Sistemas Operativos: Threads POSIX (`libpthread`)

Pedro F. Souto (`pfs@fe.up.pt`)

March 16, 2012

Sumário

libpthread (*pthreads*)

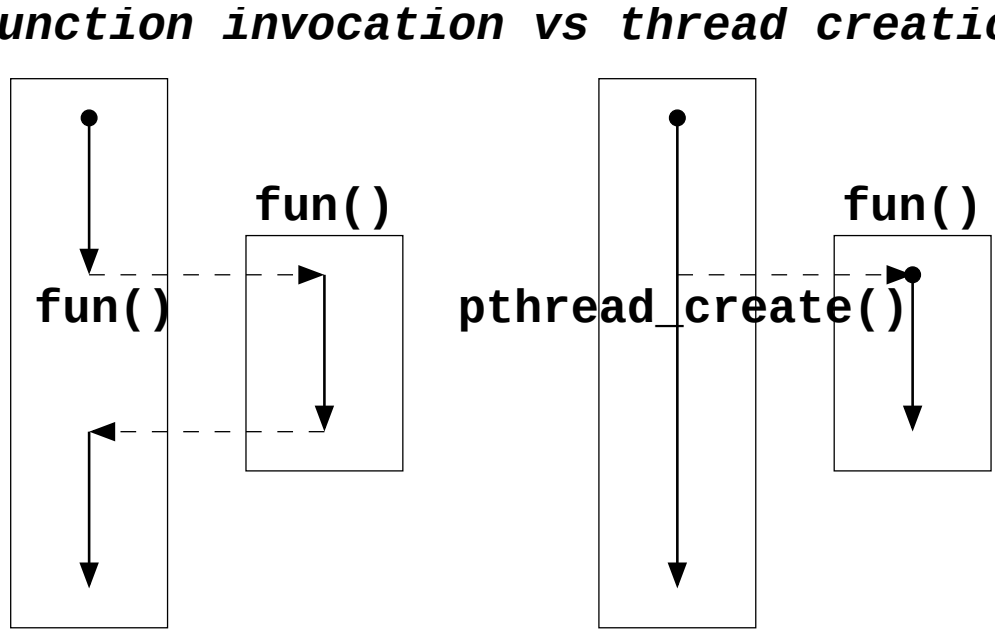
- Biblioteca de *threads* especificada em POSIX:

  A sua utilização promove portabilidade do código.

- Esta biblioteca pode ser implementada usando:
  - *kernel-level threads*, p.ex. a biblioteca distribuída com Linux;
  - *user-level threads*, p.ex. algumas bibliotecas disponíveis para Linux;
  - usando ambos tipos de *threads*, p.ex. a biblioteca distribuída com Solaris.

Funções básicas de gestão de *threads*

`int pthread_create(pthread_t *id, ...)` cria um *thread* que executa a função especificada no seu argumento:



`void pthread_exit(void *value_ptr)` termina o *thread*;

`int pthread_join(pthread_t thread, void **value_ptr` espera que o *thread* especificado no argumento `thread` termine.

Execução de Programas *Multithreaded*

- Num programa *multithreaded* um *thread* é criado quando:
  - O programa inicia: `main()` é executado pelo *thread principal*.
  - Quando da invocação de `pthread_create()`: todos os outros *threads*.
- Um *thread* termina se, p.ex.:
  - retorna da função inicial que executou (argumento de `pthread_create()` ou `main()`);
  - executa `pthread_exit()`.
- Um programa *multithreaded* termina se, p.ex.:
  - O *thread* principal terminar (ver acima);
  - Qualquer *thread* invocar a chamada ao sistema `_exit()`.

pthread_create()

```
int pthread_create(pthread_t *id,
                   const pthread_att_t attr,
                   void *(*start_fn)(void *), void *arg)
```

onde:

- `*id` será inicializado com a identidade do *thread* criado;
- `*attr` é uma estrutura de dados que configura o modo de funcionamento: pode ser inicializada com valores por omissão usando:

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

- `*start_fn` uma função com o seguinte protótipo:

  ```
  void *thr_fun(void *)
  ```

  que é a primeira função executada pelo *thread* a criar.
- `*arg` é a estrutura de dados a passar à função `thr_fun()`.

`pthread_create()`: exemplo

```
#include <pthread.h>
void *fun(void *arg) { /* Actually the ar- */
   ...                 /* gument is not used */
}
   ...
   pthread_attr_t attr;
   pthread_t      tid;
   ...
   pthread_attr_init(&attr); /* Initialize attr
                              * default values *
   pthread_create(&tid, &attr, fun, NULL);
   ...
```

- No caso geral, o último argumento de `pthread_create()` é o endereço duma estrutura de dados contendo os argumentos da função `fun()`.

`void *fun(void *arg)`

Permite definir praticamente qualquer função.

- Para evitar avisos (*warnings*) do `gcc` o mais fácil é definir um tipo de apontador para uma função.

```
#include <pthread.h>
typedef void *(thr_fun_t)(void *arg);
   int *fun(int *arg) {
        ...
   }
   ...
   pthread_attr_t attr;
   pthread_t      tid;
   int            thr_arg;
   ...
   pthread_attr_init(&attr); /* Initialize attr */
   pthread_create(&tid, &attr, (thr_fun_t *) fun,
                  (void *)&thr_arg);
   ...
```

Múltiplos *Threads*

Normalmente applicações *multithreaded* usam mais do que 2 *threads*

- É necessário alocar variáveis diferentes para cada *thread*.

```
#include <pthread.h>
#define T 3 /* number of threads */
typedef void *(thr_fun_t)(void *arg);
    ...
    pthread_attr_t attr[T];
    pthread_t      tid[T];
    int            thr_arg[T];
    ...
    for( i = 0; i < T; i++ ) {
        pthread_attr_init(&attr); /* Initialize attr */
        pthread_create(&(tid[i]), &(attr[i]),
                       (thr_fun_t *) fun,
                       (void *)&(thr_arg[i]));
    }
    ...
```

Modos de Partilha do CPU em *libpthread*

- *libpthread* suporta 2 modos da partilha do CPU:

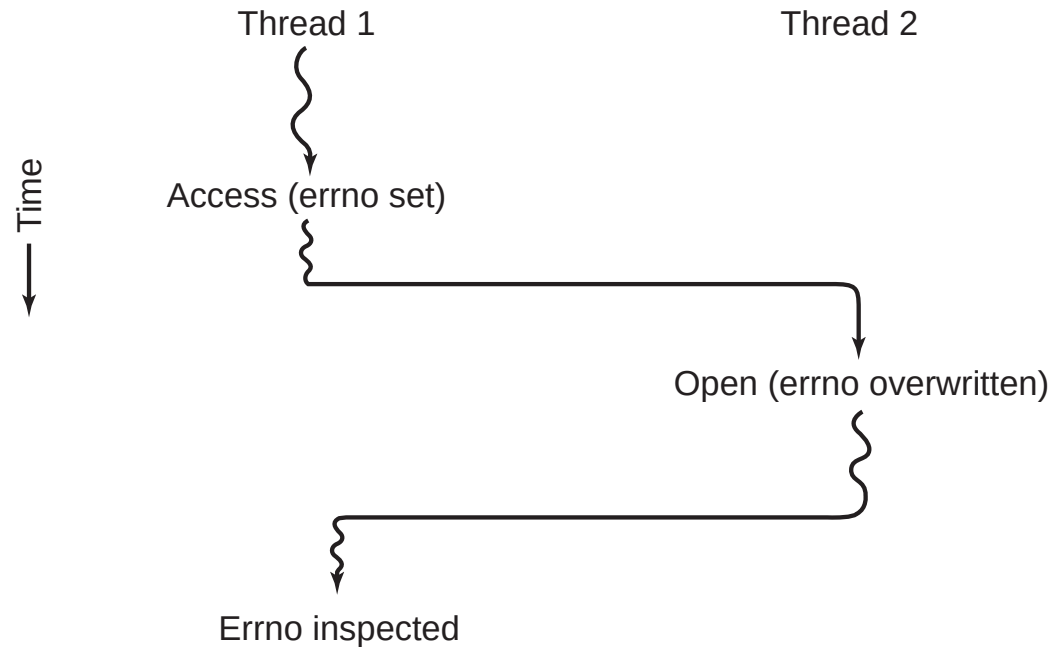
  `PTHREAD_SCOPE_PROCESS` *threads* disputam o CPU apenas com outros *threads* do mesmo processo:
  - *user-level threads* implica este modo de escalonamento.

  `PTHREAD_SCOPE_SYSTEM` *threads* disputam o CPU com todos os *threads* no sistema
  - impossível de implementar com *user-level threads*: SO não conhece todos os *threads* em execução;
  - modo *natural* quando se usa *kernel-level threads*.
- Note-se que este atributo pode ser configurado por cada *thread*, usando `pthread_attr_setscope()`.
  - Este atributo deve ser inicializado antes de o passar como argumento a `pthread_create()`

Sumário

Programação com Múltiplos *Threads*

- Código escrito para processos com um único *thread* raramente funciona correctamente com múltiplos *threads*:
  - variáveis globais:



  - funções não reentrantes;
  - concorrência (*race conditions*).
- Esta observação aplica-se também a código das bibliotecas, incluindo a "C standard library":

  Com `gcc`, deve usar-se a opção `-pthread`

Sumário

Leitura Adicional

Sistemas Operativos

- Secção 3.6.4: *Tarefas - Interface POSIX*

Modern Operating Systems, 2nd. Ed.

- Secção 2.2.8: *Making Single-Threaded Code Multithreaded*

Operating Systems Concepts

- Secção 4.3: *Thread Libraries* (only 4.3.1)
- Secção 4.4: *Threading Issues* (for your education)