Threads

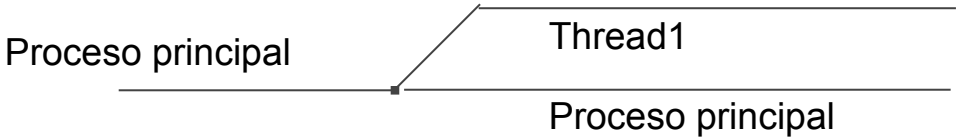
Mariana Prieto
2018

¿Qué es un thread?: Procesos “paralelizables”

Muchas veces nos encontramos haciendo más de un procesamiento al mismo tiempo, en el marco del mismo programa.

De ser así, podemos “paralelizar” los procesos para:

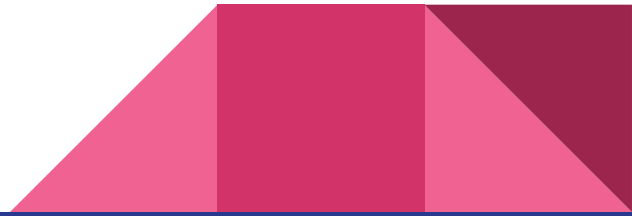
- Hacer más eficiente nuestro programa desde un punto de vista del tiempo
- Ejecutar procesos en simultáneo que no son secuenciales (no necesitan de la finalización de otro proceso para comenzar a funcionar)



¿Por qué un thread y no un fork?

- Los threads son una herramienta más “liviana” que la división de procesos (no tienen un número de PID asignado, lo que implica que desde el punto de vista del sistema operativo solo se ejecuta un proceso, y lo hace más ágil)

- Al estar trabajando en el marco de un mismo proceso, debemos ser cuidadosos porque los distintos threads tienen acceso al mismo juego de variables (globales) que el proceso original, y por lo tanto hay mayor posibilidad de errores.



¿Para qué puedo usar threads?

Los threads están pensados para ejecutar un proceso (típicamente, una función), en paralelo con el programa principal. Algunos ejemplos de threads pueden ser:

- Operaciones matemáticas extensas
- Programas Cliente - Servidor
- Procesamiento de señales

```
int main () {

        …

        pthread_create( … , CalculoExtenso() , ...);

        …

        pthread_exit ( NULL );
}
```

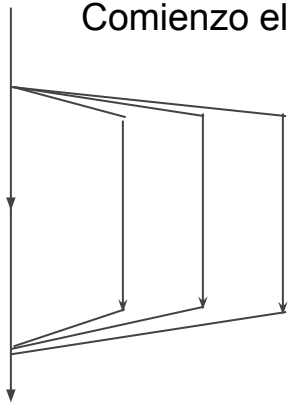
```
void CalculoExtenso (void) {

        …

        pthread_exit ( NULL );
}
```

Ejemplo:

Sea f(x) = (g(x) x h(x)) / y(x)

```
int main () {
        …

        pthread_create( thread1 , FuncionG() , ...);
        pthread_create( thread2 , FuncionH() , ...);
        pthread_create( thread3 , FuncionY() , ...);

        … //ACA HAGO OTRAS COSAS //…

        pthread_join( thread1 , FuncionG() , ...);
        pthread_join( thread2 , FuncionH() , ...);
        pthread_join( thread3 , FuncionY() , ...);
        ...
        pthread_exit ( NULL );
}
```

```
void FuncionG (void) {

        //Hago las operaciones matemáticas

        pthread_exit ( NULL );
}
```



Algunos prototipos y su funcionamiento

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

Header con todos los prototipos para usar POSIX threads

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

Crea un nuevo thread, que invoca la rutina start_routine

```
int pthread_join(pthread_t thread, void **retval);
```

Espera a que termine thread y continua la ejecución como un solo thread

```
void pthread_exit(void *retval);
```

Finaliza el thread desde donde se la llama

pthread_t: tipo de datos para almacenar el ID del thread

pthread_attr_t: se utiliza si queremos que el thread tenga un atributo en especial (sino es NULL)

Ejemplo:

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

#define N 100
int a= 0;

void* hola( void* arg){

        printf("Soy el thread %d\n", a++);
        pthread_exit(NULL);
}

int main(void){

        pthread_t id[N];
        for( int i = 0 ; i < N ; i++ ){
                pthread_create( id+i, NULL, hola, (void*) i);
        }

        pthread_join(id[N-1],NULL);
        pthread_exit(NULL);
}
```

Threads - bloqueo de recursos

```
int pthread_mutex_int(pthread_mutex_t *mutex,
                      const pthread_mutexattr_t *attr);
```

→ inicializa el mutex

o

```
pthread_mutex_t mutexVar = PTHREAD_MUTEX_INITIALIZER;
```

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

→ destruye el mutex (lo elimina de la memoria)

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

→ Bloquea el mutex si no lo tiene nadie. Si alguien tiene bloqueado el mutex el hilo espera hasta que el hilo que lo tiene bloqueado lo libere

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

→ libera el mutex