

RELATÓRIO DE LABORATÓRIO 3

SISTEMAS OPERACIONAIS

VICTOR MONEGO

ENGENHARIA ELETRÔNICA – UTFPR

9 DE ABRIL DE 2024

1.Introdução Geral

O seguinte relatório diz respeito ao Laboratório 03 de Sistemas Operacionais, focado na lógica de threads POSIX em UNIX.

Para a realização das análises a seguir, foi utilizada a plataforma WSL(Ubuntu) no Windows 11, e os códigos foram comentados e modificados usando o programa Notepad++.

2. Exercício 01: “thread-create.c”

O exerppto da figura 01 abaixo apresenta o código “thread-create.c” utilizado no exercício 1. Note que o código não é de autoria própria e as únicas alterações feitas foram os comentários a respeito dele.

```
9  #include <pthread.h>
10 #include <stdio.h>
11 #include <stdlib.h>
12 #include <unistd.h>
13
14 #define NUM_THREADS 16
15
16 void *threadBody (void *id)
17 {
18     long tid = (long) id ; // ID da thread
19
20     printf ("t%02ld: Olá!\n", tid) ;
21     sleep (3) ;
22     printf ("t%02ld: Tchau!\n", tid) ;
23     pthread_exit (NULL) ;
24 }
25
26 int main (int argc, char *argv[])
27 {
28     pthread_t thread [NUM_THREADS] ;
29     long i, status ;
30
31     for (i=0; i<NUM_THREADS; i++)
32     {
33         printf ("Main: criando thread %02ld\n", i) ;
34         status = pthread_create (&thread[i], NULL, threadBody, (void *) i) ;
35         if (status)
36         {
37             perror ("pthread_create") ;
38             exit (1) ;
39         }
40     }
41     printf ("Main: fim\n") ;
42     pthread_exit (NULL) ;
43
44 }
```

Figura 1: Código thread-create.c

Comentários:

- 16: função que denota o comportamento das threads criadas
- 18: ID do thread
- 19: imprime uma mensagem de saudação juntamente com o ID do thread
- 20: a função dorme por 3 segundos
- 21: a thread se despede
- 22: encerramento da thread
- 27: um vetor de threads é criado baseado no tamanho pré-estabelecido
- 28: declara uma variável para armazenar a iteração e outra para o valor de retorno da função de criação da thread
- 30: entramos em um laço de iteração
- 32: imprime uma mensagem para sinalizar a criação da próxima thread
- 33: status recebe o valor de retorno da criação da thread
- 34: se houve erro na criação dessa thread
- 36: imprime uma mensagem de erro
- 40: ao fim da CRIAÇÃO das threads, o programa imprime "fim"
- 41: encerra a thread do programa principal, mantendo as outras threads funcionando independentemente para terminarem sua execução
- 42: em resumo, a thread 0 é o programa principal nesse caso.

A imagem 02 abaixo indica o diagrama de tempo do programa acima.

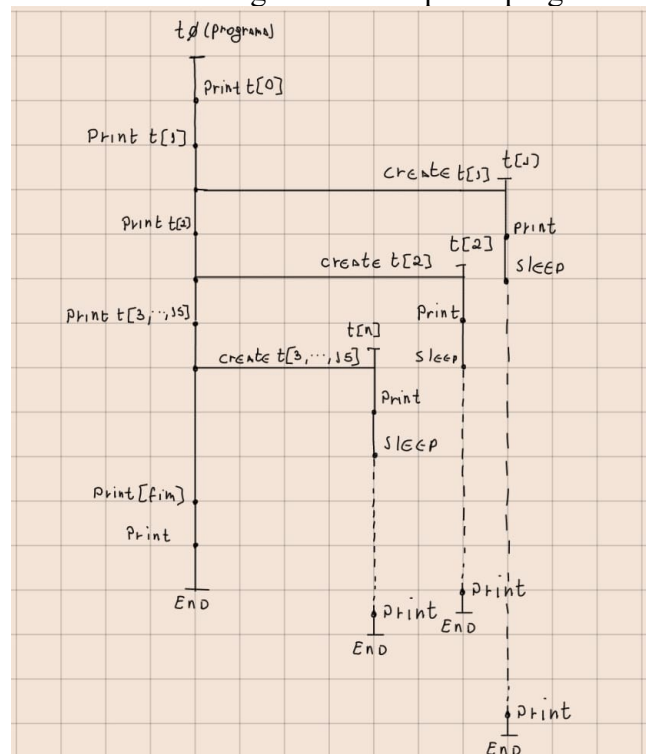
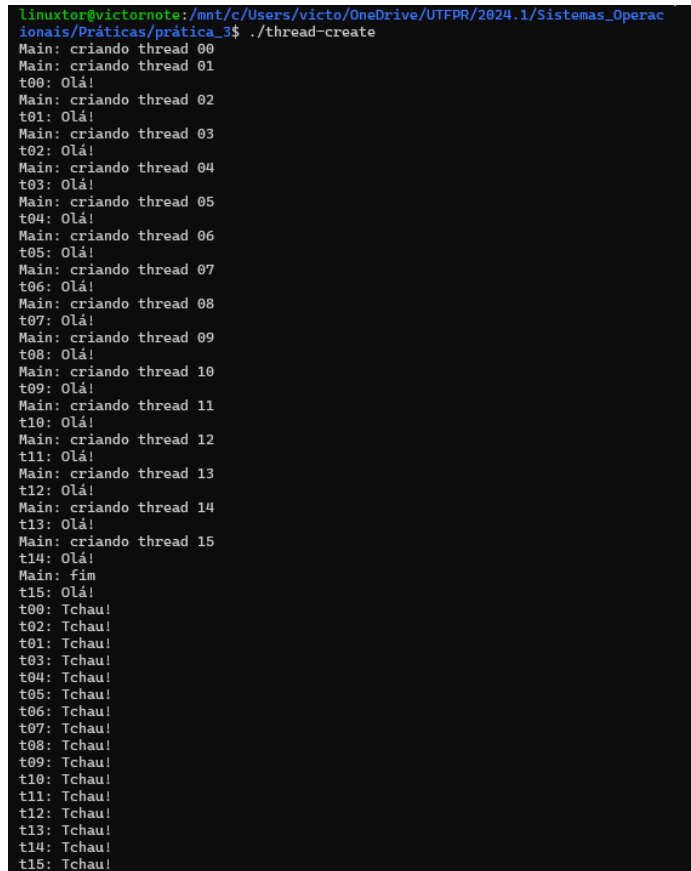


Figura 2: Diagrama de Tempo thread-create.c

É importante ressaltar que em uma situação em que são criadas 16 threads, é uma tarefa difícil desenhar um diagrama de tempo que ilustre detalhadamente todo o fluxo do programa. Tendo isso em mente, a intenção do diagrama é ilustrar que a thread 00 (o programa principal) se responsabiliza por criar todas as outras threads, portanto é a primeira thread a ser criada e também a primeira a ser destruída. Dessa forma, as outras threads funcionam de forma independente, e seguem o seu corpo de funcionamento normalmente.

E por fim, a imagem 03 indica como funciona o programa quando executado no prompt da WSL.



```
Linuxtor@victornote: /mnt/c/Users/victo/OneDrive/UTFPR/2024.1/Sistemas_Operacionais/Práticas/prática_3$ ./thread-create
Main: criando thread 00
Main: criando thread 01
t00: Olá!
Main: criando thread 02
t01: Olá!
Main: criando thread 03
t02: Olá!
Main: criando thread 04
t03: Olá!
Main: criando thread 05
t04: Olá!
Main: criando thread 06
t05: Olá!
Main: criando thread 07
t06: Olá!
Main: criando thread 08
t07: Olá!
Main: criando thread 09
t08: Olá!
Main: criando thread 10
t09: Olá!
Main: criando thread 11
t10: Olá!
Main: criando thread 12
t11: Olá!
Main: criando thread 13
t12: Olá!
Main: criando thread 14
t13: Olá!
Main: criando thread 15
t14: Olá!
Main: fim
t15: Olá!
t00: Tchau!
t02: Tchau!
t01: Tchau!
t03: Tchau!
t04: Tchau!
t05: Tchau!
t06: Tchau!
t07: Tchau!
t08: Tchau!
t09: Tchau!
t10: Tchau!
t11: Tchau!
t12: Tchau!
t13: Tchau!
t14: Tchau!
t15: Tchau!
```

Figura 3: Execução thread-create.c

Analisando o código, percebe-se que a criação e o desligamento das threads ocorre de forma que não existe ordem específica. A ordem exata de ativação e desativação das threads pode variar de execução para execução e depende de vários fatores, incluindo o sistema operacional, o escalonador de threads e a concorrência entre as próprias threads. O escalonador de threads decide quando cada thread é executada e em que ordem, e essa ordem pode não seguir exatamente a ordem de criação delas.

3. Exercício 02: “thread-join.c”

O exerpto da figura 04 abaixo apresenta o código “thread-join.c” utilizado no exercício 2. Note que o código não é de autoria própria. As alterações feitas foram os comentários.

```
9      #include <pthread.h>
10     #include <stdio.h>
11     #include <stdlib.h>
12     #include <unistd.h>
13
14     #define NUM_THREADS 16
15
16     void *threadBody (void *id)
17     {
18         long tid = (long) id ;
19
20         printf ("t%02ld: Olá!\n", tid) ;
21         sleep (3) ;
22         printf ("t%02ld: Tchau!\n", tid) ;
23         pthread_exit (NULL) ;
24     }
25
26     int main (int argc, char *argv[])
27     {
28         pthread_t thread [NUM_THREADS] ;
29         pthread_attr_t attr ;
30         long i, status ;
31
32         // para permitir a operação "join" sobre as threads
33         pthread_attr_init (&attr) ;
34         pthread_attr_setdetachstate (&attr, PTHREAD_CREATE_JOINABLE) ;
35
36         for (i=0; i<NUM_THREADS; i++)
37         {
38             printf ("Main: criando thread %02ld\n", i) ;
39             status = pthread_create (&thread[i], &attr, threadBody, (void *) i) ;
40             if (status)
41             {
42                 perror ("pthread_create") ;
43                 exit (1) ;
44             }
45         }
46
47         for (i=0; i<NUM_THREADS; i++)
48         {
49             printf ("Main: aguardando thread %02ld\n", i) ;
50             status = pthread_join (thread[i], NULL) ;
51             if (status)
52             {
53                 perror ("pthread_join") ;
54                 exit (1) ;
55             }
56         }
57         printf ("Main: fim\n") ;
58         pthread_attr_destroy (&attr) ;
59         pthread_exit (NULL) ;
60     }
```

Figura 4: thread-join.c

Comentários:

- 16: função que denota o comportamento das threads criadas
- 18: ID da thread
- 20: imprime uma mensagem de saudação juntamente com o ID da thread
- 21: a função dorme por 3 segundos
- 22: a thread se despede
- 23: encerramento da thread
- 28: um vetor de threads é criado baseado no tamanho pré estabelecido
- 33: torna a thread capaz de receber atributos
- 34: uma thread ser "joinable" significa que ela pode ser atrelada ao programa principal
- 38: imprime uma mensagem para sinalizar a criação da próxima thread
- 39: status recebe o valor de retorno da criação da thread
- 40: se houve erro na criação dessa thread
- 42: imprime uma mensagem de erro
- 49: sinaliza que o programa principal vai aguardar essa thread terminar
- 50: atrela a thread ao programa principal
- 51: se houver erro de join
- 53: imprime mensagem de erro
- 57: ao fim da CRIAÇÃO das threads, o programa imprime "fim"
- 58: limpa os atributos das threads
- 59: encerra a thread do programa principal, mantendo as outras threads funcionando independentemente para terminarem sua execução

Analizando o código, vemos a adição do parâmetro attr e a função pthread_join. Esses opcodes servem para tornar a thread capaz de se atrelar a outra. O parâmetro attr torna a thread capaz de absorver atributos, e a função Join efetivamente atrela uma thread a outra.

A imagem 05 abaixo indica o diagrama de tempo do programa acima.

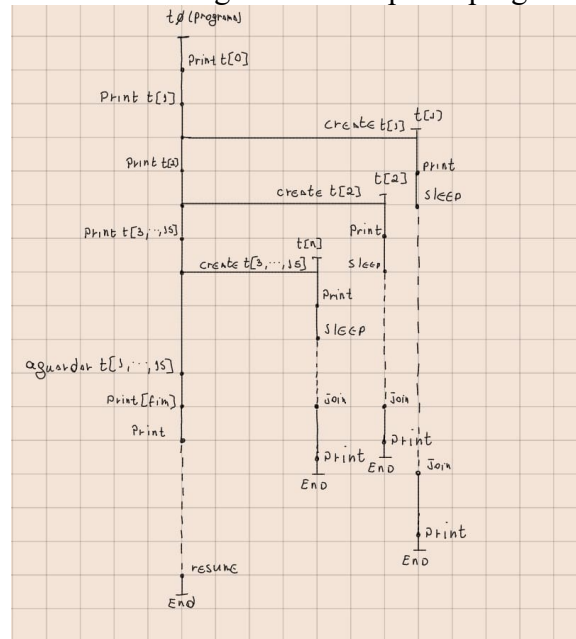


Figura 5: Diagrama de tempo thread-join.c

A intenção do diagrama é representar que ao utilizar as funções de atributo, e utilizar o comando Join nas threads, a thread principal aguarda todas as outras threads atreladas se desativarem, de forma que ela seja a ultima a ser desativada.

E por fim, a imagem 06 indica como funciona o programa quando executado no prompt da WSL.

```

$ cd /mnt/c/Users/victo/OneDrive/UFPR/2024-1/Sistemas_Operacionais/Práticas/prática_35 && ./thread-join
Main: criando thread 00
Main: criando thread 01
000: Olá!
Main: criando thread 02
001: Olá!
Main: criando thread 03
002: Olá!
Main: criando thread 04
003: Olá!
Main: criando thread 05
004: Olá!
Main: criando thread 06
005: Olá!
Main: criando thread 07
006: Olá!
Main: criando thread 08
007: Olá!
Main: criando thread 09
008: Olá!
Main: criando thread 10
009: Olá!
Main: criando thread 11
010: Olá!
Main: criando thread 12
011: Olá!
Main: criando thread 13
012: Olá!
Main: criando thread 14
013: Olá!
Main: criando thread 15
014: Olá!
Main: aguardando thread 00
015: Olá!
001: Tchau!
002: Tchau!
003: Tchau!
004: Tchau!
005: Tchau!
006: Tchau!
007: Tchau!
008: Tchau!
009: Tchau!
010: Tchau!
011: Tchau!
012: Tchau!
013: Tchau!
014: Tchau!
Main: aguardando thread 01
Main: aguardando thread 02
Main: aguardando thread 03
Main: aguardando thread 04
Main: aguardando thread 05
Main: aguardando thread 06
Main: aguardando thread 07
Main: aguardando thread 08
Main: aguardando thread 09
Main: aguardando thread 10
Main: aguardando thread 11
Main: aguardando thread 12
Main: aguardando thread 13
Main: aguardando thread 14
Main: aguardando thread 15
Main: fim

```

Figura 6: Execução thread-join.c

4. Exercício 03: “thread-print.c”

O exerpto da figura 07 abaixo apresenta o código “thread-print.c” utilizado no exercício 3. Note que o código não é de autoria própria. As alterações feitas foram os comentários

```
9      #include <pthread.h>
10     #include <stdio.h>
11     #include <stdlib.h>
12     #include <unistd.h>
13
14     #define NUM_THREADS 16
15
16     int x = 0 ;
17
18     void *threadBody (void *id)
19     {
20         long tid = (long) id ; // ID da thread
21
22         x++ ;
23         printf ("t%02ld: Olá! (x=%02d)\n", tid, x) ;
24         sleep (3) ;
25         x++ ;
26         printf ("t%02ld: Tchau! (x=%02d)\n", tid, x) ;
27         pthread_exit (NULL) ;
28     }
29
30     int main (int argc, char *argv[])
31     {
32         pthread_t thread [NUM_THREADS] ;
33         long i, status ;
34
35         for (i=0; i<NUM_THREADS; i++)
36         {
37             printf ("Main: criando thread %02ld\n", i) ;
38             status = pthread_create (&thread[i], NULL, threadBody, (void *) i) ;
39             if (status)
40             {
41                 perror ("pthread_create") ;
42                 exit (1) ;
43             }
44         }
45         printf ("Main: fim\n") ;
46         pthread_exit (NULL) ;
47     }
```

Figura 7: Código thread-print.c

Comentários:

- 16: variável global
- 18: função que denota o comportamento das threads criadas
- 20: ID da thread
- 22: incrementa a variável global em 1
- 23: a thread se identifica e reporta o valor de x
- 24: a função dorme por 3 segundos
- 25: incrementa a variável global em 1
- 26: a thread se despede e reporta o valor de x
- 27: a thread se extingue
- 32: criação do vetor de threads
- 33: cria a variável de iteração e de armazenamento do status da thread
- 37: sinaliza que uma thread está sendo criada
- 38: sinaliza o status da thread
- 39: se houve erro de criação das threads
- 41: imprime a mensagem de erro
- 45: ao fim da CRIAÇÃO das threads, o programa imprime "fim"
- 46: encerra a thread do programa principal, mantendo as outras threads funcionando independentemente para terminarem sua execução

A imagem 08 abaixo indica o diagrama de tempo do programa acima.

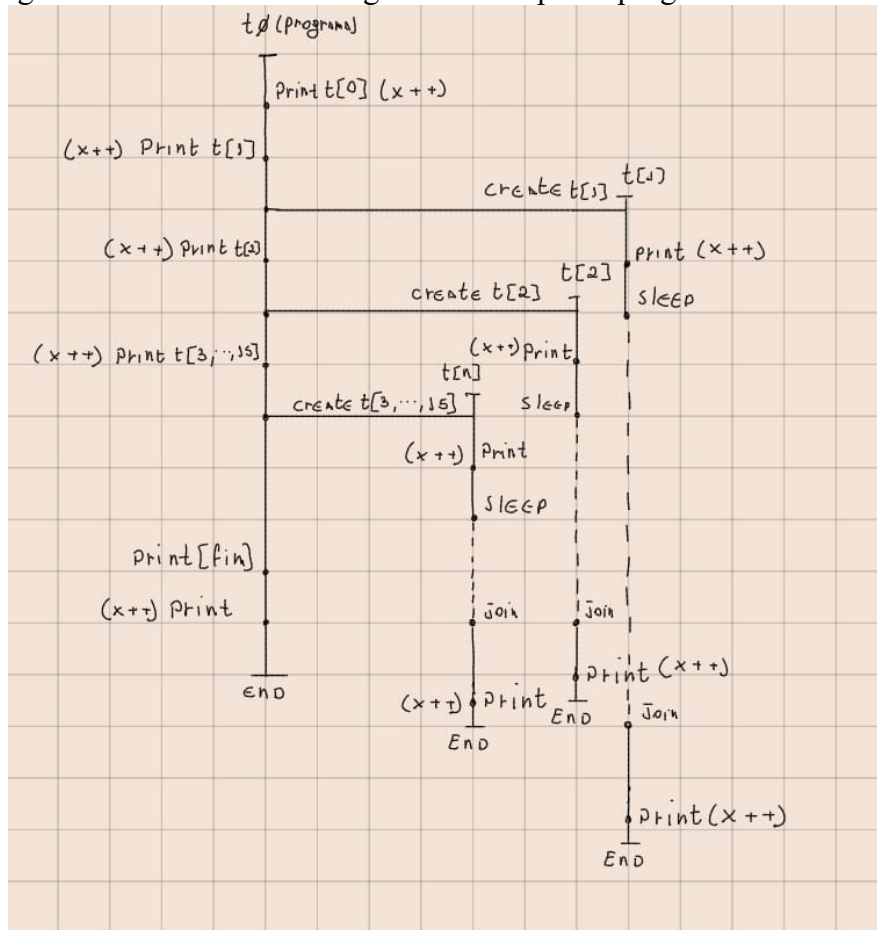


Figura 8: Diagrama de tempo thread-print.c

O diagrama nos mostra que cada thread incrementa uma variável global duas vezes durante sua execução. Como nesse contexto não foi utilizada a função Join, novamente todas as threads criadas se executam de forma independente com relação à thread principal (00)

E por fim, a imagem 10 indica como funciona o programa quando executado no prompt da WSL.

```
linuxtor@victornote:/mnt/c/Users/victo/OneDrive/UTFPR/2024.1/Sistemas_Operacionais/Thread/Thread-Print:
gcc -Wall thread-print.c -o thread-print -lpthread
linuxtor@victornote:/mnt/c/Users/victo/OneDrive/UTFPR/2024.1/Sistemas_Operacionais/Thread/Thread-Print:
./thread-print
Main: criando thread 00
Main: criando thread 01
t00: Olá! (x=01)
Main: criando thread 02
t01: Olá! (x=02)
Main: criando thread 03
t02: Olá! (x=03)
Main: criando thread 04
t03: Olá! (x=04)
Main: criando thread 05
t04: Olá! (x=05)
Main: criando thread 06
t05: Olá! (x=06)
Main: criando thread 07
t06: Olá! (x=07)
Main: criando thread 08
t07: Olá! (x=08)
Main: criando thread 09
t08: Olá! (x=09)
Main: criando thread 10
t09: Olá! (x=10)
Main: criando thread 11
t10: Olá! (x=11)
Main: criando thread 12
t11: Olá! (x=12)
Main: criando thread 13
t12: Olá! (x=13)
Main: criando thread 14
t13: Olá! (x=14)
Main: criando thread 15
t14: Olá! (x=15)
Main: fim
t15: Olá! (x=16)
t00: Tchau! (x=17)
t01: Tchau! (x=18)
t02: Tchau! (x=19)
t03: Tchau! (x=20)
t04: Tchau! (x=21)
t05: Tchau! (x=22)
t06: Tchau! (x=23)
t07: Tchau! (x=24)
t08: Tchau! (x=25)
t09: Tchau! (x=26)
t10: Tchau! (x=27)
t12: Tchau! (x=28)
t11: Tchau! (x=29)
t13: Tchau! (x=30)
t14: Tchau! (x=31)
t15: Tchau! (x=32)
```

Figura 10: Execução thread-print.c

Uma observação importante a se fazer é a maneira como a variável. Se fizéssemos um comparativo rápido com o laboratório anterior, “Criação de Processos em UNIX”, ao compilar um programa de incrementação utilizando o comando “fork()” para criar processos diferentes, perceberíamos que a principal diferença é que processos diferentes armazenaram sua própria versão da variável. Ao contrário do programa utilizando processos, as threads criadas se referem e compartilham a mesma variável, logo, toda incrementação é feita em apenas uma variável. Visto que cada thread incrementa duas vezes, ao final do programa x recebe 32 como valor.

Observações: Os códigos apresentados no relatório não são autorais, e são de autoria do Prof. Carlos Maziero, vide a seção “Referências”.

Referências:

- MAZIERO, C. **Criação de Processos em Unix**. Disponível em:
https://wiki.inf.ufpr.br/maziero/doku.php?id=so:criacao_de_processos. Acesso em: março de 2024
- MONEGO, V. A. **RELATÓRIO DE LABORATÓRIO 2**. Acesso em: abril de 2024