**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 exerpto 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.

Texto

Descrição gerada automaticamente

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.

Diagrama, Esquemático

Descrição gerada automaticamente

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.

Texto

Descrição gerada automaticamente

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 especifica. 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.

Texto

Descrição gerada automaticamente

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

Analisando 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.

Diagrama, Esquemático

Descrição gerada automaticamente

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.

Texto

Descrição gerada automaticamente

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

Texto

Descrição gerada automaticamente

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.

Texto preto sobre fundo branco

Descrição gerada automaticamente

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.

Uma imagem contendo Interface gráfica do usuário

Descrição gerada automaticamente

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