FOLHA DE PROBLEMAS Nº 6

Criação e terminação de *threads*

---

1. – Criação e escalonamento de *threads*. Passagem de parâmetros a uma *thread*.

Considere o seguinte programa :

```
// PROGRAMA p01.c
#include <stdio.h>
#include <unistd.h>
#include <pthread.h>

#define STDERR 2
#define NUMITER 10000

void * thrfunc(void * arg)
{
  int i;

  fprintf(stderr, "Starting thread %s\n", (char *) arg);
  for (i = 1; i <= NUMITER; i++) write(STDERR,arg,1);
  return NULL;
}

int main()
{
  pthread_t ta, tb;

  pthread_create(&ta, NULL, thrfunc, "1");
  pthread_create(&tb, NULL, thrfunc, "2");

  pthread_join(ta, NULL);
  pthread_join(tb, NULL);

  return 0;
}
```

a) Execute o programa várias vezes e verifique que as *threads* executam alternadamente. Se necessário, aumente o número de iterações executadas por cada *thread* até que isso aconteça.

b) Altere o programa por forma a que o parâmetro das *threads* seja uma variável, em vez de ser uma constante. Implemente duas versões do programa, uma usando uma variável de tipo `char` e outra usando uma variável de tipo `int` para guardar o parâmetro. Identifique um problema que pode acontecer caso seja usada a mesma variável para guardar o parâmetro de cada uma das *threads*.

2. – Problemas de sincronização no acesso a variáveis globais. Recolha de resultados de uma *thread*.

Altere o programa do problema 1 por forma a que, conjuntamente, as *threads* escrevam um total de N (ex: 50000) caracteres. Para o efeito, defina uma variável global, inicializada a N, que vai sendo decrementada por cada *thread*, cada vez que escreve um carácter, até chegar a zero. Cada *thread* deve contabilizar e devolver à *thread* principal (`main` *thread*) o número de caracteres que escreveu; a *thread* principal deve apresentar esses números no ecrã. Verifique se o total dos caracteres escritos por ambas as

threads é N. Identifique o problema que pode ocorrer (uma solução para este problema será estudada no capítulo sobre sincronização de processos/*threads*).

3. – Passagem de parâmetros . Cuidado …!

Analise, compile e execute o seguinte programa:

```
// PROGRAMA p03.c
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>

#define NUM_THREADS 10

void *PrintHello(void *threadnum)
{
   printf("Hello from thread no. %d!\n", *(int *) threadnum);
   pthread_exit(NULL);
}

int main()
{
   pthread_t threads[NUM_THREADS];
   int t;
   for(t=0; t< NUM_THREADS; t++){
      printf("Creating thread %d\n", t);
      pthread_create(&threads[t], NULL, PrintHello, (void *)&t);
   }
   pthread_exit(0);
}
```

Numa das execuções deste programa obteve-se a seguinte saída:

```
Creating thread 0
Hello from thread no. 1!
Creating thread 1
Creating thread 2
Hello from thread no. 2!
Creating thread 3
Creating thread 4
Hello from thread no. 4!
Hello from thread no. 4!
Creating thread 5
Hello from thread no. 5!
Creating thread 6
Creating thread 7
Hello from thread no. 7!
Hello from thread no. 7!
Creating thread 8
Hello from thread no. 8!
Creating thread 9
Hello from thread no. 10!
Hello from thread no. 10!
```

a) Interprete os resultados.

b) Indique a correcção necessária para que os resultados sejam os aparentemente pretendidos.

c) Verifique o que acontece se substituir a chamada `pthread_exit()` da função `main()` por `exit()`. Explique.

4. – Criação e terminação de *threads*. Passagem de parâmetros e recolha de resultados.

a) Escreva um programa que crie N novas *threads*. Cada *thread* deve "dormir" durante um segundo, escrever a sua *TID* e terminar retornando um número inteiro igual ao seu número de ordem (1..N) que lhe deve ser passado com parâmetro.

b) Verifique o que acontece se a *thread* principal não esperar pelo fim das outras *threads* e terminar com `exit()` ou `return`.

c) Altere o programa por forma a que a *thread* principal escreva o valor retornado por cada uma das outras *threads*.

5. – Programa *multithreaded.* Passagem de parâmetros e recolha de resultados.

a) Escreva um programa que leia 2 números do teclado e crie 4 *threads* que façam, respectivamente, a soma, a subtracção, o produto e o quociente dos números lidos. Os operandos devem ser passados como parâmetros às *threads* (usar duas formas diferentes de o fazer, através de um *array* e de uma *struct*). O valor dos operandos e o resultado de cada operação devem ser apresentados no ecrã pela *thread* que executou cada operação.

b) Alterar o programa anterior por forma a que os resultados sejam apresentados pela main *thread*.

6. – Aplicação cliente-servidor muito simples, com servidor *multithreaded*

Implemente um aplicação cliente-servidor, constituída por um programa servidor, *multithreaded*, em que o servidor executa operações aritméticas (!) que lhe são solicitadas através de um *FIFO* pelos clientes, sendo a resposta enviada a cada cliente através de outro *FIFO*. Os pedidos de execução de uma operação são recebidos através de um *FIFO* de nome `fifo_req`, criado pelo servidor. Cada pedido deve conter, para além dos operandos, o nome de outro *FIFO*, de nome `fifo_ans_XXXXX`, criado pelo cliente, em que XXXXX representa a PID do cliente, através do qual deve ser enviado, ao cliente, o resultado da operação. O processo-servidor deve manter-se em funcionamento até que os números a processar sejam ambos iguais a zero, situação em que não deve efectuar qualquer cálculo, mas apenas deve destruir o *FIFO* `fifo_req`. O *FIFO* `fifo_ans_XXXXX` deverá ser destruído pelo cliente que o criou, quando não pretender solicitar a execução de mais operações. Sugestão: junte e adapte o código do problema 3 desta folha e o do problema 7 da folha nº 5.

7. – Processos *multithreaded* (1)

Escreva um programa que copie, "em paralelo", todos os ficheiros regulares de um directório para outro já existente; os sub-directórios não devem ser considerados. Os directórios devem ser passados como argumentos do programa. Por cada ficheiro regular deverá ser criada uma nova *thread* que o copia do directório original para o directório de destino. Sugestão: recorra ao código já desenvolvido anteriormente para determinar os ficheiros comuns de um directório e para copiar um ficheiro para outro.

8. – Processos *multithreaded* (2)

a) Escreva um programa que faça a pesquisa de uma *string* num ficheiro de texto. A *string* e o nome do ficheiro devem ser passados como argumentos da linha de comandos. O programa deve conter uma função cujos parâmetros sejam a *string* e o nome do ficheiro e que devolva o(s) número(s) da(s) linha(s) em que a *string* foi encontrada.

b) Escreva um programa *multithreaded* que faça uma pesquisa "em paralelo" de uma *string* em vários ficheiros de texto. O programa deverá chamar-se `grep_mt`. A *string* a pesquisar, bem como os nomes dos ficheiros de texto, devem ser passados como argumentos da linha de comando.

Exemplo: `grep_mt "texto a pesquisar" a.txt b.txt c.txt`

O programa deverá recorrer à função desenvolvida na alínea anterior, a qual será constituída como *thread* para procurar a *string* em cada um dos ficheiros passados na linha de comando. A *thread* principal deverá escrever os resultados da pesquisa, indicando, para cada ficheiro, se a *string* foi ou não encontrada e, em caso afirmativo, em que linhas.

9. – Processos *multithreaded* (3)

Pretende-se um programa *multithreaded* para fazer a multiplicação de matrizes. As duas matrizes a multiplicar devem estar guardadas num ficheiro de texto. A 1ª linha do ficheiro contém as dimensões da 1ª matriz (separadas por espaço), seguindo-se-lhe os valores dos seus elementos, linha a linha (dentro de cada linha, as colunas são separadas por espaço); após a 1ª matriz, segue-se imediatamente a 2ª, com o mesmo formato.

Exemplo :

matrizes:	ficheiro:	comentários:
$\begin{bmatrix}1 & 1 & 1\end{bmatrix}$ e $\begin{bmatrix}2 & 2\\ 2 & 2\\ 2 & 2\end{bmatrix}$	1 3 1 1 1 3 2 2 2 2 2 2 2	n_linhas n_colunas (matriz 1) linha 1 n_linhas n_colunas (matriz 2) linha 1 linha 2 linha 3

O resultado deverá também ser escrito num ficheiro de texto, com o mesmo formato. Os nomes dos ficheiros são passados como argumentos da linha de comando. Os elementos das matrizes de entrada deverão ser previamente lidos para memória, e o espaço para a matriz resultado também deverá ser previamente alocado (tudo pela *thread* principal).

a) Faça uma versão em que o número de *threads* seja igual ao número de colunas do resultado.

b) Faça uma 2.a versão em que o número de *threads* seja igual ao número de elementos do resultado.

c) Faça uma 3.a versão em que o número de *threads* seja fixo, sendo esse número passado como argumento na linha de comando. A *thread* principal deverá distribuir à partida os elementos a calcular pelas *threads* trabalhadoras, passando essa informação através do respectivo parâmetro de entrada.

d) Tente fazer uma nova versão, partindo da anterior, em que cada *thread* trabalhador vai buscar o próximo elemento do resultado a calcular a duas variáveis globais, que deverá actualizar. O que poderá correr mal nesta versão ?

10. – Processos *multithreaded* (4)

Escreva um programa *multithreaded* para a ordenação de um *array* de valores. O *array* deve ser dividido num número de partes que é uma potência de 2 (passado como parâmetro ao programa). Para cada parte deve criar uma nova *thread* que faz a ordenação dessa parte, usando a função da biblioteca standard `qsort()`. As partes ordenadas devem depois ser fundidas aos pares, usando também *threads* diferentes. A fusão deverá usar um algoritmo standard de *merge*.