4. Threads em Unix

Só recentemente a especificação POSIX da API do Unix estabeleceu uma interface standard para a criação e terminação de *threads*. Esta interface é geralmente implementada como uma biblioteca (libpthreads.a), podendo ou não ser directamente suportada pelo kernel do sistema operativo. Existem versões do Unix em que os *threads* são directamente implementados no kernel e escalonados independentemente (*kernel level threads*), enquanto que em outras versões o que o kernel vê são apenas processos, "existindo" os *threads* unicamente na biblioteca de suporte, e sendo escalonados para execução apenas na "fatia de tempo" dos respectivos processos (*user level threads*).

Quando um programa começa a executar na função `main()` constitui um novo processo e pode considerar-se que contém já um *thread*. Novos *threads* são criados através da chamada de um serviço e começam a executar numa função que faz parte do código do programa. O serviço de criação de novos *threads* é então:

```
#include <pthread.h>

int pthread_create(pthread_t *tid, const pthread_attr_t *attr,
                   void * (* start)(void *), void *arg);
```

Retorna 0 no caso de sucesso e um código de erro no caso contrário.
No caso de erro o código retornado pode ser passado directamente a `strerror()` ou a variável global `errno` poderá ser preenchida com esse valor antes de se chamar `perror()`.

Cada novo *thread* é representado por um identificador (*thread identifier* ou tid) de tipo `pthread_t`. É necessário passar o endereço de uma variável desse tipo, como primeiro parâmetro de `pthread_create()`, para receber o respectivo tid. O segundo parâmetro serve para indicar uma série de atributos e propriedades que o novo *thread* deverá ter. Se aqui se fornecer o valor `NULL`, o novo *thread* terá as propriedades por defeito, que são adequadas na maior parte dos casos. O terceiro parâmetro indica a função de início do *thread*. É uma função que deve existir com o seguinte protótipo:

```
void * nome_da_função(void *arg)
```

A função aceita um apontador genérico como parâmetro, que serve para passar qualquer informação, e retorna também um apontador genérico, em vez de um simples código de terminação. Se o valor de retorno for usado, é necessário que aponte para algo que não deixe de existir quando o *thread* termina. Finalmente o quarto parâmetro é o valor do apontador a ser passado à função de inicíco, como seu parâmetro.

Uma vez criado o novo *thread* ele passa a executar concorrentemente com o principal e com outros que porventura sejam criados.

Um *thread* termina quando a função de início, indicada quando da criação, retornar, ou quando o próprio *thread* invocar o serviço de terminação:

```
#include <pthread.h>

void pthread_exit(void *value_ptr);
```

onde **`value_ptr`** é o apontador que o *thread* deve ter como resultado.

Quando um *thread* termina pode retornar um apontador para uma área de memória que contenha qualquer tipo de resultado. Essa área de memória deve sobreviver o *thread*, ou seja, não pode ser nenhuma variável local, porque essas deixam de existir quando o *thread* termina.

Qualquer *thread* pode esperar que um dado *thread* termine, e ao mesmo tempo obter o seu valor de retorno, que é um apontador genérico (`void *`). Basta para isso usar o serviço:

```
#include <pthread.h>

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

onde **`thread`** é o identificador do *thread* (tid) que se pretende esperar, e **`value_ptr`** é o endereço de um apontador onde será colocado o resultado do *thread* que vai terminar. Se se passar para este parâmetro o valor `NULL`, o retorno do *thread* é ignorado.
O serviço retorna 0 no caso de sucesso e um código de erro no caso contrário.

Por defeito, quando um *thread* termina, o seu valor de retorno (o apontador genérico) não é destruído, ficando retido em memória até que algum outro *thread* execute um `pthread_join()` sobre ele. É possível criar *threads* que não são *joinable* e que por isso, quando terminam, libertam todos os seus recursos, incluindo o valor de retorno. No entanto não é possível esperar por esses *threads* com `pthread_join()`. Estes *threads* designam-se por *detached*, podendo ser já criados nessa situação (usando o parâmetro **`attr`** de `pthread_create()`), ou podendo ser colocados nessa situação após a criação, com o serviço:

```
#include <pthread.h>

int pthread_detach(pthread_t thread);
```

onde **`thread`** é o identificador do *thread* (tid) que se pretende tornar *detached*.
O serviço retorna 0 no caso de sucesso e um código de erro no caso contrário.

O normal será o próprio *thread* tornar-se ele próprio *detached*. Para isso necessita de conhecer o seu próprio tid. Um *thread* pode obter o seu tid com o serviço:

```
#include <pthread.h>

pthread_t pthread_self(void);
```

Retorna sempre o tid do *thread* que o invoca.

Segue-se um exemplo simples de demonstração do uso de *threads* em UNIX:

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

int global;

void *thr_func(void *arg);

int main(void)
{
```

```
  pthread_t tid;

  global = 20;
  printf("Thread principal: %d\n", global);
  pthread_create(&tid, NULL, thr_func, NULL);
  pthread_join(tid, NULL);
  printf("Thread principal: %d\n", global);
  return 0;
}

void *thr_func(void *arg)
{
    global = 40;
    printf("Novo thread: %d\n", global);
    return NULL;
}
```

Quando se cria um novo *thread* é possível especificar uma série de atributos e propriedades passando-os a `pthread_create()` através de uma variável do tipo `pthread_attr_t`. Essa variável terá de ser previamente inicializada com o serviço `pthread_attr_init()` e depois modificada através da chamada de serviços específicos para cada atributo.

Algumas dessas funções de construção da variável `pthread_attr_t` estão listadas a seguir:

```
#include <pthread.h>

int pthread_attr_init(pthread_attr_t *attr);
int pthread_attr_destroy(pthread_attr_t *attr);
int pthread_attr_setstacksize(pthread_attr_t *attr, int size);
int pthread_attr_getstacksize(pthread_attr_t *attr, int *size);
int pthread_attr_setstackaddr(pthread_attr_t *attr, int addr);
int pthread_attr_getstackaddr(pthread_attr_t *attr, int *addr);
int pthread_attr_setdetachstate(pthread_attr_t *attr, int state);
int pthread_attr_getdetachstate(pthread_attr_t *attr, int *state);
int pthread_attr_setscope(pthread_attr_t *attr, int scope);
int pthread_attr_getscope(pthread_attr_t *attr, int *scope);
int pthread_attr_setinheritsched(pthread_attr_t *attr, int sched);
int pthread_attr_getinheritsched(pthread_attr_t *attr, int *sched);
int pthread_attr_setschedpolicy(pthread_attr_t *attr, int policy);
int pthread_attr_getschedpolicy(pthread_attr_t *attr, int *policy);
int pthread_attr_setschedparam(pthread_attr_t *attr,
                               struct sched_param *param);
int pthread_attr_getschedparam(pthread_attr_t *attr,
                               struct sched_param *param);
```

Retornam 0 no caso de sucesso e um código de erro no caso contrário.

Por exemplo, para criar um *thread* já no estado de *detached* deveria usar-se:

```
pthread_attr_t attr;

…
pthread_attr_init(&attr);
pthread_attr_setdetachstate(&attr, PTHREAD_CREATE_DETACHED);
pthread_create(&tid, &attr, …, …);
…
```

```
pthread_attr_destroy(&attr);
…
```

Para outros atributos e propriedades suportadas, consultar o manual (`man`) do Unix.