tiempo, menos cambios de contexto

Un thread está ligado a un proceso, pero no funciona de manera independiente.

- Diseño de aplicación con varias tareas concurrentes:
  - Un proceso receptor de peticiones y lanzar un proceso por petición.
  - Un proceso receptor y un conjunto fijo de procesos de tratamiento de peticiones.

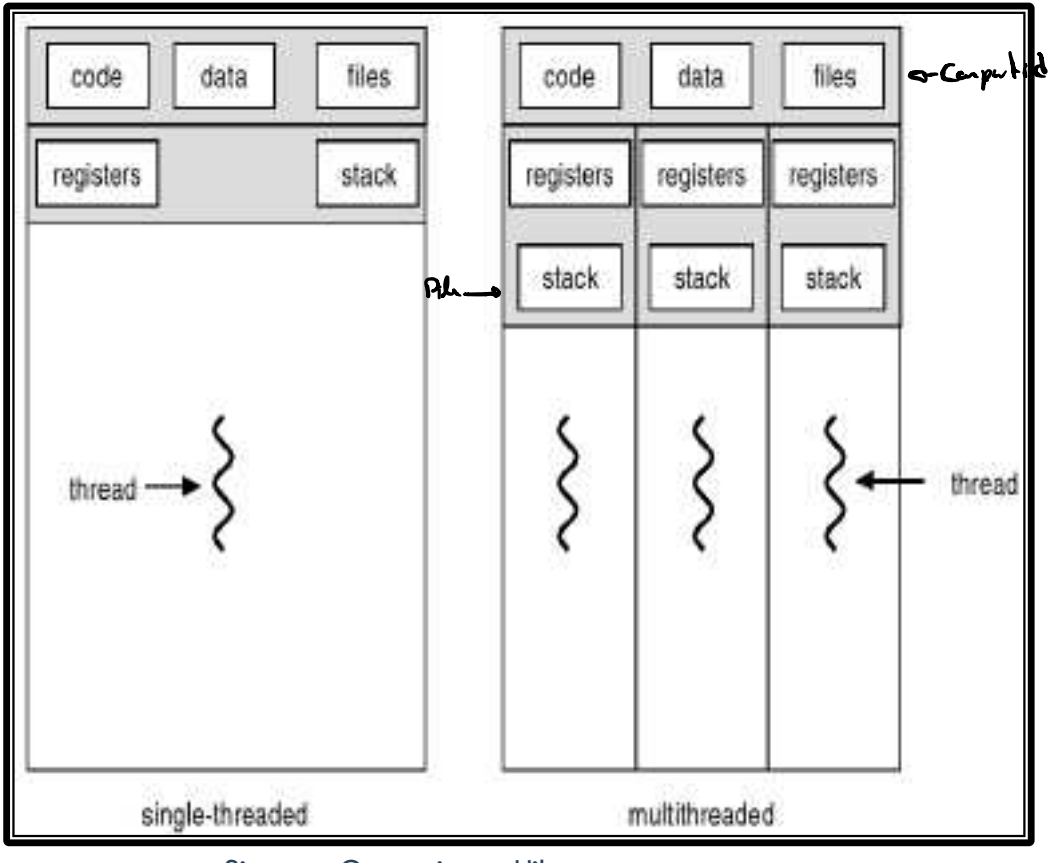
Rendimiento de aplicaciones concurrentes

Resuelve los problemas del fork.

Se creaba para reducir:

- Consumo de tiempo en la creación y terminación de procesos.
- Consume de tiempo en cambios de contexto.
- Problemas en la compartición de recursos.

Hilo, ~~Proceso ligero o~~ *Thread*

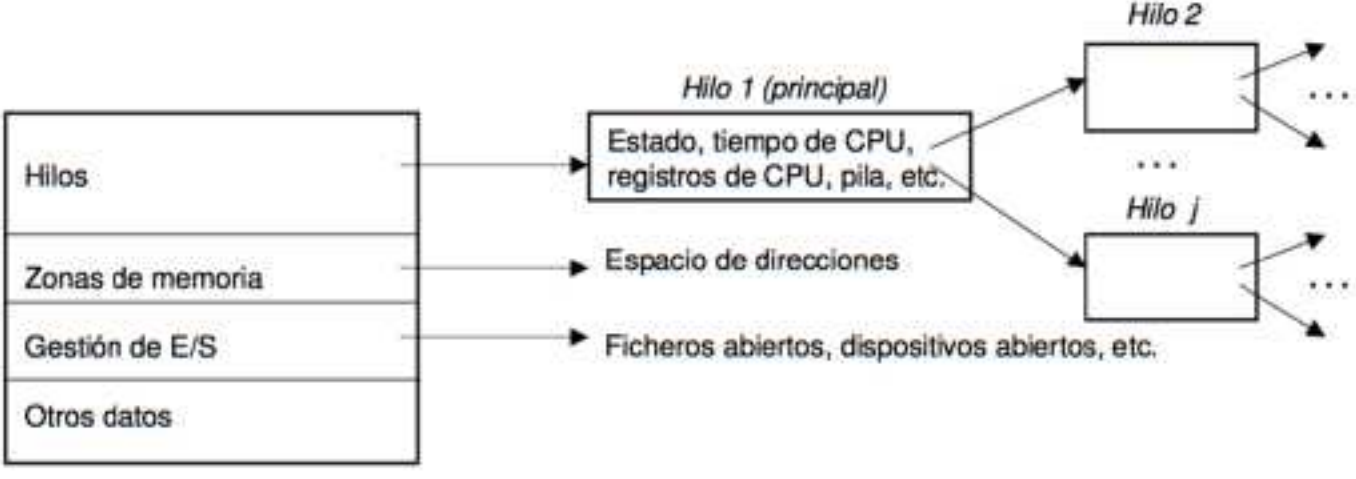


Hilos

Cada uno necesita una pila para almacenar sus datos y un minicontexto.
La región de datos y ficheros la comparten
En el BCP hay una tabla de threads, donde se almacena la info de la pila y su contexto
ThreadTable

- La mayoría de los modernos SO proporcionan procesos con múltiples secuencias o hilos de control en su interior.
- Se considera una unidad básica de utilización de la CPU.
- Cada uno comprende:
  - Identificador de thread
  - Contador de programa
  - Conjunto de registros
  - Pila
- Comparten con el resto de hilos del proceso:
  - Mapa de memoria (sección de código, sección de datos, shmem)
  - Ficheros abiertos
  - Señales, semáforos y temporizadores.

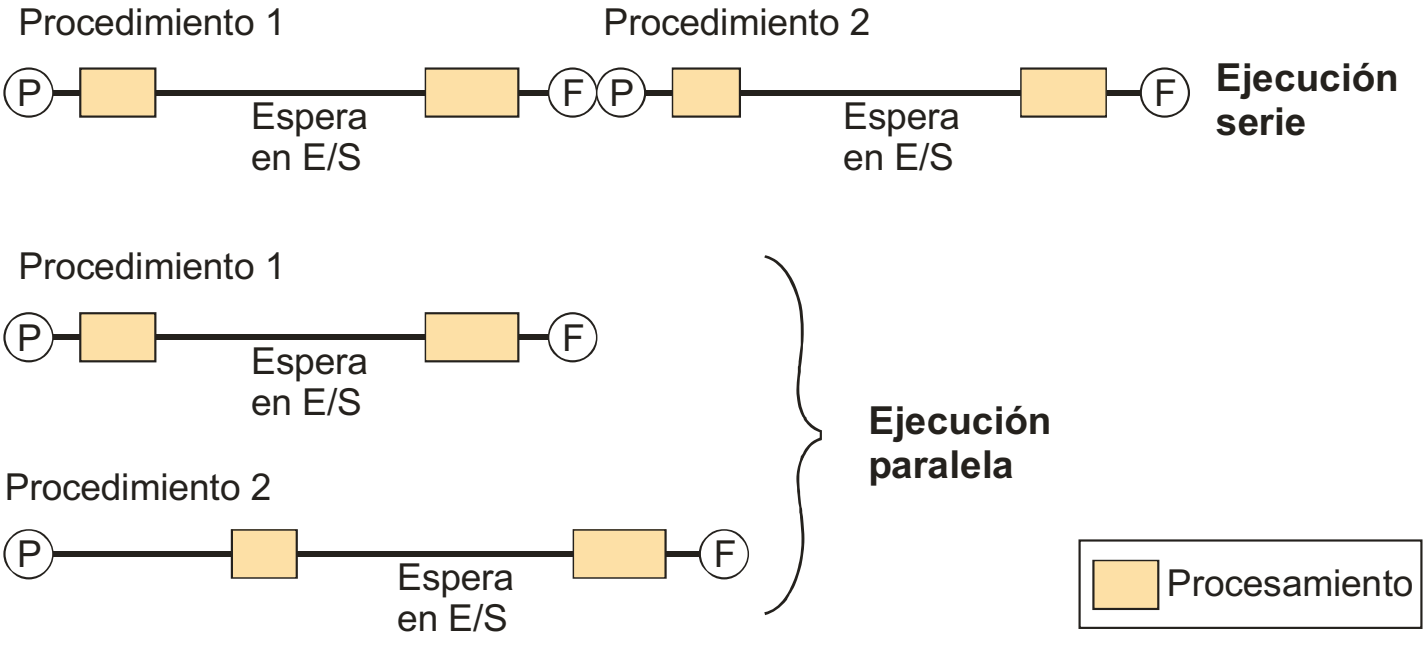
BCP de proceso con hilos



Beneficios

- Capacidad de respuesta.
  - Mayor interactividad al separar las interacciones con el usuario de las tareas de procesamiento en distintos hilos.
- Compartición de recursos.
  - Los hilos comparten la mayor parte de los recursos de forma automática.
- Economía de recursos.
  - Crear un proceso consume mucho más tiempo que crear un hilo (Ejemplo: en Solaris relación 30 a 1).
- Utilización sobre arquitecturas multiprocesador.
  - Mayor nivel de concurrencia asignando distintos hilos a distintos procesadores.
  - La mayoría de los sistemas operativos modernos usan el hilo como unidad de planificación.

Los threads permiten paralelizar la ejecución de una aplicación



Soporte de hilos

Paralelo

Espacio de usuario

ULT – User Level Threads

- Implementados en forma de biblioteca de funciones en espacio de usuario.
- El kernel no tiene conocimiento sobre ellos, no ofrece soporte de ningún tipo.
- Es mucho más rápido pero presentan algunos problemas → Llamadas al sistema bloqueantes.

Solo lo ve el que crea el thread

Se puede pasar un hilo de un proceso al Kernel para que lo gestione, deja de depender del proceso

Tiene otro id y gestión propia por el planificador, no se bloquea ni depende del proceso.

Lo ve el proceso

Espacio de núcleo

KLT – Kernel Level Threads

- El kernel se ocupa de crearlos, planificarlos y destruirlos.
- Es un poco más lento ya que hacemos participar al kernel y esto supone un cambio de modo de ejecución.
- En llamadas al sistema bloqueantes sólo se bloquea el thread implicado.
- En sistemas SMP, varios threads pueden ejecutarse a la vez.
- No hay código de soporte para thread en las aplicaciones.
- El kernel también puede usar threads para llevar a cabo sus funciones.

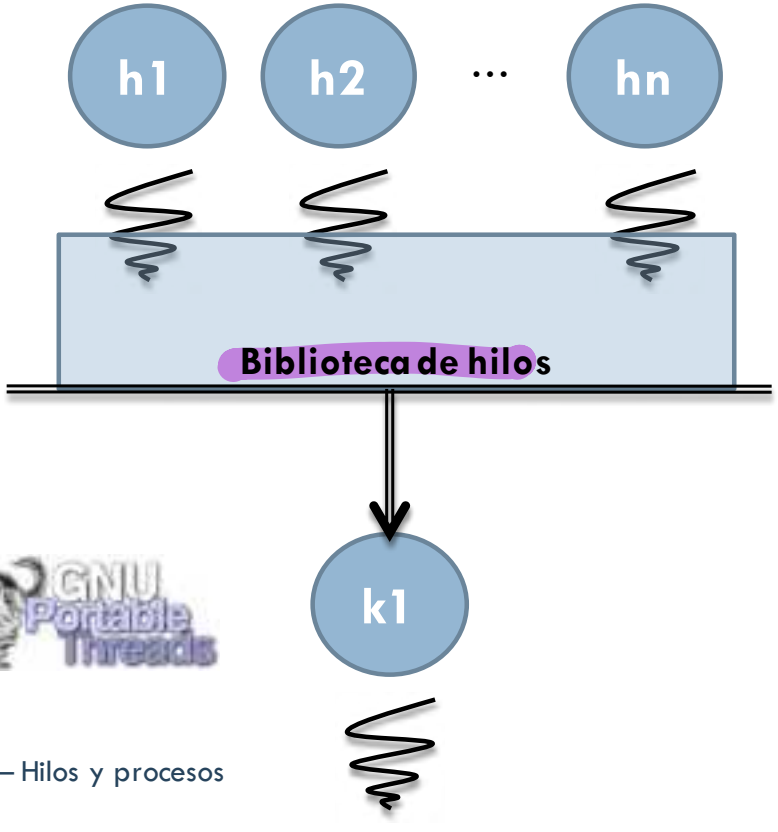
Hay un número máximo de threads que pueden estar ejecutando.

Contenido

- Concepto de hilo.
- **Modelos de hilos.**
- Aspectos de diseño.
- Hilos en pthreads.
- Planificación de hilos

Modelos de múltiples hilos: Muchos a uno

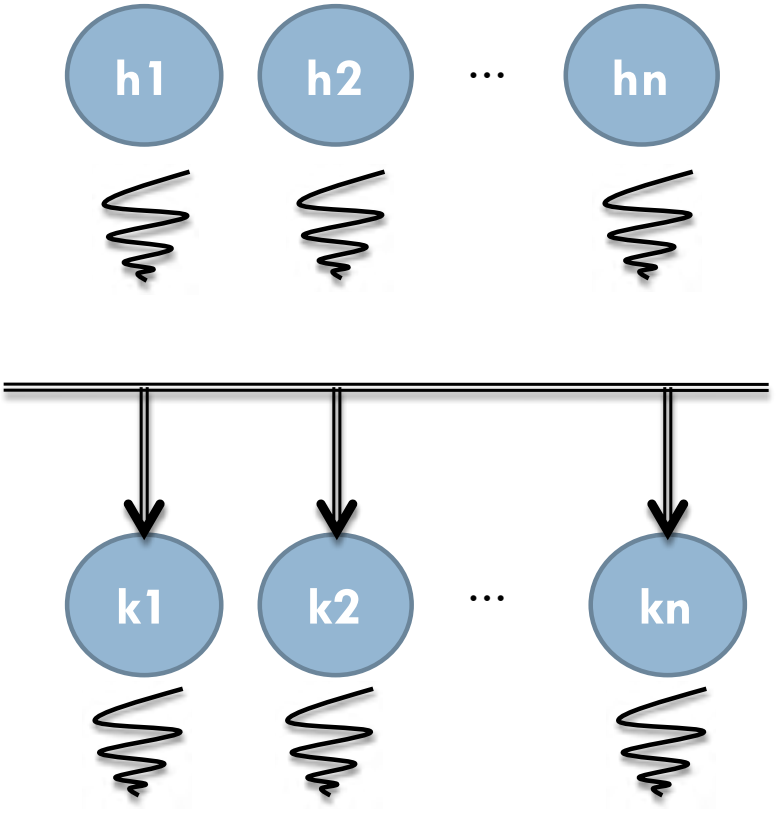
- Hace corresponder múltiples hilos de usuario a un único hilo del núcleo.
- Biblioteca de hilos en espacio de usuario.
- Llamada bloqueante:
  - Se bloquean todos los hilos.
- En multiprocesadores no se pueden ejecutar varios hilos a la vez.



Modelo de múltiples hilos: Uno a uno

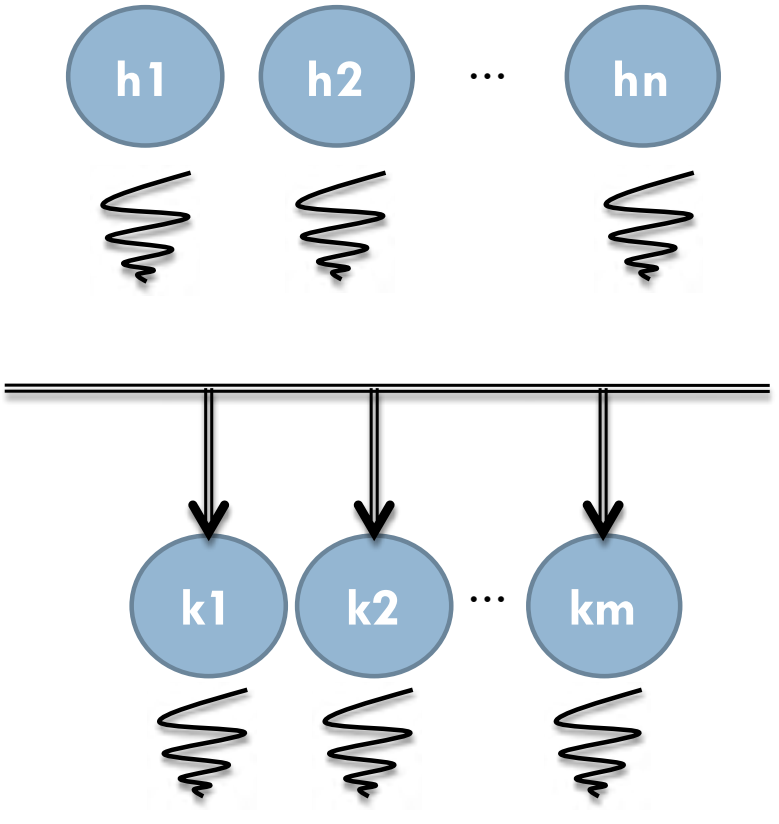
kernel

- Hace corresponder un hilo del kernel a cada hilo de usuario.
- La mayoría de las implementaciones restringen el número de hilos que se pueden crear.
- Ejemplos:
  - Linux 2.6.
  - Windows.
  - Solaris 9.



Modelos de múltiples hilos: Muchos a muchos

- Este modelo multiplexa los threads de usuario en un número determinado de threads en el kernel.
- El núcleo del sistema operativo se complica mucho.
- Ejemplos:
  - Solaris (versiones anteriores a 9).
  - HP-UX.
  - IRIX.



Contenido

- Concepto de hilo.
- Modelos de hilos.
- **Aspectos de diseño.**
- Planificación de hilos
- Hilos en pthreads.

Llamadas a fork y exec

- En los sistemas tipo UNIX ¿Qué se debe hacer si se llama a fork desde un hilo? No tiene sentido
  - Duplicar el proceso con todos sus hilos.
    - Apropiado si no se va a llamar luego a exec para sustituir la imagen del proceso. Se quita todo y los hilos también.
  - Duplicar el proceso solo con el hilo que llama a fork.
    - Más eficiente si se va a llamar a exec y se van a cancelar todos los hilos.
- Solución en Linux: Dos versiones de fork.

Cancelación de hilos

- Situación en la que un hilo notifica a otros que deben terminar.
- Opciones:
  - Cancelación asíncrona: Se fuerza la terminación inmediata del hilo.
    - Problemas con los recursos asignados al hilo.
  - Cancelación diferida: El hilo comprueba periódicamente si debe terminar.
    - Preferible.

Hilos y procesamiento de solicitudes

- Las aplicaciones que reciben peticiones y las procesan pueden usar hilos para el tratamiento.

- Pero:
  - El tiempo de creación/destrucción del hilo supone un retraso (aunque sea menor que el de creación/destrucción de un proceso).
  - No se establece un límite en el número de hilos concurrentes.
  - Si llega una avalancha de peticiones se pueden agotar los recursos del sistema.

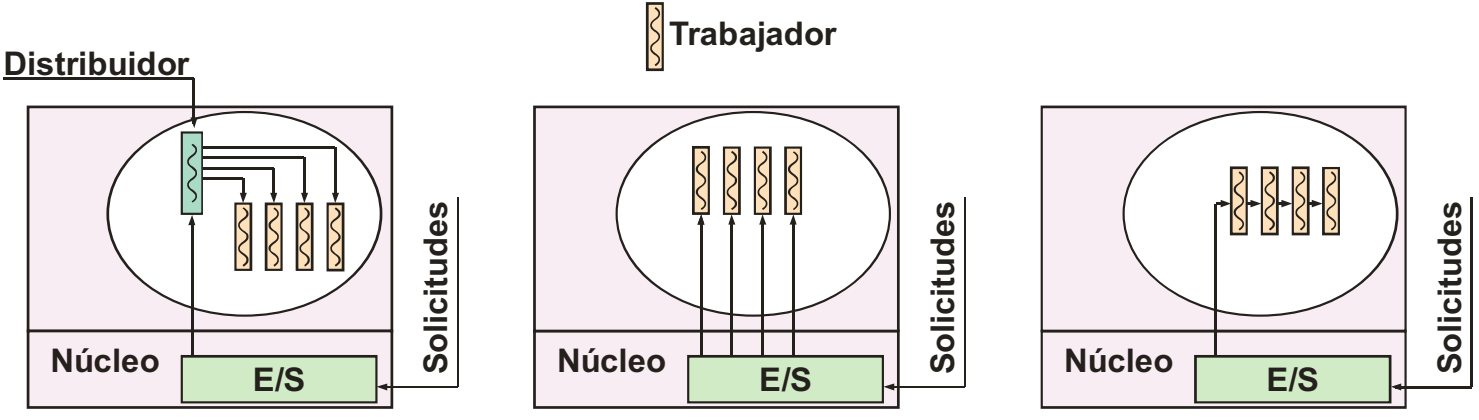
Thread Pools o Conjuntos de Hilos

- Se crea un conjunto de hilos que quedan en espera a que lleguen peticiones.

Cuando no puede procesar la cola de peticiones, se da denegación de servicio.

- Ventajas:
  - Se minimiza el retardo: El hilo ya existe.
  - Se mantiene un límite sobre el número de hilos concurrentes.

Arquitecturas software basadas en threads



Contenido

- Concepto de hilo.
- Modelos de hilos.
- Aspectos de diseño.
- **Hilos en pthreads.**
- Planificación de hilos

Creación de hilos

- `int pthread_create(pthread_t *thread, const pthread_attr_t *attr, void *(*func)(void *), void *arg)`
  - Crea un hilo e inicia su ejecución.
  - **thread**: Se debe pasar la dirección de una variable del tipo **pthread_t** que se usa como manejador del hilo.
  - **attr**: Se debe pasar la dirección de una estructura con los atributos del hilo. Se puede pasar NULL para usar atributos por defecto.
  - **func**: Función con el código de ejecución del hilo.
  - **arg**: Puntero al parámetro del hilo. Solamente se puede pasar un parámetro.
- `pthread_t pthread_self(void)`
  - Devuelve el identificador del thread que ejecuta la llamada.

Espera y terminación

- `int pthread_join(pthread_t thread, void **value)`
  - El hilo que invoca la función se espera hasta que el hilo cuyo manejador se especifique haya terminado.
  - **thread**: Manejador de del hilo al que hay que esperar.
  - **value**: Valor de terminación del hilo.
- `int pthread_exit(void *value)`
  - Permite a un proceso ligero finalizar su ejecución, indicando el estado de terminación del mismo.
  - El estado de terminación no puede ser un puntero a una variable local.

Ejemplo: crear y esperar


```
#include ..
#include <pthread.h>
void * thread_function(void *arg)
{  int i;
   for ( i=0 ; i < 2 ; i++ ) {
    printf("Thread says hi!\n"); sleep(1);  }
    printf("Thread exit\n");
    return NULL;
}
int main(int argc, char ** argv) {
  pthread_t mythread;
  if ( pthread_create(&mythread,NULL,thread_function, NULL)==-1 )
  {    printf("error creating thread.");    abort();  }

   printf("Wait for join to exit\n");

   if (pthread_join(mythread,NULL) == -1)
   {  printf("error joining thread.\n");    abort();  }
    exit(0);
}
```

Ejemplo: sumador con threads


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

struct sumapar {
  int n, m, r;
};
typedef struct sumapar
   sumapar_t;

void suma(sumapar_t * par) {
  int i;
  int suma=0;
  for (i=par->n;i<=par->m;i++)
   {
    suma +=i;
  }
  par->r=suma;
}
```

```
int main() {
  pthread_t th1, th2;
  sumapar_t s1 = {1,50,0};
  sumapar_t s2 = {51,100,0};

  pthread_create(&th1, NULL,
   (void*)suma, (void*)&s1);
  pthread_create(&th2, NULL,
   (void*)suma, (void*)&s2);

  pthread_join(th1, NULL);
  pthread_join(th2, NULL);

  printf("Suma=%d\n",
    s1.r+s2.r);
}
```

Atributos de un hilo

- Cada hilo tiene asociados un conjunto de atributos.
- Atributos representados por una variable de tipo **pthread_attr_t.**
- Los atributos controlan:
  - Un hilo es independiente o dependiente.
  - El tamaño de la pila privada del hilo.
  - La localización de la pila del hilo.
  - La política de planificación del hilo.

Si es de kernel
o de biblioteca.

↳ Sólo si es un thread de kernel

Atributos

Crear atributos:

- `int pthread_attr_init(pthread_attr_t * attr);`
  - Inicia una estructura de atributos de hilo.
- `int pthread_attr_destroy(pthread_attr_t * attr);`
  - Destruye una estructura de atributos de hilo.
- `int pthread_attr_setstacksize(pthread_attr_t * attr, int stacksize);`
  - Define el tamaño de la pila para un hilo
- `int pthread_attr_getstacksize(pthread_attr_t * attr, int *stacksize);`
  - Permite obtener el tamaño de la pila de un hilo.

Hilos dependientes e hilos independientes

- `int pthread_attr_setdetachstate(pthread_attr_t *attr, int detachstate)`
  - Establece el estado de terminación de un proceso ligero.
  - Si "detachstate" = PTHREAD_CREATE_DETACHED el proceso ligero liberara sus recursos cuando finalice su ejecución.
  - Si "detachstate" = PTHREAD_CREATE_JOINABLE no se liberan los recursos, es necesario utilizar pthread_join().

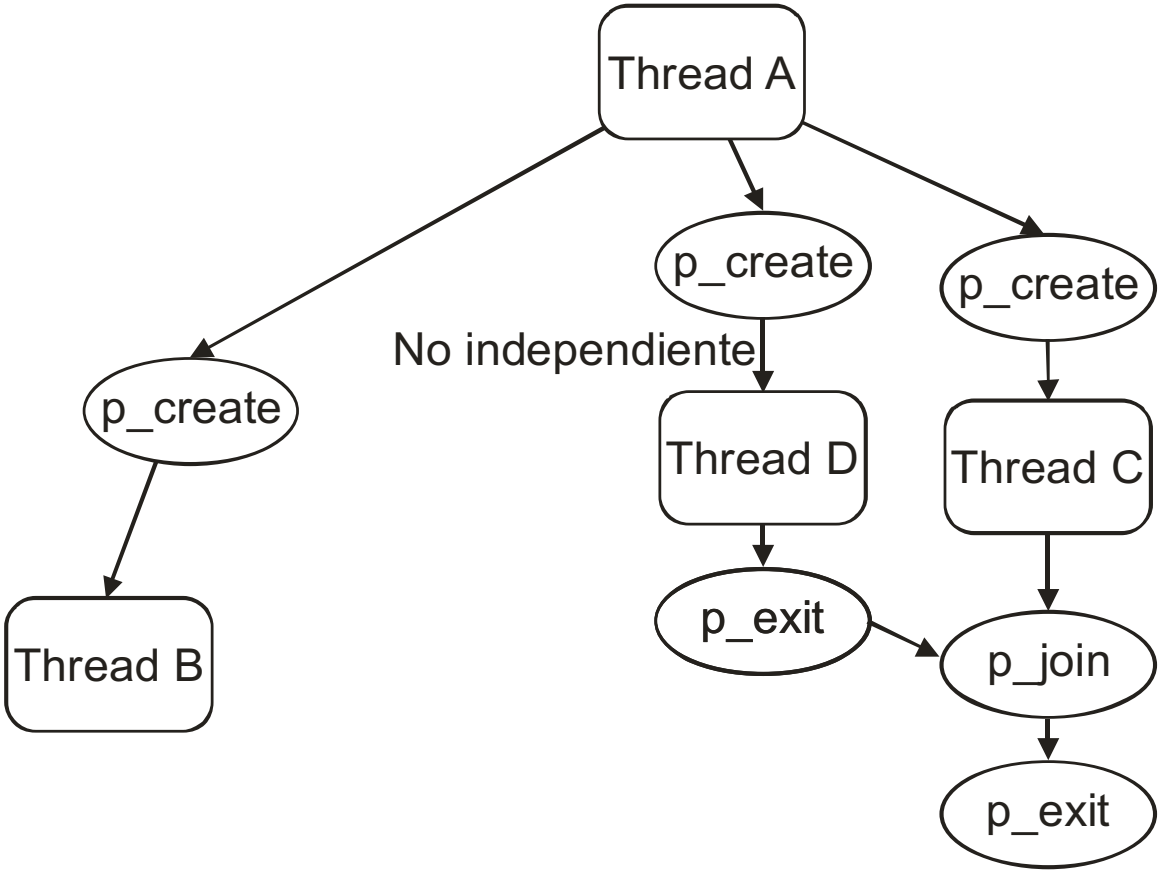
- `int pthread_attr_getdetachstate(pthread_attr_t *attr, int *detachstate)`
  - Permite conocer el estado de terminación

Ejemplo: Hilos independientes


```
#include <stdio.h>
#include <pthread.h>
#define MAX_THREADS 10
void func(void)  {
  printf("Thread %d \n", pthread_self());
  pthread_exit(0);
}

int main() {
  int j;
  pthread_attr_t attr;
  pthread_t thid[MAX_THREADS];
  pthread_attr_init(&attr);
  pthread_attr_setdetachstate(&attr,PTHREAD_CREATE_DETACHED);
  for(j = 0; j < MAX_THREADS; j ++)
    pthread_create(&thid[j], &attr,  func, NULL);
  sleep(5);
}
```

Ejemplo de jerarquía de threads

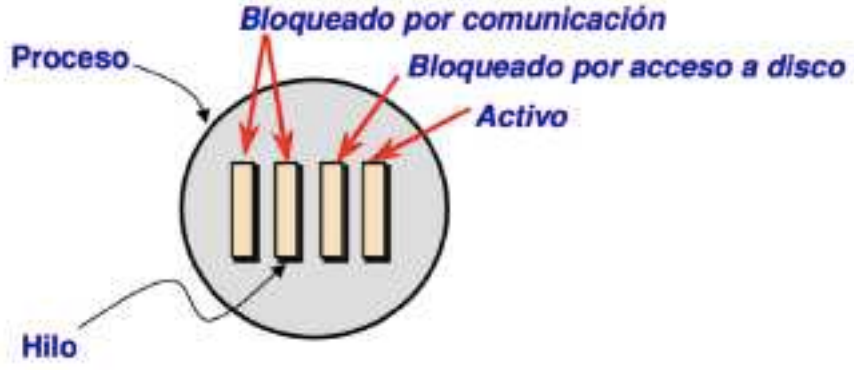


Contenido

- Concepto de hilo.
- Modelos de hilos.
- Aspectos de diseño.
- Hilos en pthreads.
- **Planificación de hilos**

Estados de un proceso con hilos

- Estado de un proceso con hilos:
  - Combinación de los estados de sus hilos:
  - Si hay un hilo en ejecución ->Proceso en ejecución
  - Si no hay hilos en ejecución pero sí preparados -> Proceso preparado
  - Si todos sus hilos bloqueados ->Proceso bloqueado



Planificación de threads.

- La planificación de ejecución de threads se basa en el modelo de prioridades y no utiliza el modelo de segmentación por segmentos de tiempo.

- Un thread continuará ejecutandose en la CPU hasta pasar a un estado que no le permita seguir en ejecución. S
  - Si se quiere alternancia entre threads, se debe asegurar que el thread permite la ejecución de otros threads
  - Por ejemplo usando *sleep()*.

Planificación de threads

- API permite especificar la planifiación (PCS o SCS) durante creación thread
  - PTHREAD_SCOPE_PROCESS: el thread usa planificación PCS
  - PTHREAD_SCOPE_SYSTEM: el thread usa planificación SCS
- El SO lo puede limitar
  - Para usuarios, Linux y Mac OS X solo permiten PTHREAD_SCOPE_SYSTEM

API de planificación


```
#include <pthread.h>
#include <stdio.h>
#define NUM_THREADS 5
int main(int argc, char *argv[]) {
   int i, scope;
   pthread_t tid[NUM THREADS];
   pthread_attr_t attr;
   /* get the default attributes */
   pthread_attr_init(&attr);
   /* first inquire on the current scope */
   if (pthread_attr_getscope(&attr, &scope) != 0)
      fprintf(stderr, "Unable to get scheduling scope\n");
   else {
      if (scope == PTHREAD_SCOPE_PROCESS)
         printf("PTHREAD_SCOPE_PROCESS");
      else if (scope == PTHREAD_SCOPE_SYSTEM)
         printf("PTHREAD_SCOPE_SYSTEM");
      else
         fprintf(stderr, "Illegal scope value.\n");
   }
```

API de planificación


```
   /* set the scheduling algorithm to PCS or SCS */
   pthread_attr_setscope(&attr, PTHREAD_SCOPE_SYSTEM);
   /* create the threads */
   for (i = 0; i < NUM_THREADS; i++)
      pthread_create(&tid[i],&attr,runner,NULL);
   /* now join on each thread */
   for (i = 0; i < NUM_THREADS; i++)
      pthread_join(tid[i], NULL);
}
/* Each thread will begin control in this function */
void *runner(void *param)
{
   /* do some work ... */
   pthread_exit(0);
}
```

Lecturas recomendadas

Básica

- Carretero 2007:
  - 3.6. Señales y excepciones.
  - 3.7. Temporizadores.
  - 3.13. Servicios.
  - 3.9 Threads

Complementaria

- Stallings 2005:
  - 4.1 Procesos e hilos.
  - 4.4 Gestión de hilos y SMP en Windows.
  - 4.5 Gestión de hilos y SMP en Solaris.
  - 4.6 Gestión de procesos e hilos en Linux.
- Silberschatz 2006:
  - 4 Hebras.

SISTEMAS OPERATIVOS: PROCESOS

Hilos y Procesos