HILOS POSIX

LABORATORIO DE SISTEMAS DE OPERACIÓN I
(ci-3825)

Prof. Yudith Cardinale

Enero-marzo 2011

HILOS POSIX

- Librería

  #include <pthread.h>

- Función que se comportará como hilo:

```
void * mihilo(void *arg)  {
    . . . //instrucciones
 }
```

- Algunos tipos y macros:

  pthread_t idh; //id del hilo

  pthread_attr_t atrh; //atributos del hilo

  PTHREAD_SCOPE_SYSTEM: hilo a nivel de kernel

  PTHRAD_SCOPE_PROCESS: hilo a nivel de usuario

- Inicializar y destruir atributos

  pthread_attr_init(pthread_att_t *attr);

  pthread_attr_destroy(pthread_att_t *attr);

HILOS POSIX

- Ejemplo:

```
p_thread_attr_t misattr;
…
pthread_attr_init(&misattr);
pthread_attr_setscope(&misattr,PTHREAD_SCOPE_PROCESS);
…..
pthread_attr_destroy(&misattr);
```

- Algunos atributos que pueden ser inicializados:

```
int pthread_attr_setdetachstate(pthread_attr_t *attr, int detachstate);
int pthread_attr_setguardsize(pthread_attr_t *attr, size_t guardsize);
int pthread_attr_setinheritsched(pthread_attr_t *attr, int inheritsched);
int pthread_attr_setschedparam(pthread_attr_t *attr, const struct
    sched_param *param);
int pthread_attr_setschedpolicy(pthread_attr_t *attr, int policy);
int pthread_attr_setscope(pthread_attr_t *attr, int contentionscope);
int pthread_attr_setstackaddr(pthread_attr_t *attr, void *stackaddr);
int pthread_attr_setstacksize(pthread_attr_t *attr, size_t stacksize);
```

HILOS POSIX

- Ejemplo:

```
p_thread_attr_t misattr;
…
pthread_attr_init(&misattr);
pthread_attr_setscope(&misattr,PTHREAD_SCOPE_PROCESS);
…..
pthread_attr_destroy(&misattr);
```

- Algunos atributos que pueden ser inicializados:

* Por defecto, los hilos se crean con ciertos atributos: hilos de kernel, todos con la misma cantidad de stack, la misma prioridad que el hilo padre, la misma política de scheduling, etc.
* Estos atributos se pueden cambiar.
* Si se quiere trabajar con los atributos por defecto, se usa NULL

```
int pthread_attr_setscope(pthread_attr_t *attr, int contentionscope);
int pthread_attr_setstackaddr(pthread_attr_t *attr, void *stackaddr);
int pthread_attr_setstacksize(pthread_attr_t *attr, size_t stacksize);
```

HILOS POSIX

- **Creación**

  int pthread_create(pthread * tid, const pthread_attr_t *attr,

  void *(start_ routine)(void *), void * arg) ==> arguments:

  - tid: A unique identifier for the new thread returned by the subroutine.
  - attr: An attribute object that may be used to set thread attributes. You can specify a thread attributes object, or NULL for the default values.
  - start_routine: the C routine that the thread will execute once it is created.
  - arg: A single argument that may be passed to start_routine. It must be passed by reference as a pointer cast of type void. NULL may be used if no argument is to be passe

- pthread_create devuelve 0 si la creación es exitosa o !=0 si hay error
- Ejemplo:

```
pthread_t tid; pthread_attr_t mis_atr;
….
pthread_create(&tid,&mis_atr, mihilo,NULL)
```

HILOS POSIX

- **Terminación**

  There are several ways in which a Pthread may be terminated:

  - The thread returns from its starting routine (the main routine for the initial thread).
  - The thread makes a call to the **pthread_exit** subroutine (covered below).
  - The thread is canceled by another thread via the **pthread_cancel(tid)**.
  - The entire process is terminated due to a call to either the exec or exit subroutines

- void pthread_exit(void *value)
- Ejemplos:

trozo de código de hilo 1:

```
int *contador;
contador = (int *)malloc(sizeof(int));
….
pthread_exit((void*)contador);
```

trozo de código de hilo 2:

```
int contador;
   ….
   ….
pthread_exit((void*)&contador);
```

HILOS POSIX

- **Terminación**
  - El valor retornado pasado como argumento en **pthread_exit** es recibido en **pthread_join** llamado por otro hilo (al estilo de **exit** y **wait** de procesos)
  - La llamada a **exit()** en cualquier parte del programa ocasiona que todos los hilos terminen
  - La salida de un hilo deberá ser con **pthread_exit** y no con **exit. Si lo hace con exit() todos los hilos finalizarán.**

HILOS POSIX

- **Joining hilos:**

  int pthread_join(pthread_t tid,void **value)

  Devuelve 0 si es éxitoso, !=0 en caso contrario.

- Ejemplos:
  - pthread_join(tid,NULL);
  - void *status; int rc;

    …..

    rc = pthread_join(tid, &status);

    if (rc) {

        printf("ERROR; return code from pthread_join() is %d\n", rc);

        exit(-1);

    }

HILOS POSIX

- **Pase de argumentos:**

Ejemplo 1

```
main () {
        long *taskids[NUM_THREADS];
        pthreads_t threads[NUM_THREADS];
        int t;
        for(t=0; t<NUM_THREADS; t++){
          taskids[t] = (long *) malloc(sizeof(long));
          *taskids[t] = t;
          printf("Creating thread %ld\n", t);
          rc = pthread_create(&threads[t], NULL, PrintHello, (void *) taskids[t]);
          ...
}
```

HILOS POSIX

- **Pase de argumentos:**

Ejemplo 2:

```
struct thread_data{

   int  thread_id;
   int  sum;
  char *message;
};
struct thread_data thread_data_array[NUM_THREADS];

void *PrintHello(void *threadarg) {
   struct thread_data *my_data;
   int taskid, sum;
   char *hello_msg=(char *)malloc(sizeof(char)*MAX_CHAR);
   …..
  my_data = (struct thread_data *) threadarg;
  taskid = my_data->thread_id;
  sum = my_data->sum;
  hello_msg = my_data->message;
  ….
 pthread_exit(NULL);
}
```

HILOS POSIX

- **Pase de argumentos: Ejemplo 2 (cont.):**

```
int main (int argc, char *argv[]){
    pthreads_t threads[NUM_THREADS];
    int t, rc;
    void *status;
    for(t=0; t<NUM_THREADS; t++){
        ...
        thread_data_array[t].thread_id = t;
        thread_data_array[t].sum = sum;
        thread_data_array[t].message = messages[t];

        rc = pthread_create(&threads[t], NULL, PrintHello, (void *) &thread_data_array[t]);

    }
    ....
   for(t=0; t<NUM_THREADS; t++) {
      rc = pthread_join(thread[t], &status);
      if (rc) {
          printf("ERROR; return code from pthread_join() is %d\n", rc);
          exit(-1);
      }
     printf("Main: completed join with thread %ld having a status of %ld\n",t,(long)status);
  }
```

HILOS POSIX

- **Compilación**
  - Es importante haber incluido la librería pthreads (#include <pthreads.h>)
  - Un solo programa:

    gcc hilos.c -lpthread -o ejecutable
  - En el makefile

    ejecutable: depend1.o depend2.o … dependN.o

    gcc -o ejecutable depend1.o depend2.o … dependN.o -lpthread
- **Enlaces de interés**
  - https://computing.llnl.gov/tutorials/pthreads/

SINCRONIZACIÓN EN HILOS

- INICIALIZACIÓN:

```
#include <pthread.h>

int pthread_mutex_init(pthread_mutex_t * restrict mutex, const pthread_mutexattr_t *
    restrict attr);

pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
```

- LIBERACIÓN:

```
int pthread_mutex_destroy(pthread_mutex_t *mutex);
```

- ACCESO:

```
int pthread_mutex_lock(pthread_mutex_t *mutex);

int pthread_mutex_trylock(pthread_mutex_t *mutex);

int pthread_mutex_unlock (pthread_mutex_t *mutex);
```

SINCRONIZACIÓN EN HILOS

- **pthread_mutex_lock() -** acquire a lock on the specified mutex variable. If the mutex is already locked by another thread, this call will block the calling thread until the mutex is unlocked.
- **pthread_mutex_unlock()** - unlock a mutex variable. An error is returned if mutex is already unlocked or owned by another thread.
- **pthread_mutex_trylock()** - attempt to lock a mutex or will return error code if busy. Useful for preventing deadlock conditions.

SINCRONIZACIÓN EN HILOS

- **Ejemplo 1:**

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

void *functionC();
pthread_mutex_t mutex1 = PTHREAD_MUTEX_INITIALIZER;
int  counter = 0;

main() {
   int rc1, rc2;
   pthread_t thread1, thread2;

   if( (rc1=pthread_create( &thread1, NULL, &functionC, NULL)) {
      printf("Thread creation failed: %d\n", rc1);
   }

   if( (rc2=pthread_create( &thread2, NULL, &functionC, NULL)) ) {
      printf("Thread creation failed: %d\n", rc2);
   }

   pthread_join( thread1, NULL);
   pthread_join( thread2, NULL);
   exit(0);
}
```

```
void *functionC() {
   pthread_mutex_lock( &mutex1 );
   counter++;
   printf("Counter value: %d\n",counter);
   pthread_mutex_unlock( &mutex1 );
}
```

SINCRONIZACIÓN EN HILOS

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

    void *functionC1();
    void *functionC2();
     pthread_mutex_t mutex1 = PTHREAD_MUTEX_INITIALIZER,
                     mutex2 = PTHREAD_MUTEX_INITIALIZER;
     int  counter1 = 0, counter2=0;
     main() {
        int rc1, rc2;
        pthread_t threads1[MAXTHREADS], threads2[MAXTHREADS];
     for (i = 0; i < MAXTHREADS; i ++)
        if( (rc1=pthread_create(  &threads1[i], NULL, &functionC, NULL)) {
           printf("Thread creation failed: %d\n", rc1);
        }
     for (i = 0; i < MAXTHREADS; i ++)
        if( (rc2=pthread_create( &threads2[i], NULL, &functionC, NULL)) ) {
           printf("Thread creation failed: %d\n", rc2);
        }
        for (i = 0; i < MAXTHREADS; i ++)
          pthread_join( threads1[i], NULL);
        for (i = 0; i < MAXTHREADS; i ++)
          pthread_join( threads2[i], NULL);
        exit(0);
     }
```

SINCRONIZACIÓN EN HILOS

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

   void *functionC1();
   void *functionC2();
    pthread_mutex_t mutex1 = PTHREAD_MUTEX_INITIALIZ
                    mutex2 = PTHREAD_MUTEX_INITIALIZ
    int  counter1 = 0, counter2=0;
    main() {
      int rc1, rc2;
      pthread_t threads1[MAXTHREADS], threads2[MAXTHREADS];
    for (i = 0; i < MAXTHREADS; i ++)
      if( (rc1=pthread_create(  &threads1[i], NULL, &functionC, NULL)) {
        printf("Thread creation failed: %d\n", rc1);
      }
    for (i = 0; i < MAXTHREADS; i ++)
      if( (rc2=pthread_create( &threads2[i], NULL, &functionC, NULL)) ) {
        printf("Thread creation failed: %d\n", rc2);
      }
      for (i = 0; i < MAXTHREADS; i ++)
        pthread_join( threads1[i], NULL);
      for (i = 0; i < MAXTHREADS; i ++)
        pthread_join( threads2[i], NULL);
      exit(0);
    }
```

```
void *functionC1() {
  pthread_mutex_lock( &mutex1 );
  counter1++;
  printf("Counter1 value: %d\n",counter1);
  pthread_mutex_unlock( &mutex1 );
}
```

```
void *functionC2() {
  pthread_mutex_lock( &mutex2 );
  counter2=counter2+2;
  printf("Counter2 value: %d\n",counter2);
  pthread_mutex_unlock( &mutex2 );
}
```