Universidade Federal de São Paulo - Instituto de Ciência e Tecnologia

ATIVIDADE 3 - SEÇÃO CRÍTICA POR ESPERA OCUPADA

Alunos: João Victor de Mesquita Cândido dos Santos, RA:102028 , Raphael Ribeiro Faria, RA: 104120

Unidade Curricular: Programação Concorrente e Distribuída

Docente: Prof. Dr. Álvaro Luiz Fazenda

Problema: Implemente, usando linguagem C com *PThreads* ou *OpenMP* ou ainda em *JavaThreads*, o algoritmo de Manna-Pnueli que implementa entrada em SC por algoritmo Cliente-Servidor. Demonstre o funcionamento do código para 2 e 4 *threads* como processos clientes, realizando o correto incremento em uma determinada variável global ou ainda através de "prints" que demonstrem o funcionamento correto da Exclusão Mútua para a seção crítica.

Solução do Problema

- **Solução com exclusão mútua.**

A resolução do problema foi bem simples, onde temos duas variáveis globais chamadas *request* e *respond* que controla as requisições do cliente para o servidor.

Primeiramente ambas as variáveis de respond e request se iniciam com um valor 0, na função do cliente chamada (Consumidor_function) há um *while* que verifica se o valor da variável *respond* é diferente do id da thread em questão, no caso é diferente, então a variável request recebe o valor de id da thread como mostra a Figura 1 abaixo.

```
void *Consumidor_function(void){

    int i, id;
    for(i=0;i<N;i++){
        id = pthread_self();
        while(respond!=id){
            request = id;
        }
```

Figura 1 - Pré-protocolo da função consumidor.

Com isso, de maneira concorrente a função (Servidor_function) verifica que a variável request foi alterada, dessa maneira a variável respond recebe o valor que está armazenado em request que é o valor de id da thread como mostra a Figura 2 abaixo.

```
void *Servidor_function(void){

    while(1){
        if(request != 0){
            respond = request;
```

Figura 2 - Pré-protocolo da função servidor.

Após isso retornando na função do consumidor, o valor da variável respond é igual ao id da thread, assim sendo, a thread entra na região crítica onde nessa parte há prints identificando a entrada e saída da mesma na região como mostra a Figura 3 abaixo.

```
if(respond==id){
printf("Thread %d entrando na regiao critica!\n", pthread_self()-1);
Sleep(500);
printf("Thread %d saindo da regiao critica!\n\n", pthread_self()-1);
Sleep(500);
```

Figura 3 - Região Crítica.

Após isso a variável respond recebe zero na função do consumidor e a variável request recebe zero na função do servidor para que se reinicie o processo só que com outra thread. Assim sendo com essa solução a região crítica é acessada por uma thread por vez.

- **Solução sem exclusão mútua**

A solução sem exclusão mútua é bem simples, nesse caso não há o controle de uma thread por vez entrar na região crítica da função do consumidor, assim sendo a única mudança no código ocorre na função do servidor, onde sempre a variável responde recebe o valor de request que é o valor de id da thread, ou seja, todas as threads podem acessar a região crítica de uma vez só. A mudança na função pode ser vista na Figura 4 abaixo.

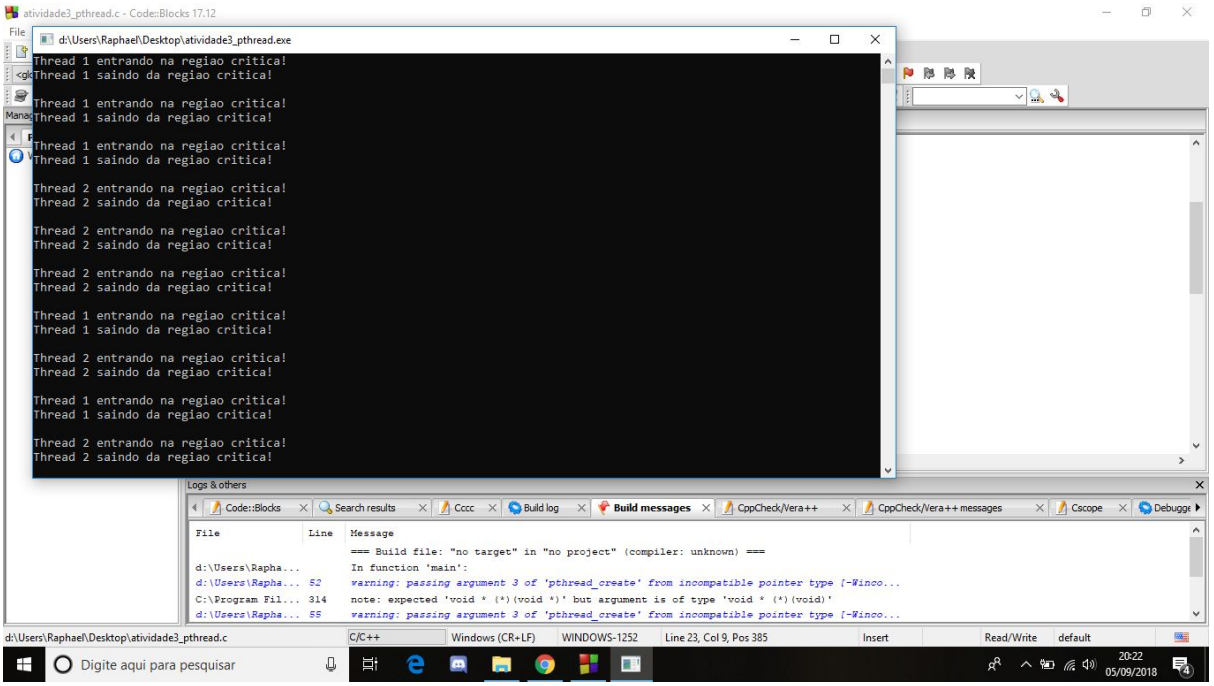
```
void *Servidor_function(void){

    while(1){
            respond = request;
        }
    }
```

Figura 4 - Solução sem exclusão mútua.

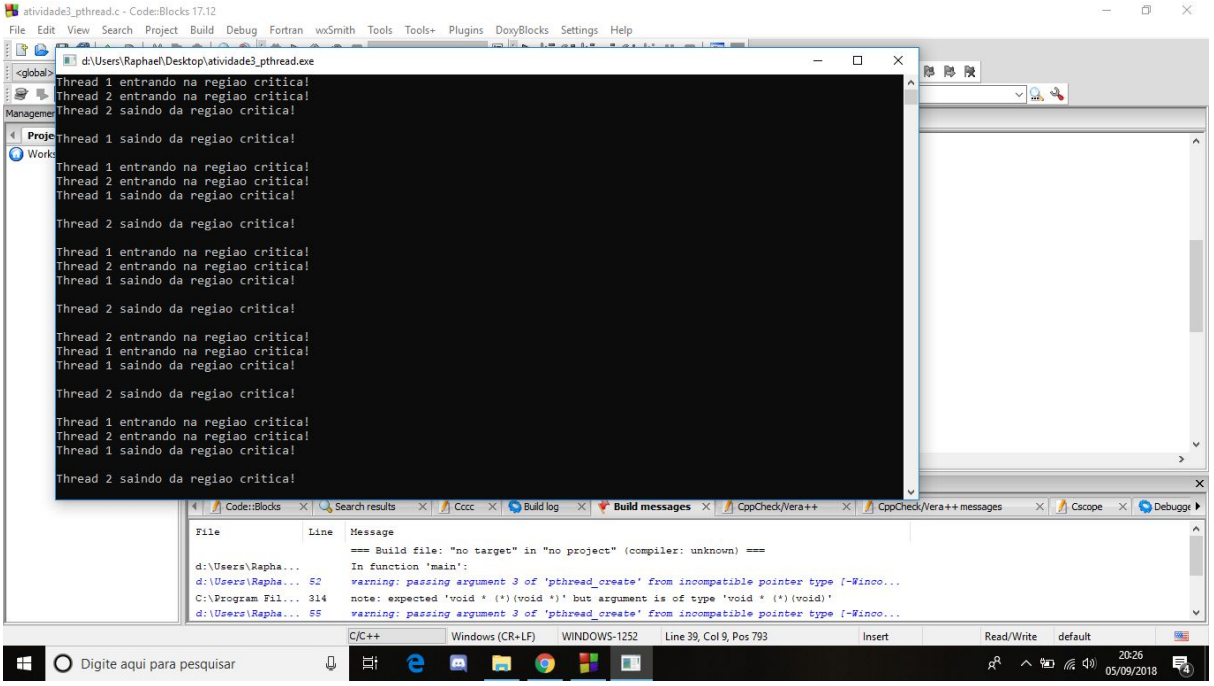
Resultados Obtidos

Como proposto, através da resolução do problema em questão foram feitos os testes e consequentemente tirados prints para comprovar a veracidade das soluções, tais resultados podem serem vistos abaixo.

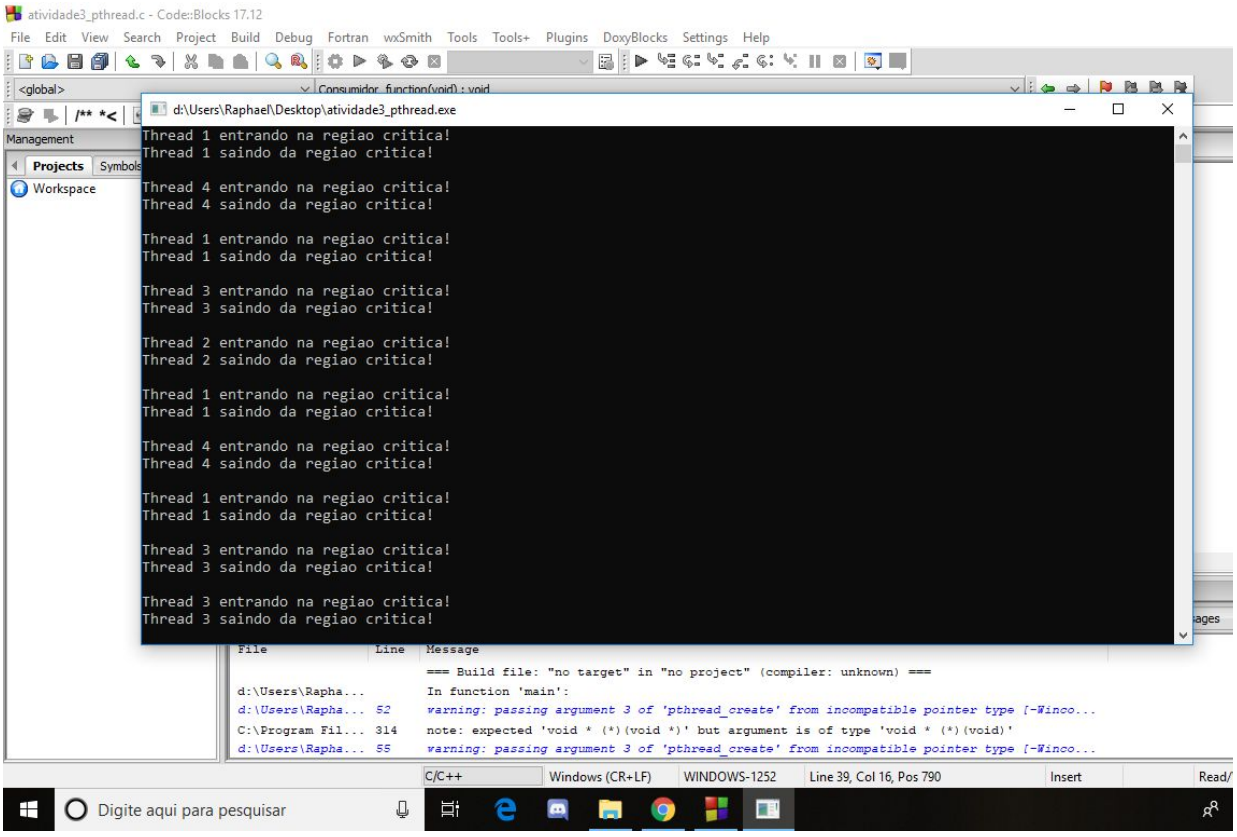
- **2 Threads:**
  - **Com exclusão mútua:**



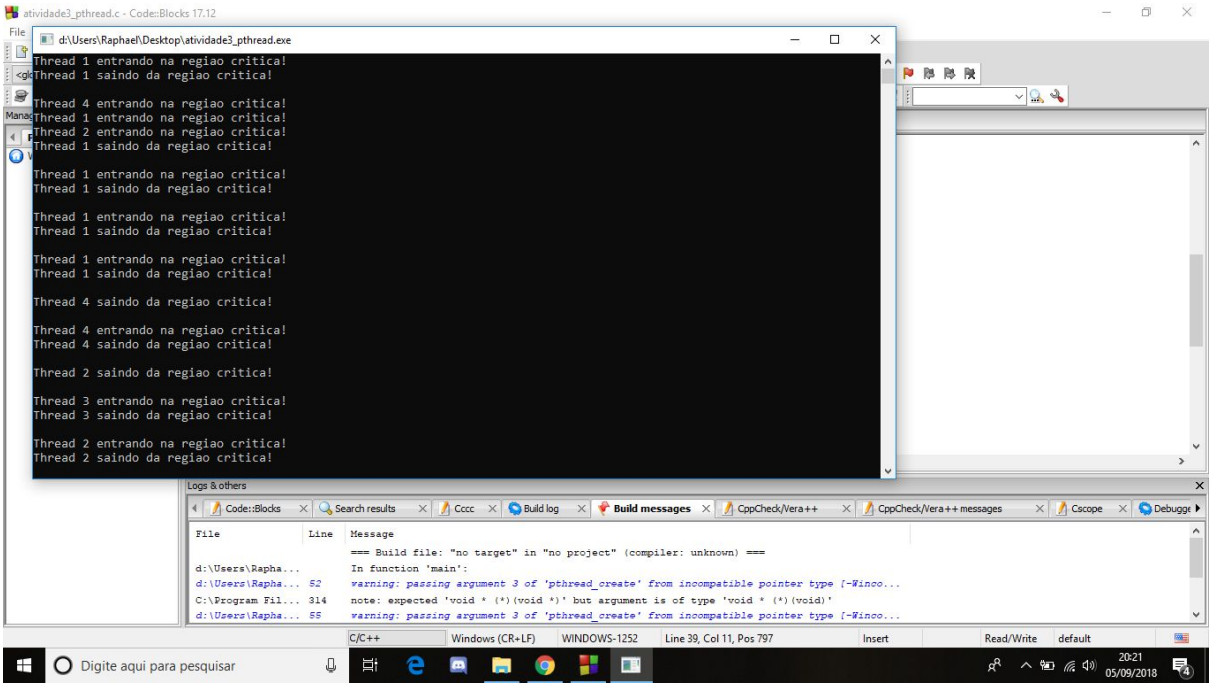
- **Sem exclusão mútua**



- **4 Threads:**
  - **Com exclusão mútua**



- **Sem exclusão mútua**



Descrição do Problema

Para a realização dos testes feitos, foi usado o computador Dell Inspiron com processador Intel Core i5-5200U, que possui dois núcleos físicos com Hyper-Threading de velocidade 2,2 - 2,7 GHz (2 Núcleos: 2,5 GHz), além de 1 TB e 8GB de memória principal e memória RAM, respectivamente. Os algoritmos foram executados utilizando o compilador da IDE Code Blocks MinGW GNU Compiler de versão 32 bits, sendo o sistema operacional da máquina o Windows 10 na versão de 64 bits.