Laboratório 1 - Computação Concorrente

Tiago Santos Martins de Macedo

116022689

Atividade 1

Segue abaixo a saída do terminal:

```
$ gcc cods-mod1-lab1/hello.c -lpthread -o hello

$ ./hello
--Cria a thread 0
--Cria a thread 1
Hello World
--Cria a thread 2
Hello World
--Cria a thread 3
Hello World
--Cria a thread 4
Hello World
--Cria a thread 5
Hello World
Hello World
--Cria a thread 6
--Cria a thread 7
--Cria a thread 8
Hello World
--Cria a thread 9
Hello World
Hello World
--Thread principal terminou
Hello World

$ ./hello
--Cria a thread 0
--Cria a thread 1
Hello World
Hello World
--Cria a thread 2
--Cria a thread 3
Hello World
--Cria a thread 4
Hello World
--Cria a thread 5
--Cria a thread 6
--Cria a thread 7
--Cria a thread 8
Hello World
--Cria a thread 9
Hello World
```

```
Hello World
Hello World
--Thread principal terminou
Hello World
Hello World
```

O programa `hello` foi executado mais vezes. Observa-se que a criação das threads ocorre sequencialmente, sempre na mesma ordem, mas o momento em que cada thread imprime "Hello World" varia a cada execução.

Quanto à pergunta "Ha mudanças na ordem de execução das threads?", a resposta é, sem dúvida, "muito frequentemente sim", apesar dessa mudança não ser aparente no programa `hello.c` pois todas as threads escrevem a mesma coisa na tela. Na atividade 2, resolveremos esse problema.

Essa mudança ocorre porque a ordem da execução das threads não é necessariamente a ordem de sua criação, e sim é determinada pelo sistema operacional em tempo de execução.

Atividade 2

```
$ gcc cods-mod1-lab1/hello_arg.c -lpthread -o arg

$ ./arg
--Cria a thread 0
--Cria a thread 1
Hello World da thread: 0
Hello World da thread: 1
--Cria a thread 2
--Cria a thread 3
Hello World da thread: 2
--Cria a thread 4
--Cria a thread 5
Hello World da thread: 4
--Cria a thread 6
Hello World da thread: 5
--Cria a thread 7
--Cria a thread 8
Hello World da thread: 7
--Cria a thread 9
Hello World da thread: 6
Hello World da thread: 8
Hello World da thread: 9
--Thread principal terminou
Hello World da thread: 3

$ ./arg
--Cria a thread 0
--Cria a thread 1
Hello World da thread: 0
--Cria a thread 2
Hello World da thread: 1
--Cria a thread 3
--Cria a thread 4
Hello World da thread: 3
--Cria a thread 5
```

```
Hello World da thread: 4
--Cria a thread 6
--Cria a thread 7
--Cria a thread 8
--Cria a thread 9
Hello World da thread: 5
--Thread principal terminou
Hello World da thread: 7
Hello World da thread: 9
Hello World da thread: 8
Hello World da thread: 6
Hello World da thread: 2
```

Como dito anteriormente, agora que cada thread escreve na tela seu número (passada a ela como argumento), podemos observar que sua execução ocorre fora de ordem.

Foi necessário alocar espaço para o argumento de cada thread (na forma do array `tid_local[NTHREADS]`) para cumprir as especificações da função `pthread_create`. De acordo com ela, o argumento da nova thread é um ponteiro que aponta para a informação que deseja-se transmití-la. Assim, qualquer variável passada à thread será passada por referência, não por cópia. Isso significa que, se a variável `thread` (o contador de iteração do loop principal do programa) tivesse sido usada, todas as threads terial recebido referência ao mesmo valor, e não era essa a intenção do programa.

Note que, tecnicamente, poderíamos ter forçado uma passagem de argumento por cópia, sem precisarmos portanto do array `tid_local`. Numa máquina de 32 bits, poderíamos "disfarçar" a variável `thread` de ponteiro fazendo um typecast ao passá-lo para `pthread_create` e, dentro da nova thread, fazendo um typecast de volta. O código ficaria assim:

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

#define NTHREADS  10

void *PrintHello (void* arg) {
    int idThread = (int) arg; // Typecast de volta para inteiro
    printf("Hello World da thread: %d\n", idThread);
    pthread_exit(NULL);
}

int main() {
    pthread_t tid_sistema[NTHREADS];
    int thread; //variavel auxiliar
//  int tid_local[NTHREADS];

    for(thread=0; thread<NTHREADS; thread++) {
        printf("--Cria a thread %d\n", thread);
        if (pthread_create( &tid_sistema[thread],
                            NULL,
                            PrintHello,
                            (void*) thread) // Typecast para ponteiro
           ) {
            printf("--ERRO: pthread_create()\n"); exit(-1);
        }
    }
```

```
    printf("--Thread principal terminou\n");
    pthread_exit(NULL);
}
```

Abaixo, o resultado desse experimento:

```
$ gcc hello_arg_typecast.c -lpthread -o arg
arg.c: In function ‘PrintHello’:
arg.c:14:18: warning: cast from pointer to integer of different size [-Wpointer-to-int-cas ]
   14 |   int idThread = (int) arg;
      |                  ^
arg.c: In function ‘main’:
arg.c:29:64: warning: cast to pointer from integer of different size [-Wint-to-pointer-cas ]
   29 |     if (pthread_create(&tid_sistema[thread], NULL, PrintHello, (void*) thread)) {
      |                                                                ^

$ ./arg
--Cria a thread 0
--Cria a thread 1
Hello World da thread: 0
--Cria a thread 2
--Cria a thread 3
Hello World da thread: 2
--Cria a thread 4
Hello World da thread: 3
--Cria a thread 5
Hello World da thread: 4
--Cria a thread 6
Hello World da thread: 5
--Cria a thread 7
--Cria a thread 8
Hello World da thread: 6
--Cria a thread 9
Hello World da thread: 1
Hello World da thread: 7
Hello World da thread: 8
--Thread principal terminou
Hello World da thread: 9
```

Repare que, apesar de o código ter funcionado, o compilador reclamou. Isso se deve ao fato de que meu computador é de 64bits, e assim o tamanho dos ponteiros, 8 bytes, é incompatível com o tamanho dos inteiros. Essa falta de portabilidade é exatamente o motivo pelo qual esse método é uma má ideia.

Atividade 3

```
$ gcc cods-mod1-lab1/hello_args.c -lpthread -o args

$ ./args
--Aloca e preenche argumentos para thread 0
--Cria a thread 0
```

```
--Aloca e preenche argumentos para thread 1
--Cria a thread 1
--Aloca e preenche argumentos para thread 2
--Cria a thread 2
Sou a thread 0 de 10 threads
Sou a thread 1 de 10 threads
--Aloca e preenche argumentos para thread 3
--Cria a thread 3
--Aloca e preenche argumentos para thread 4
--Cria a thread 4
Sou a thread 3 de 10 threads
Sou a thread 2 de 10 threads
--Aloca e preenche argumentos para thread 5
--Cria a thread 5
--Aloca e preenche argumentos para thread 6
--Cria a thread 6
Sou a thread 5 de 10 threads
--Aloca e preenche argumentos para thread 7
Sou a thread 6 de 10 threads
--Cria a thread 7
--Aloca e preenche argumentos para thread 8
--Cria a thread 8
--Aloca e preenche argumentos para thread 9
--Cria a thread 9
Sou a thread 8 de 10 threads
Sou a thread 9 de 10 threads
--Thread principal terminou
Sou a thread 7 de 10 threads
Sou a thread 4 de 10 threads

$ ./args
--Aloca e preenche argumentos para thread 0
--Cria a thread 0
--Aloca e preenche argumentos para thread 1
--Cria a thread 1
--Aloca e preenche argumentos para thread 2
--Cria a thread 2
Sou a thread 0 de 10 threads
--Aloca e preenche argumentos para thread 3
--Cria a thread 3
Sou a thread 1 de 10 threads
--Aloca e preenche argumentos para thread 4
--Cria a thread 4
Sou a thread 2 de 10 threads
Sou a thread 3 de 10 threads
--Aloca e preenche argumentos para thread 5
--Cria a thread 5
Sou a thread 4 de 10 threads
--Aloca e preenche argumentos para thread 6
--Cria a thread 6
Sou a thread 5 de 10 threads
--Aloca e preenche argumentos para thread 7
--Cria a thread 7
Sou a thread 6 de 10 threads
Sou a thread 7 de 10 threads
--Aloca e preenche argumentos para thread 8
--Cria a thread 8
--Aloca e preenche argumentos para thread 9
```

```
--Cria a thread 9
Sou a thread 8 de 10 threads
--Thread principal terminou
Sou a thread 9 de 10 threads
```

Sim, o código funcionou como esperado.

Atividade 4

```
$ gcc cods-mod1-lab1/hello_join.c -lpthread -o join

$ ./join
--Aloca e preenche argumentos para thread 0
--Cria a thread 0
--Aloca e preenche argumentos para thread 1
--Cria a thread 1
--Aloca e preenche argumentos para thread 2
--Cria a thread 2
Sou a thread 0 de 10 threads
Sou a thread 1 de 10 threads
--Aloca e preenche argumentos para thread 3
--Cria a thread 3
--Aloca e preenche argumentos para thread 4
--Cria a thread 4
Sou a thread 2 de 10 threads
--Aloca e preenche argumentos para thread 5
--Cria a thread 5
Sou a thread 3 de 10 threads
--Aloca e preenche argumentos para thread 6
--Cria a thread 6
Sou a thread 4 de 10 threads
--Aloca e preenche argumentos para thread 7
--Cria a thread 7
Sou a thread 5 de 10 threads
--Aloca e preenche argumentos para thread 8
--Cria a thread 8
--Aloca e preenche argumentos para thread 9
--Cria a thread 9
Sou a thread 7 de 10 threads
Sou a thread 9 de 10 threads
Sou a thread 6 de 10 threads
Sou a thread 8 de 10 threads
--Thread principal terminou

$ ./join
--Aloca e preenche argumentos para thread 0
--Cria a thread 0
--Aloca e preenche argumentos para thread 1
--Cria a thread 1
--Aloca e preenche argumentos para thread 2
--Cria a thread 2
Sou a thread 1 de 10 threads
Sou a thread 0 de 10 threads
--Aloca e preenche argumentos para thread 3
--Cria a thread 3
```

```
--Aloca e preenche argumentos para thread 4
--Cria a thread 4
Sou a thread 3 de 10 threads
--Aloca e preenche argumentos para thread 5
--Cria a thread 5
--Aloca e preenche argumentos para thread 6
--Cria a thread 6
Sou a thread 5 de 10 threads
--Aloca e preenche argumentos para thread 7
--Cria a thread 7
--Aloca e preenche argumentos para thread 8
--Cria a thread 8
--Aloca e preenche argumentos para thread 9
--Cria a thread 9
Sou a thread 7 de 10 threads
Sou a thread 4 de 10 threads
Sou a thread 8 de 10 threads
Sou a thread 6 de 10 threads
Sou a thread 2 de 10 threads
Sou a thread 9 de 10 threads
--Thread principal terminou
```

A diferença desta versão do programa é que, devido ao loop de `pthread_join`, podemos ter certeza de que a thread principal (aquela com a função `main`) esperará o retorno de todas as outras threads antes de imprimir "--Thread principal terminou" na tela.

Atividade 5

O arquivo de código fonte se chama `duplo_incremento.c`. Segue abaixo a sua saída no terminal:

```
vector = [ 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 ]
Incrementando de 0 até 29.
Incrementando de 30 até 59.
Thread 1 retornado.
Thread 2 retornado.
vector = [ 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 ]
```