Hilos de ejecución POSIX

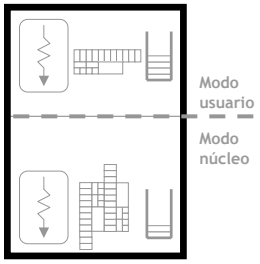
Sistemas Operativos (prácticas)
E.U. Informática en Segovia
Universidad de Valladolid

Hilos de ejecución

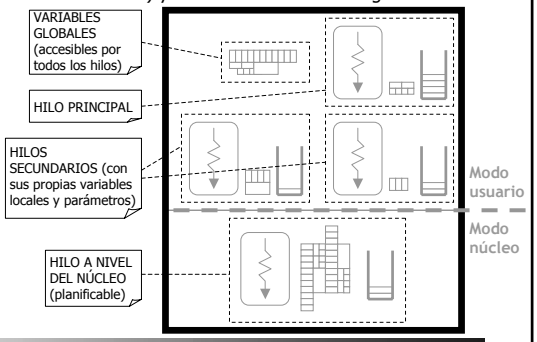
- Concepto
  - Un proceso convencional se caracteriza por:
    - Ser la *unidad de propiedad de recursos*, es decir, un conjunto de recursos asignados a un proceso, en particular:
      - Espacio de direcciones virtuales que contiene la imagen del proceso
      - Otros recursos que puede solicitar (dispositivos de E/S, archivos, etc.)
    - Ser la *unidad de ejecución*:
      - Es una determinada secuencia de instrucciones, ejecutadas dentro de la imagen de memoria de un proceso (en su contexto)
      - Esta secuencia se puede intercalar con las secuencias de otros procesos (mediante cambios de contexto) y alcanzándose de este modo concurrencia a nivel de procesos
  - La noción de hilo de ejecución (*thread*) surge cuando el sistema operativo gestiona independientemente las dos características fundamentales de un proceso tradicional

Hilos de ejecución

- Procesos tradicionales (pesados)
  - Concurrencia fuera del proceso (ínter procesos)
  - Unidad de propiedad de recursos
  - Tiene muchos atributos
  - Es costoso su creación, destrucción, cambio de contexto, etc.



- Procesos multihilo
  - Actividad concurrente dentro de un proceso pesado (intra proceso): pueden existir varios hilos
  - Comparten los recursos del proceso (variables globales, ficheros, etc.)
  - Tienen pocos atributos (contexto de hilo) y son menos costosos de gestionar



Hilos de ejecución

- ¿Cómo se programa con hilos? Se tienen básicamente dos posibilidades:
  - Utilizando un lenguaje de programación convencional y llamadas al sistema (o funciones de biblioteca)
    - Ejemplo: Hilos POSIX utilizados desde C, (enlazando con la librería pthreads, opción –lptrhead)
  - Utilizando construcciones lingüísticas (o clases) de un lenguaje de programación concurrente
    - Ejemplo: Tareas en Ada95, threads en Java
    - En este caso, el compilador traduce las construcciones lingüísticas a:
      - Llamadas al sistema (hilos a nivel de núcleo)
      - Llamadas a bibliotecas propias de soporte de hilos (hilos a nivel de usuario)

Hilos POSIX (pthreads)

- Dentro de un proceso POSIX convencional:
  - Existe un **hilo inicial** que ejecuta la función **`main()`**
  - Este hilo puede crear más hilos para ejecutar otras funciones dentro del espacio de direcciones del proceso
  - Todos los hilos de un proceso se encuentran al mismo nivel
    - Esto significa que son "hermanos", a diferencia de los proceso cuya relación es "padre-hijo"
  - Los hilos de un proceso comparten las variables y recursos globales (archivos, manejadores de señales, etc.) del proceso
    - Además, cada uno tiene una **copia privada** de sus parámetros iniciales y de las variables locales de la función que ejecuta (almacenados en su pila particular)
- El estándar POSIX define, entre otras, funciones para:
  - Creación de hilos
  - Creación/destrucción de atributos de creación de hilos
  - Terminación/espera a la terminación de un hilo
  - Identificación de hilos

Hilos POSIX (pthreads): creación

- **`pthread_create`**: Creación de hilos

```
#include <pthread.h>

int pthread_create(pthread_t *thread,
                   const pthread_attr_t *attr,
                   void *(* start_routine)(void *),
                   void *arg);
```

- Descripción
  - Crea inmediatamente el hilo en estado **preparado**, por lo que el hilo creado y su hilo creador **compitan** por la CPU según la política de planificación del sistema
  - Puede ser invocada por cualquier hilo del proceso (no sólo por el "hilo inicial") para crear otro hilo
  - Parámetros:
    - **`attr`** es el atributo que contiene las características del hilo creado (véanse atributos de un hilo en las siguientes transparencias)
    - **`start_routine`** es la función que ejecutará el hilo
    - **`arg`** es un puntero a los parámetros iniciales del hilo
    - En **`thread`** se devuelve el identificador del hilo creado si la llamada tiene éxito
  - Valor de retorno:
    - 0 si éxito y un valor negativo si hay error

Hilos POSIX (pthreads): atributos

- **`pthread_attr_init/destroy`**: Manipulación atributos de un hilo

```
#include <pthread.h>

int pthread_attr_init(pthread_attr_t *attr);
int pthread_attr_destroy(pthread_attr_t *attr);
```

- Descripción
  - **`pthread_attr_init`** inicializa el objeto de atributos de un hilo **`attr`** y establece los valores por defecto
  - Posteriormente, este objeto, con los atributos por defecto de un hilo, se puede utilizar para crear múltiples hilos
  - **`pthread_attr_destroy`**, destruye el objeto de atributos de un hilo, **`attr`**, y éste no puede volver a utilizarse hasta que no se vuelva a inicializar

Hilos POSIX (pthreads): atributos

- Descripción (continuación)
  - Atributos (más relevantes) de un hilo POSIX:
    - **`detachstate`**: controla si otro hilo hilo podrá esperar por la terminación de este hilo (mediante la invocación a **`pthread_join`**):
      - **`PTHREAD_CREATE_JOINABLE`** (valor por defecto)
      - **`PTHREAD_CREATE_DETACHED`**
    - **`schedpolicy`**: controla cómo se planificará el hilo
      - **`SCHED_OTHER`** (valor por defecto, planificación normal + no tiempo real)
      - **`SCHED_RR`** (*Round Robin* + tiempo real + privilegios *root*)
      - **`SCHER_FIFO`**(*First In First Out* + tiempo real + privilegios *root*)
    - **`scope`**: controla a qué nivel es reconocido el hilo
      - **`PTHREAD_SCOPE_SYSTEM`** (valor por defecto, el hilo es reconocido por el núcleo)
      - **`PTHREAD_SCOPE_PROCESS`** (no soportado en la implementación LinuxThreads de hilos POSIX)

Hilos POSIX (pthreads): atributos

- **`pthread_attr_get/getxxxxx`**: Establecimiento/Consulta atributos particulares de un objeto con los atributos de un hilo

```
#include <pthread.h>

int pthread_attr_setdetachstate (pthread_attr_t *attr,
                                 int detachstate);
int pthread_attr_getdetachstate (const pthread_attr_t *attr,
                                 int *detachstate);
int pthread_attr_setschedpolicy (pthread_attr_t *attr,
                                 int policy);
int pthread_attr_getdetachstate (const pthread_attr_t *attr,
                                 int *policy);
int pthread_attr_setscope       (pthread_attr_t *attr,
                                 int contentionscope);
int pthread_attr_getscope       (const pthread_attr_t *attr,
                                 int *contentionscope);
...
```


Hilos POSIX (pthreads): terminación

- **`pthread_exit`**: Terminación de un hilo

```
#include <pthread.h>

void pthread_exit(void *status);
```

- Descripción
  - **`pthread_exit`** finaliza explícitamente la ejecución del hilo que la invoca
    - La finalización de un hilo también se hace cuando finaliza la ejecución de las instrucciones de su función
  - La finalización del último hilo de un proceso finaliza la ejecución del proceso
  - Si el hilo es sincronizable (*joinable*) el identificador del hilo y su valor de retorno puede examinarse por otro hilo mediante la invocación a **`pthread_join`** a través del parámetro **`status`**

Hilos POSIX (pthreads): espera por terminación

- **`pthread_join`**: Esperar por la terminación de un hilo

```
#include <pthread.h>

int pthread_join(pthread_t tid, void **status);
```

- Descripción
  - Esta función suspende la ejecución del hilo que la invoca hasta que el hilo identificado por el valor **`tid`** finaliza, bien por la invocación a la función **`pthread_exit`** o por estar cancelado
  - Si **`status`** no es **`NULL`**, el valor devuelto por el hilo (el argumento de la función **`pthread_exit`**, cuando el hilo hijo finaliza) se almacena en la dirección indicada por **`status`**
  - El valor devuelto es o bien el argumento de la función **`pthread_exit`** o el valor **`PTHREAD_CANCELED`** si el hilo **`tid`** está cancelado
  - El hilo por el que se espera su terminación debe estar en estado sincronizable (*joinable state*)
    - Cuando un hilo en este estado termina, no se liberan sus propios recursos (descriptor del hilo y pila) hasta que otro hilo espere por él
    - La espera por la terminación de un hilo para el cual ya hay otro hilo esperando, genera un error

Hilos POSIX (pthreads): cancelación

- **`pthread_cancel`**: Solicitar la cancelación de un hilo

```
#include <pthread.h>

int pthread_cancel(pthread_t tid);
```

- Descripción
  - La cancelación es el mecanismo por el cual un hilo puede solicitar la terminación de la ejecución de otro
  - Dependiendo de la configuración del hilo al que se solicita su cancelación, puede aceptar peticiones de cancelación (**`PTHREAD_CANCEL_ENABLE`**, estado por defecto) o rechazarlas (**`PTHREAD_CANCEL_DISABLE`**)
  - En caso de aceptar peticiones de cancelación, un hilo puede completar la cancelación de dos formas diferentes:
    - De forma asíncrona (**`PTHREAD_CANCEL_ASYNCHRONOUS`**), o
    - De forma diferida (**`PTHREAD_CANCEL_DEFERRED`**, valor por defecto) hasta que se alcance un punto de cancelación
      - Un punto de cancelación (*cancelation point*) es un punto en el flujo de control de un hilo en el que se comprueba si hay solicitudes de cancelación pendientes
  - Cuando un hilo acepta una petición de cancelación, el hilo actúa como si se hubiese realizado la siguiente invocación **`pthread_exit(PTHREAD_CANCELED)`**

Hilos POSIX (pthreads): identificación

- **`pthread_self`**: devuelve el identificador de un hilo

```
#include <pthread.h>

pthread_t pthread_self(void);
int pthread_equal(pthread_t tid1, pthread_t tid2);
```

- Descripción
  - La función **`pthread_self`** devuelve el identificador de hilo (tid, *thread identifier*) del hilo que la invoca
  - Para comparar diferentes identificadores de hilo debe utilizarse la función **`pthread_equal`** que:
    - Devuelve 0 si los identificares no son iguales
    - Otro valor si los identificadores sí son iguales

Hilos POSIX (pthreads): Ejemplo 1

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

void *func_hilo (void *arg) {
   printf("Hilo creado: ¡Hola mundo!\n");
}

int main(void) {
   pthread_t      tid;
   pthread_attr_t atrib;

   printf("Hilo principal: INICIO\n");
   pthread_attr_init(&atrib);
   pthread_create(&tid, &atrib, func_hilo, NULL);
   printf("Hilo principal: HILO CREADO\n");
   pthread_join(tid, NULL);
   printf("Hilo principal: FIN\n");
}
```

Hilos POSIX (pthreads): Ejemplo 2

```
#include <stdio.h>
#include <pthread.h>
#define NUM_IT 20000

int a= 0;   /* Variable global */

void *hilo (void *arg) {
   int id, i, aux;

   id= (int)arg;
   printf("Soy el hilo %d con tid=
           %d\n",id, pthread_self());
   for (i=0; i<NUM_IT; i++)
      if (id==0) {
         aux= a; aux++; a= aux;
      } else {
         aux= a; aux--; a= aux;
   pthread_exit(NULL); /*opcional*/
}
```

```
int main(void) {
   pthread_t tid1, tid2, tid3;

   printf("Valor inicial de a: %d\n", a);
   pthread_create(&tid1, NULL, hilo, (void *)0);
   pthread_create(&tid2, NULL, hilo, (void *)1);
   pthread_create(&tid1, NULL, hilo, (void *)2);
      /* Nótese el paso de parámetros al hilo */
   printf("Esperando terminación hilos ...\n");
   pthread_join(tid1, NULL);
   pthread_join(tid2, NULL);
   pthread_join(tid3, NULL);

   printf("Hilos terminados.\n");
   printf("Valor final de a: %d\n", a);

   return 0;
}
```


Hilos POSIX (pthreads): Ejemplo 3

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

int status= 0;          /*Variable global, donde se almacena estado terminación hilos */

void *escribir_mensaje( void *arg ) {  /* función con el código de los hilos creados */
   printf("%s \n", (char *)arg);
   pthread_exit(&status);
}

int main(void) {
   pthread_t tid1, tid2;
   char *mensaje1 = "Hilo 1";
   char *mensaje2 = "Hilo 2";
   int *ptr_status;

   pthread_create( &tid1, NULL, escribir_mensaje, (void *)mensaje1);
   pthread_create( &tid2, NULL, escribir_mensaje, (void *)mensaje2);
   pthread_join(tid1, (void **)&ptr_status);              /* Nótese cómo se recupera */
   printf("Estado de terminación Hilo 1: %d\n", *ptr_status);  /* estado terminación */
   pthread_join(tid2, (void **)&ptr_status);
   printf("Estado de terminación Hilo 2: %d\n", *ptr_status);
}
```