Sistemas Operativos

SC's Pthreads

[Ste05]: cap. 11

SC's Pthreads

- Identificación de threads
- Creación de threads
- Terminación de threads
- Threads + fork(), exec()
- Ejemplo

Identificación de threads

- Tipo de dato `pthread_t` para el identificador de thread (unsigned int en Solaris)

```
#include <pthread.h>

pthread_t  pthread_equal(pthread_t tid1,
  pthread_t tid2);
```

  - Devuelve (≠0,TRUE) si `tid1=tid2`
  - Devuelve (0, FALSE) si `tid1≠tid2`

```
pthread_t pthread_self(void);
```

  - Devuelve el identificador del thread que llama (similar a `getpid()` en procesos)

Creación de threads

```
#include <pthread.h>
int pthread_create(pthread_t *tidp,
  const pthread_attr_t *attr,
  void *(*start_rtn)(void *),
  void *arg);
```

- Devuelve: 0 si OK, número de error si falla
  - `tidp`: Variable donde pone el identif. de thread creado
  - `attr`: Atributos de creación del thread (NULL para atributos por defecto).
  - `start_rtn`: función donde comienza la ejecución del thread
  - `arg`: argumento(s) de la función `start_rtn`

Terminación de threads

- Si un thread ejecuta `exit`, termina el proceso entero (todos los threads).
- Para que un thread termine sin acabar el proceso entero debe hacer:
  - Retornar de su rutina de comienzo. El valor devuelto es el código de terminación del thread.
  - Thread cancelado por otro thread en el mismo proceso (`pthread_cancel`)
  - Ejecutar `pthread_exit`

```
#include <pthread.h>
int pthread_exit(void *rval_ptr);
```

    - El puntero `rval_ptr` está disponible para el resto de threads por medio de `pthread_join`

Terminación de threads

```
#include <pthread.h>
int pthread_join(pthread_t tid,
  void **rval_ptr);
```

- Devuelve: 0 si OK, número de error si falla
- El thread que ejecuta `pthread_join` se bloquea hasta que el thread `tid` termina (similar a `waitpid` en procesos). `rval_ptr` vale:
  - Si el thread `tid` simplemente retorna de su función start, `rval_ptr` toma el valor devuelto
  - `rval_ptr=PTHREAD_CANCELED` si el thread `tid` es cancelado
  - Si no interesa el valor devuelto por el thread `tid`, poner `NULL` en `rval_ptr`

Threads + fork() exec()

- Si un thread hace fork(), dos posibilidades
  - Nuevo proceso con solo un thread (el que hace fork)
    - Sería lo lógico si luego se llama a exec()
  - Nuevo proceso con todos los threads
    - Sería lo lógico si no hay exec() posterior
- En hendrix (solaris 10)
  - fork() duplica unicamente el thread que llama a fork()
  - forkall() nueva SC para replicar todos los threads
- Si un thread hace exec(), funciona como siempre, es decir, desaparecen todos los threads y se sustituye por el ejecutable indicado en exec()

Thr_n.c

```
#include <stdio.h> <stdlib.h> <pthread.h> "error.h"
struct arg {        /* estructura para pasar arg/res al thread */
    int ini;        /* indice inicial*/
    int fin;        /* indice final */
    int res;        /* resultado devuelto */
};
int n,n_thr,*v;      /* var global compartidas por todos los threads */

void main(int argc,char *argv[]) {
    int i,tam,error,suma;
    pthread_t *tid;
    struct arg *param;

    if (argc != 3){printf("Uso: thr_n <int> <n_thr>\n");exit(1);}
    n=atoi(argv[1]);n_thr=atoi(argv[2]);

    v=calloc(n,sizeof(int));  /* reserva vector a sumar */
    tid=calloc(n_thr,sizeof(pthread_t));          /* reserva tid_threads */
    param=calloc(n_thr,sizeof(struct arg));       /* reserva parametros */

    for (i=0;i<n;i++) v[i]=1; /* inicializacion vector a sumar */
    tam=n/n_thr;    /* tamaño trozo para cada thread */
    printf("Calculando S(%d) en %d threads -> tam=%d\n",n,n_thr,tam);
```

Thr_n.c

```
/* creando threads para suma parciales */
   for (i=0;i<n_thr;i=i+1) {
       param[i].ini=tam*i;                  /* param1 del thread */
       param[i].fin=tam*(i+1);    /* param2 del thread */
       error=pthread_create(&tid[i],NULL,start,&param[i]);
       if (error!=0) syserr(pthread_create);
   }

   suma=0;
   if (n_thr*tam<n)                /* falta sumar el resto */
       for (i=n_thr*tam;i<n;i=i+1) suma=suma+v[i];

  for (i=0;i<n_thr;i=i+1) {        /* espera terminacion threads */
       pthread_join(tid[i],NULL);
       suma=suma+param[i].res;    /* extrae resultado del thread */
  }
   printf("Terminado. S(%d)=%d\n",n,suma);
}
```

Thr_n.c

```
void *start(void *p) {
   pthread_t tid;
   int i,ini,fin,tmp;

   ini=((struct arg *)p)->ini;     /* ini=arg1 */
   fin=((struct arg *)p)->fin;      /* fin=arg2 */
   tmp=0;
   for (i=ini;i<fin;i=i+1) tmp=tmp+v[i];/* calcula la suma parcial */
    ((struct arg *)p)->res=tmp;    /* almacena resultado */
    pthread_exit(NULL);            /* acaba "sin devolver" resultado */
}
```