

Laboratório 1 - Computação Concorrente

Tiago Santos Martins de Macedo

116022689

Atividade 1

Segue abaixo a saída do terminal:

```
1  $ gcc cods-mod1-lab1/hello.c -lpthread -o hello
2
3  $ ./hello
4  --Cria a thread 0
5  --Cria a thread 1
6  Hello World
7  --Cria a thread 2
8  Hello World
9  --Cria a thread 3
10 Hello World
11 --Cria a thread 4
12 Hello World
13 --Cria a thread 5
14 Hello World
15 Hello World
16 --Cria a thread 6
17 --Cria a thread 7
18 --Cria a thread 8
19 Hello World
20 --Cria a thread 9
21 Hello World
22 Hello World
23 --Thread principal terminou
24 Hello World
25
26 $ ./hello
27 --Cria a thread 0
28 --Cria a thread 1
29 Hello World
30 Hello World
31 --Cria a thread 2
32 --Cria a thread 3
33 Hello World
34 --Cria a thread 4
35 Hello World
36 --Cria a thread 5
37 --Cria a thread 6
38 --Cria a thread 7
39 --Cria a thread 8
40 Hello World
41 --Cria a thread 9
42 Hello World
```

```
43 | Hello World
44 | Hello World
45 | --Thread principal terminou
46 | Hello World
47 | 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

```
1 | $ gcc cods-mod1-lab1/hello_arg.c -lpthread -o arg
2 |
3 | $ ./arg
4 | --Cria a thread 0
5 | --Cria a thread 1
6 | Hello World da thread: 0
7 | Hello World da thread: 1
8 | --Cria a thread 2
9 | --Cria a thread 3
10 | Hello World da thread: 2
11 | --Cria a thread 4
12 | --Cria a thread 5
13 | Hello World da thread: 4
14 | --Cria a thread 6
15 | Hello World da thread: 5
16 | --Cria a thread 7
17 | --Cria a thread 8
18 | Hello World da thread: 7
19 | --Cria a thread 9
20 | Hello World da thread: 6
21 | Hello World da thread: 8
22 | Hello World da thread: 9
23 | --Thread principal terminou
24 | Hello World da thread: 3
25 |
26 | $ ./arg
27 | --Cria a thread 0
28 | --Cria a thread 1
29 | Hello World da thread: 0
30 | --Cria a thread 2
31 | Hello World da thread: 1
32 | --Cria a thread 3
33 | --Cria a thread 4
34 | Hello World da thread: 3
35 | --Cria a thread 5
```

```

36 Hello World da thread: 4
37 --Cria a thread 6
38 --Cria a thread 7
39 --Cria a thread 8
40 --Cria a thread 9
41 Hello World da thread: 5
42 --Thread principal terminou
43 Hello World da thread: 7
44 Hello World da thread: 9
45 Hello World da thread: 8
46 Hello World da thread: 6
47 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 transmiti-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 teriam 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:

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <pthread.h>
4
5  #define NTHREADS 10
6
7  void *PrintHello (void* arg) {
8      int idThread = (int) arg; // Typecast de volta para inteiro
9      printf("Hello World da thread: %d\n", idThread);
10     pthread_exit(NULL);
11 }
12
13 int main() {
14     pthread_t tid_sistema[NTHREADS];
15     int thread; //variavel auxiliar
16     // int tid_local[NTHREADS];
17
18     for(thread=0; thread<NTHREADS; thread++) {
19         printf("--Cria a thread %d\n", thread);
20         if (pthread_create( &tid_sistema[thread],
21                             NULL,
22                             PrintHello,
23                             (void*) thread) // Typecast para ponteiro
24             ) {
25             printf("--ERRO: pthread_create()\n"); exit(-1);
26         }
27     }

```

```

28     printf("--Thread principal terminou\n");
29     pthread_exit(NULL);
30 }

```

Abaixo, o resultado desse experimento:

```

1  $ gcc hello_arg_typecast.c -lpthread -o arg
2  arg.c: In function 'PrintHello':
3  arg.c:14:18: warning: cast from pointer to integer of different size [-Wpointer-to-int-cas]
4      14 |     int idThread = (int) arg;
5          |                  ^
6  arg.c: In function 'main':
7  arg.c:29:64: warning: cast to pointer from integer of different size [-Wint-to-pointer-cas]
8      29 |     if (pthread_create(&tid_sistema[thread], NULL, PrintHello,
9          |                               ^
10
11  $ ./arg
12  --Cria a thread 0
13  --Cria a thread 1
14  Hello World da thread: 0
15  --Cria a thread 2
16  --Cria a thread 3
17  Hello World da thread: 2
18  --Cria a thread 4
19  Hello World da thread: 3
20  --Cria a thread 5
21  Hello World da thread: 4
22  --Cria a thread 6
23  Hello World da thread: 5
24  --Cria a thread 7
25  --Cria a thread 8
26  Hello World da thread: 6
27  --Cria a thread 9
28  Hello World da thread: 1
29  Hello World da thread: 7
30  Hello World da thread: 8
31  --Thread principal terminou
32  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

```

1  $ gcc cods-mod1-lab1/hello_args.c -lpthread -o args
2
3  $ ./args
4  --Aloca e preenche argumentos para thread 0
5  --Cria a thread 0

```

```
6  --Aloca e preenche argumentos para thread 1
7  --Cria a thread 1
8  --Aloca e preenche argumentos para thread 2
9  --Cria a thread 2
10 Sou a thread 0 de 10 threads
11 Sou a thread 1 de 10 threads
12 --Aloca e preenche argumentos para thread 3
13 --Cria a thread 3
14 --Aloca e preenche argumentos para thread 4
15 --Cria a thread 4
16 Sou a thread 3 de 10 threads
17 Sou a thread 2 de 10 threads
18 --Aloca e preenche argumentos para thread 5
19 --Cria a thread 5
20 --Aloca e preenche argumentos para thread 6
21 --Cria a thread 6
22 Sou a thread 5 de 10 threads
23 --Aloca e preenche argumentos para thread 7
24 Sou a thread 6 de 10 threads
25 --Cria a thread 7
26 --Aloca e preenche argumentos para thread 8
27 --Cria a thread 8
28 --Aloca e preenche argumentos para thread 9
29 --Cria a thread 9
30 Sou a thread 8 de 10 threads
31 Sou a thread 9 de 10 threads
32 --Thread principal terminou
33 Sou a thread 7 de 10 threads
34 Sou a thread 4 de 10 threads
35
36 $ ./args
37 --Aloca e preenche argumentos para thread 0
38 --Cria a thread 0
39 --Aloca e preenche argumentos para thread 1
40 --Cria a thread 1
41 --Aloca e preenche argumentos para thread 2
42 --Cria a thread 2
43 Sou a thread 0 de 10 threads
44 --Aloca e preenche argumentos para thread 3
45 --Cria a thread 3
46 Sou a thread 1 de 10 threads
47 --Aloca e preenche argumentos para thread 4
48 --Cria a thread 4
49 Sou a thread 2 de 10 threads
50 Sou a thread 3 de 10 threads
51 --Aloca e preenche argumentos para thread 5
52 --Cria a thread 5
53 Sou a thread 4 de 10 threads
54 --Aloca e preenche argumentos para thread 6
55 --Cria a thread 6
56 Sou a thread 5 de 10 threads
57 --Aloca e preenche argumentos para thread 7
58 --Cria a thread 7
59 Sou a thread 6 de 10 threads
60 Sou a thread 7 de 10 threads
61 --Aloca e preenche argumentos para thread 8
62 --Cria a thread 8
63 --Aloca e preenche argumentos para thread 9
```

```
64 --Cria a thread 9
65 Sou a thread 8 de 10 threads
66 --Thread principal terminou
67 Sou a thread 9 de 10 threads
```

Sim, o código funcionou como esperado.

Atividade 4

```
1 $ gcc cods-mod1-lab1/hello_join.c -lpthread -o join
2
3 $ ./join
4 --Aloca e preenche argumentos para thread 0
5 --Cria a thread 0
6 --Aloca e preenche argumentos para thread 1
7 --Cria a thread 1
8 --Aloca e preenche argumentos para thread 2
9 --Cria a thread 2
10 Sou a thread 0 de 10 threads
11 Sou a thread 1 de 10 threads
12 --Aloca e preenche argumentos para thread 3
13 --Cria a thread 3
14 --Aloca e preenche argumentos para thread 4
15 --Cria a thread 4
16 Sou a thread 2 de 10 threads
17 --Aloca e preenche argumentos para thread 5
18 --Cria a thread 5
19 Sou a thread 3 de 10 threads
20 --Aloca e preenche argumentos para thread 6
21 --Cria a thread 6
22 Sou a thread 4 de 10 threads
23 --Aloca e preenche argumentos para thread 7
24 --Cria a thread 7
25 Sou a thread 5 de 10 threads
26 --Aloca e preenche argumentos para thread 8
27 --Cria a thread 8
28 --Aloca e preenche argumentos para thread 9
29 --Cria a thread 9
30 Sou a thread 7 de 10 threads
31 Sou a thread 9 de 10 threads
32 Sou a thread 6 de 10 threads
33 Sou a thread 8 de 10 threads
34 --Thread principal terminou
35
36 $ ./join
37 --Aloca e preenche argumentos para thread 0
38 --Cria a thread 0
39 --Aloca e preenche argumentos para thread 1
40 --Cria a thread 1
41 --Aloca e preenche argumentos para thread 2
42 --Cria a thread 2
43 Sou a thread 1 de 10 threads
44 Sou a thread 0 de 10 threads
45 --Aloca e preenche argumentos para thread 3
46 --Cria a thread 3
```

```

47  --Aloca e preenche argumentos para thread 4
48  --Cria a thread 4
49  Sou a thread 3 de 10 threads
50  --Aloca e preenche argumentos para thread 5
51  --Cria a thread 5
52  --Aloca e preenche argumentos para thread 6
53  --Cria a thread 6
54  Sou a thread 5 de 10 threads
55  --Aloca e preenche argumentos para thread 7
56  --Cria a thread 7
57  --Aloca e preenche argumentos para thread 8
58  --Cria a thread 8
59  --Aloca e preenche argumentos para thread 9
60  --Cria a thread 9
61  Sou a thread 7 de 10 threads
62  Sou a thread 4 de 10 threads
63  Sou a thread 8 de 10 threads
64  Sou a thread 6 de 10 threads
65  Sou a thread 2 de 10 threads
66  Sou a thread 9 de 10 threads
67  --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:

```
1 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 ]  
2 Incrementando de 0 até 29.  
3 Incrementando de 30 até 59.  
4 Thread 1 retornado.  
5 Thread 2 retornado.  
6 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 ]
```