

## Atividade 3

### Threads

#### 1. Funcionamento do Thread

##### 1. Texto Explicativo sobre o item simulado.

Thread pode ser definido como um pequeno programa que trabalha como um subsistema, sendo uma forma de um processo se autodividir em duas ou mais tarefas. É o termo em inglês para Linha ou Encadeamento de Execução. Essas tarefas múltiplas podem ser executadas simultaneamente para rodar mais rápido do que um programa em um único bloco ou praticamente juntas, mas que são tão rápidas que parecem estar trabalhando em conjunto ao mesmo tempo (CANALTECH).

Threads no sistema operacional, é uma sequência de comandos que é executada em um programa ou processo, se houver mais de uma, serão duas sequências rodando em paralelo no mesmo processo. Geralmente um processo começa com uma única thread, que se subdivide criando uma segunda e assim por diante.

A thread pode ser como um subprocesso de um processo, que permite compartilhar a sua área de dados com o programa ou outras threads. A execução de uma thread pode passar por quatro estados, que são eles: novo, executável, bloqueado e encerrado.

O estado novo é quando a Thread é criada, quando é acionada, ela vai para o estado executável, ela não precisa estar necessariamente sendo executada pois o SO que indica o tempo de execução. O estado bloqueado ocorre quando a Thread é desativada, geralmente pelos métodos sleep, wait e no fim da execução ela vai para o encerrado.

A utilização das Threads devem ao fato de múltiplas aplicações estarem sendo executadas muitas atividades juntas, elas são mais fáceis de manipular pois os processos tem recursos próprios e elas não. Alguns benefícios incluem: poder de muitas requisições em páginas WEB possuindo um tempo de resposta bom, compartilhamento de recursos como mesmo endereçamento, memória, realização de multithreads, processamento paralelo pois a CPU alterna entre as threads na execução. Sendo assim, as threads são recursos que permitem um mesmo programa ter várias linhas de execução podendo ser concorrentes ou paralelas.

Para realizar a troca de contexto de thread cada thread possui seu contexto hardware com o conteúdo dos registradores gerais, PC e SP. Threads são implementadas internamente na memória principal através de uma estrutura de dados chamada bloco de controle de thread.

Pelo fato das threads de um mesmo processo compartilhar o mesmo espaço de endereçamento é necessário garantir que uma thread não altere dados de outra thread e por isso deve se utilizar das técnicas de sincronização e comunicação entre threads.

Agora, sobre o projeto simulado do meu tema, que é funcionamento de threads, realizei 2 simuladores, sendo o primeiro mais interativo e demonstrativo e outro mais técnico utilizando um comparativo que mostra a concorrência pelo uso do mesmo recurso variável de uma thread.

## 2. Projeto Simulado (ZIP)

Segue abaixo o Link da pasta dos arquivos que fiz sobre threads usados nesta atividade, sendo um arquivo.c contendo o primeiro simulador feito em linguagem C e outro arquivo .java do segundo simulador.

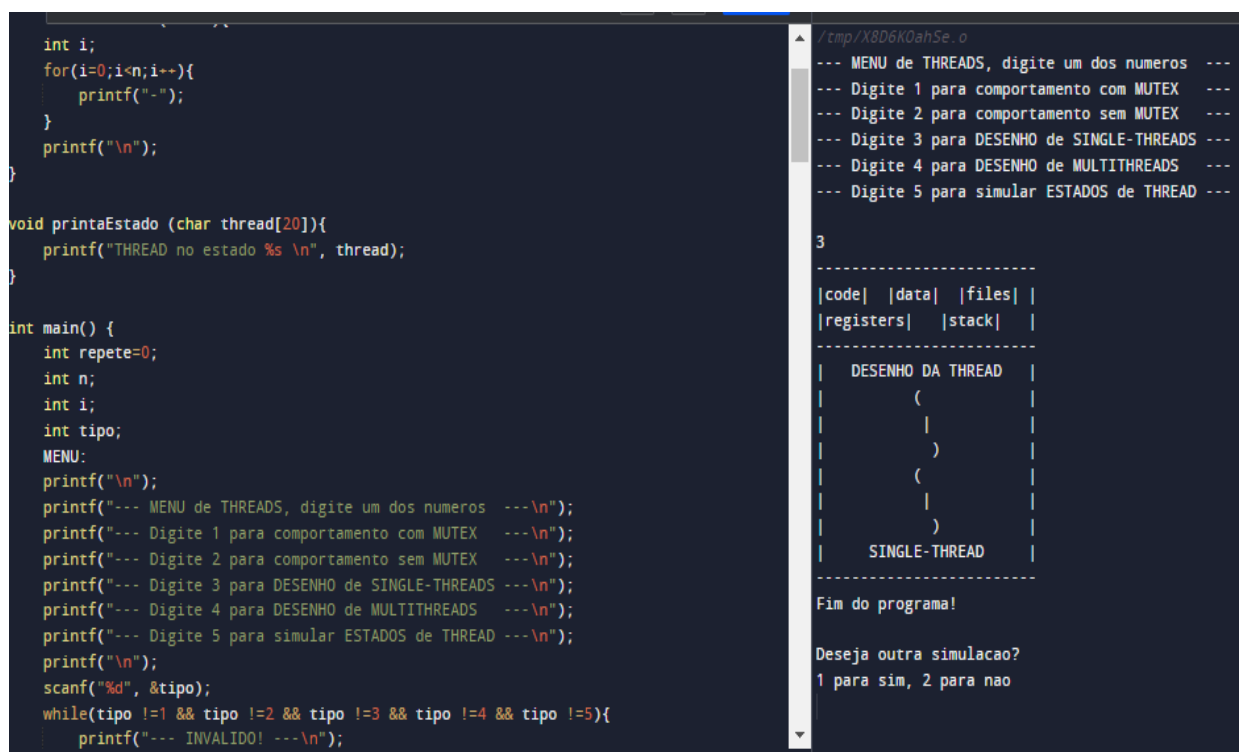
Link:

<https://drive.google.com/drive/folders/1kOS7qoeQnNCcUG3YEzmWbjagMDu7AcNc?usp=sharing>

## 3. Como utilizar o projeto

Para o uso do primeiro simulador precisa de um compilador de código C.

No primeiro simulador, realizado na linguagem C, a tela de execução é a seguinte:



```
int i;
for(i=0;i<n;i++){
    printf("-");
}
printf("\n");

void printaEstado (char thread[20]){
    printf("THREAD no estado %s \n", thread);
}

int main() {
    int repete=0;
    int n;
    int i;
    int tipo;
    MENU:
    printf("\n");
    printf("--- MENU de THREADS, digite um dos numeros ---\n");
    printf("--- Digite 1 para comportamento com MUTEX ---\n");
    printf("--- Digite 2 para comportamento sem MUTEX ---\n");
    printf("--- Digite 3 para DESENHO de SINGLE-THREADS ---\n");
    printf("--- Digite 4 para DESENHO de MULTITHREADS ---\n");
    printf("--- Digite 5 para simular ESTADOS de THREAD ---\n");
    printf("\n");
    scanf("%d", &tipo);
    while(tipo !=1 && tipo !=2 && tipo !=3 && tipo !=4 && tipo !=5){
        printf("--- INVALIDO! ---\n");
    }
```

```
--- MENU de THREADS, digite um dos numeros ---
--- Digite 1 para comportamento com MUTEX ---
--- Digite 2 para comportamento sem MUTEX ---
--- Digite 3 para DESENHO de SINGLE-THREADS ---
--- Digite 4 para DESENHO de MULTITHREADS ---
--- Digite 5 para simular ESTADOS de THREAD ---

3

-----
|code| |data| |files| |
|registers| |stack| |
-----
|  DESENHO DA THREAD  |
|  (                   |
|  |                   |
|  )                   |
|  (                   |
|  |                   |
|  )                   |
|  SINGLE-THREAD      |
|-----|
Fim do programa!

Deseja outra simulacao?
1 para sim, 2 para nao
```

Figura 1: Simulador em C, menu threads.

Neste simulador em C, é exibido um MENU de Threads contendo 5 opções. Na primeira, passa a quantidade de threads a serem criadas que são alocadas em um vetor. Digitando 1 é demonstrado um modelo de comportamento como se houvesse o MUTEX, ou seja as threads acessam seus recursos mas sem invadirem a posição do outro, não concorrendo e não invadindo a posição das outras, sendo exibido de forma correta 1 ao 5 como na imagem abaixo.

```

--- MENU de THREADS, digite um dos numeros ---
--- Digite 1 para comportamento com MUTEX ---
--- Digite 2 para comportamento sem MUTEX ---
--- Digite 3 para DESENHO de SINGLE-THREADS ---
--- Digite 4 para DESENHO de MULTITHREADS ---
--- Digite 5 para simular ESTADOS de THREAD ---

1
--- Digite quantas threads quer criar: 5
Threads Sem Concorrência
Thread acessa a variável 1
Thread acessa a variável 2
Thread acessa a variável 3
Thread acessa a variável 4
Thread acessa a variável 5
Fim do programa!

Deseja outra simulacao?
1 para sim, 2 para nao

```

Figura 2: Simulador Threads em C, opção 1.

Para o segundo caso, digitando o 2, é um visual do modelo sem MUTEX ou sem estratégias para manipulação de threads, o que ocorre compartilhamento de recursos e de mesma variável, onde uma pode invadir a posição de outra como na imagem abaixo:

```

--- MENU de THREADS, digite um dos numeros ---
--- Digite 1 para comportamento com MUTEX ---
--- Digite 2 para comportamento sem MUTEX ---
--- Digite 3 para DESENHO de SINGLE-THREADS ---
--- Digite 4 para DESENHO de MULTITHREADS ---
--- Digite 5 para simular ESTADOS de THREAD ---

2
--- Digite quantas threads quer criar: 5
Threads Concorrendo
Thread acessa a variável 3
Thread acessa a variável 2
Thread acessa a variável 3
Thread acessa a variável 2
Thread acessa a variável 4
Fim do programa!

Deseja outra simulacao?
1 para sim, 2 para nao

```

Figura 3: Simulador Threads em C, opção 2.

Nesta segunda imagem, vemos que a thread acessa a variável 2 e outra thread criada também acessa o mesmo conteúdo 2 da variável, o mesmo ocorre acessando a variável 3, já as demais não houveram concorrências. Este modelo não apresenta o MUTEX.

Mutex é um dos mecanismos de proteções de dados mais amplamente usados em linguagem C e C++, sendo importante estruturar o código para proteger os dados corretos e evitar race conditions inerentes às execuções da interface do simulador. Mutex tem alguns próprios problemas como na forma de um deadlock e proteção demais de muitos ou poucos dados.

Na terceira opção do menu, digitando o número 3, é exibido uma demonstração de um Single-Thread.

3

DESENHO DA THREAD

```
(
|
)
(
|
)
SINGLE-THREAD
```

1  
2  
3  
4  
5  
6  
7  
8



code	data	files
registers	registers	registers
stack	stack	stack
(	(	(
)	)	)
(	(	(
)	)	)
MULTITHREADS		

1  
2  
3  
4  
5  
6  
7  
8

5

```
THREADS
Digite 1 para criar a thread: 1

THREAD no estado nova

Deseja alterar o estado? 1 para sim, 2 para nao
1

THREAD no estado executando
Deseja alterar o estado? 1 para sim, 2 para nao
1|
Deseja bloquear ou finalizar?
1 para bloquear
2 para finalizar
2

THREAD no estado morta
Fim do programa!
```

O segundo simulador foi feito em Java, trata-se de compartilhamento de uma variável por 2 threads, sendo mais uma animação de demonstração. É usado os conceitos de dormir (sleep) por um tempo de uma e comparado as mudanças na variável para ver qual thread é executada e qual colocada pra dormir. Neste caso do exemplo executado no simulador, duas threads competem entre si para mudar o valor de uma variável, uma delas incrementa o valor da variável e a outra decrementa o valor. No contexto geral o valor será decrementado mais que incrementado quando a segunda thread roda mais rápido que a primeira pois seus intervalos de sono são mais curtos e se alterar o valor de sono para maiores, o valor de incremento fica maior do que decremento. Segue abaixo as execuções.

```
while(true) {
    variavelCompartilhada++;
    System.out.println("Variável vale: " + variavelCompartilhada);
    dormir(1500);
}

public void dormir(int milissegundos) {
    try {
        System.out.println(Thread.currentThread().getName() + " irá dormir por " + milissegundos);
        Thread.sleep(milissegundos);
    } catch (InterruptedException e) {
    }
}

ThreadDecrementa extends Thread {

    private Threads t1;

    public ThreadDecrementa(Threads t1) {
        this.t1 = t1;
    }

    @Override
    public void run() {
        while(true) {
            t1.variavelCompartilhada--;
            System.out.println("Variável vale: " + t1.variavelCompartilhada);
            t1.dormir(1000);
        }
    }
}
```

Figura 7: Simulador2 em JAVA, threads, tempo de sleep.

```

Thread-0 irá dormir por 1000 milissegundos.
Variável vale: -1
main irá dormir por 1500 milissegundos.
Variável vale: -2
Thread-0 irá dormir por 1000 milissegundos.
Variável vale: -1
main irá dormir por 1500 milissegundos.
Variável vale: -2
Thread-0 irá dormir por 1000 milissegundos.
Variável vale: -3
Thread-0 irá dormir por 1000 milissegundos.
Variável vale: -2
main irá dormir por 1500 milissegundos.
Variável vale: -3
Thread-0 irá dormir por 1000 milissegundos.
Variável vale: -2
main irá dormir por 1500 milissegundos.
Variável vale: -3
Thread-0 irá dormir por 1000 milissegundos.
Variável vale: -4
Thread-0 irá dormir por 1000 milissegundos.
Variável vale: -3
main irá dormir por 1500 milissegundos.
Variável vale: -4
Thread-0 irá dormir por 1000 milissegundos.
Variável vale: -3
main irá dormir por 1500 milissegundos.
Variável vale: -4
Thread-0 irá dormir por 1000 milissegundos.
Variável vale: -5
Thread-0 irá dormir por 1000 milissegundos.
Variável vale: -4
main irá dormir por 1500 milissegundos.
Variável vale: -5
Thread-0 irá dormir por 1000 milissegundos.

```

Figura 8: Simulador2 em JAVA, threads, resultado da variavel compartilhada.

Neste caso, a variável compartilhada vale -5, foi mais decrementada do que incrementada, isso ocorre porque a segunda thread roda mais rápido que a primeira por conta do intervalo de sono mais curto (sleep).

Alterando este intervalo de tempo da thread de 1500 para um menor que 1000, por exemplo 500, obtemos os seguintes resultados abaixo:

```

public void executar() {
    Thread segundaThread = new ThreadDecrementa(this);
    segundaThread.start();

    while(true) {
        variavelCompartilhada++;
        System.out.println("Variável vale: " + variavelCompartilhada);
        dormir(500);
    }
}

public void dormir(int milissegundos) {
    try {
        System.out.println(Thread.currentThread().getName() + " irá dormir por " + milissegundos + " milissegundos.");
        Thread.sleep(milissegundos);
    } catch (InterruptedException e) {}
}

ss ThreadDecrementa extends Thread {
    private Threads t1;

    public ThreadDecrementa(Threads t1) {
        this.t1 = t1;
    }

    @Override
    public void run() {
        while(true) {
            t1.variavelCompartilhada--;
            System.out.println("Variável vale: " + t1.variavelCompartilhada);
            t1.dormir(1000);
        }
    }
}

```

Figura 9: Simulador2 em JAVA, alterando o valor de dormir para 500.

```

main irá dormir por 500 milissegundos.
Variável vale: 10
main irá dormir por 500 milissegundos.
Variável vale: 9
Thread-0 irá dormir por 1000 milissegundos.
Variável vale: 10
main irá dormir por 500 milissegundos.
Variável vale: 11
main irá dormir por 500 milissegundos.
Variável vale: 10
Thread-0 irá dormir por 1000 milissegundos.
Variável vale: 11
main irá dormir por 500 milissegundos.
Variável vale: 12
main irá dormir por 500 milissegundos.
Variável vale: 11
Thread-0 irá dormir por 1000 milissegundos.
Variável vale: 12
main irá dormir por 500 milissegundos.
Variável vale: 13
main irá dormir por 500 milissegundos.
Variável vale: 12
Thread-0 irá dormir por 1000 milissegundos.
Variável vale: 13
main irá dormir por 500 milissegundos.
Variável vale: 14
main irá dormir por 500 milissegundos.
Variável vale: 13
Thread-0 irá dormir por 1000 milissegundos.
Variável vale: 14
main irá dormir por 500 milissegundos.

```

Figura 10: Simulador2 em JAVA, variavel compartilhada valendo 14.

Já nessa ocasião, como ocorre o inverso, agora a variável compartilhada entre as threads passa valer 14 positivo, mais incrementada do que decrementada, pois a outra thread está com intervalo de sono maior, 1000 milissegundos, do que a de 500 que reduzimos.

Com isso, encerra o segundo simulador feito para esta atividade.

Por fim, threads tem vantagens quando divide um programa em vários processos, uma delas é que auxilia o desenvolvimento pois cria o programa em módulos, testando-os separadamente no lugar de escrever em um único bloco de código. Uma outra vantagem dos threads é que eles não deixam o processo parado, enquanto um deles está aguardando um determinado dispositivo de I/O (entrada ou saída), ou algum outro recurso do sistema, outro thread pode estar trabalhando.

#### 4. Vídeo de como utilizar o projeto/execução.

Link do vídeo: <https://youtu.be/uW0jVZCZSuQ> e <https://youtu.be/andqfbahrVY>  
O primeiro link é o simulador em C o segundo um exemplo adicional em Java.

#### Referências:

TANENBAUM, A. S. Sistemas Operacionais Modernos. 3. ed. [S.l.]: Pearson, 2010.

Higor. Programação com Threads. DevMedia, 2007. Disponível em:  
< <https://www.devmedia.com.br/programacao-com-threads/6152/>>. Acesso em: 16 de ago. de 2021.

PANTUZA, Gustavo. O que são e como funcionam as Threads. Blog Pantuza, 2017. Disponível em:  
< <https://blog.pantuza.com/artigos/o-que-sao-e-como-funcionam-as-threads/>>. Acesso em: 16 de ago. de 2021.

CANALTECH. O que é thread. CANALTECH, 2021. Disponível em:  
< <https://canaltech.com.br/produtos/o-que-e-thread/>>. Acesso em: 16 de ago. de 2021.