EXCLUSÃO MÚTUA

Matheus Vieira de Araújo, Felipe Augusto Arnaud Alves

Universidade estadual do Ceará (UECE) Campus do itaperi - Av. Dr. Silas Munguba, 1700 - Itaperi, Fortaleza - CE, 60741-000

matheusvieira.11@hotmail.com, felipe.arnaud@aluno.uece.br

Abstract. This meta-paper describes a theater ticket system, presenting a mutual exclusion problem when more than 1 user try to buy the same seat at the same time, implemented in C language using the Dekker, Peterson and Lamport Algorithms.

Resumo. Este meta-artigo descreve um sistema de compras de ingressos para uma sala de cinema, apresentando um problema de exclusão mútua quando mais de um usuário tenta comprar um acento ao mesmo tempo, implementadas soluções na linguagem C para o mesmo usando os Algoritmos de Dekker, Peterson e Lamport.

**Bibliotecas**

No desenvolvimento deste algoritmo foram necessárias as bibliotecas: “<stdio.h>” para funções de rotina como entrada e saída de dados do teclado e exibir informações na tela do programador; “<stdlib.h>” que possui funções envolvendo alocação de memória, controle de processos; <pthread.h> para criação de threads; “<unistd.h>” para que o código tenha facilidade de rodar em mais sistemas operacionais. Utilizamos duas bibliotecas adicionais para fazer a execução no Dos do windows, são elas <dos.h> <conio.h>.

**Algorítimo**

A grande dificuldade que tivemos para implementar o mutex no nosso programa foi de entender como funciona cada algoritmo, suas diferenças e requisitos para poder saber utilizar e adaptar no código do sistema de Reserva de Cadeiras. Por exemplo na situação onde precisávamos escolher a forma de trabalhar com o vetor que representa as cadeiras da sala, se poderia ser compartilhado e qual melhor forma, como faria sua execução, em uma função separada ou apenas declarando o vetor e cada thread executando a atualização.

Cada algoritmo de mutex tivemos que mudar a forma da thread armazenar no vetor a cadeira escolhida pelo usuário

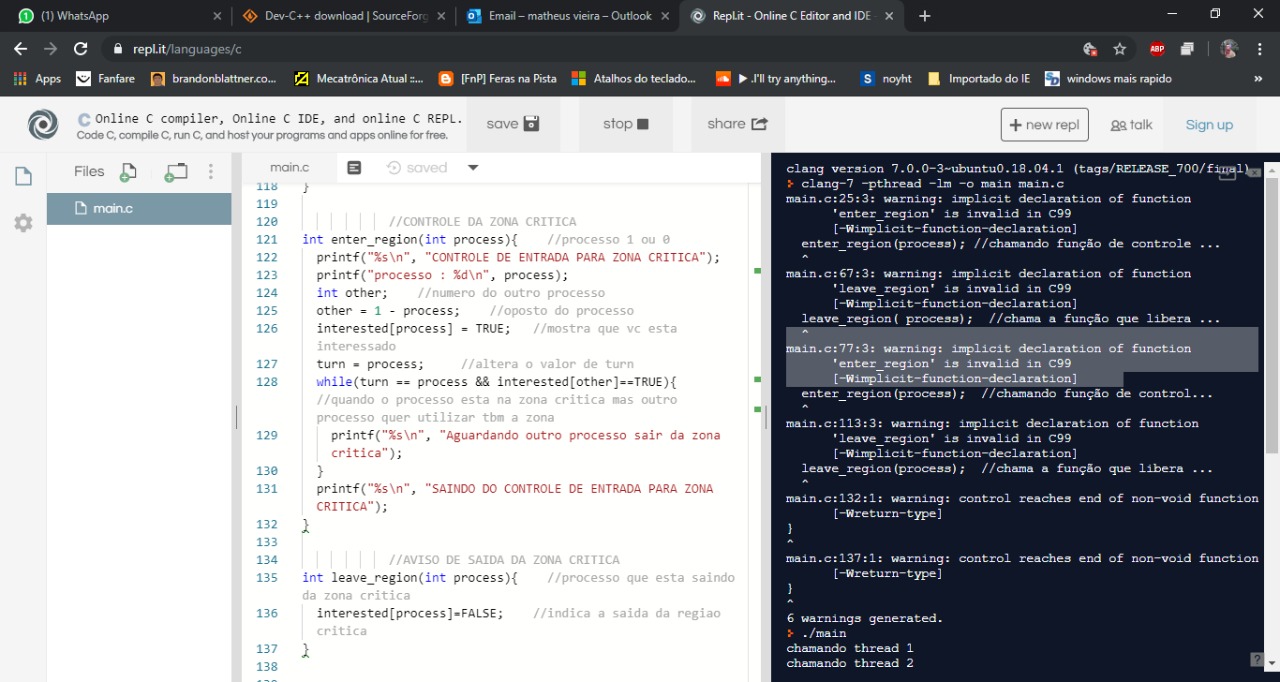
Decker:

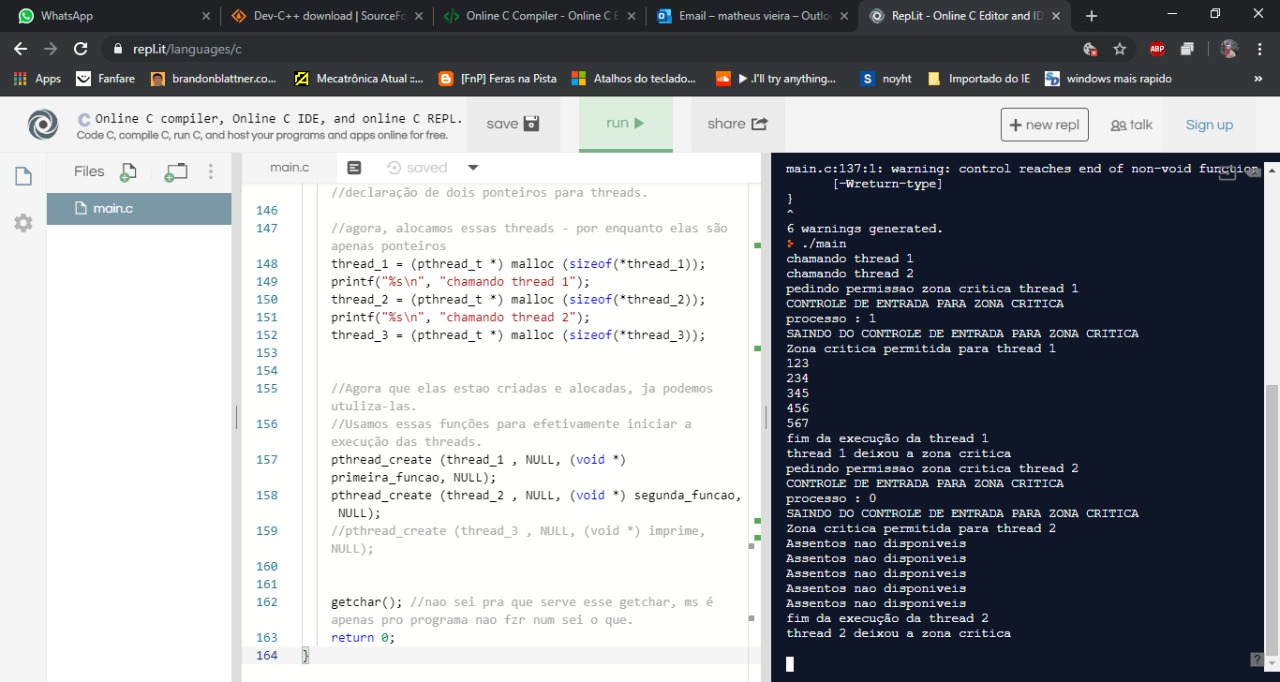
No algoritmo de decker, a gente utilizou um vetor compartilhado para ser acessado pela thread e ser escrito enquanto estiver na zona crítica, Não tivemos muita dificuldade no sistema compartilhado, o mutex tbm bem simples cada thread pedindo um lock dentro do mutex, e utilizamos um while para a thread ficar pedindo acesso, não foi de grande dificuldade.

Peterson

No algoritmo de Peterson tivemos como base o código fornecido pelo livro, onde existe uma função para realizar o controle de acesso na zona crítica, utilizando variável para indicar o interesse da thread e conferir se outra thread já está fazendo uso. tivemos dificuldade com a sintaxe do programa, em conseguir fazer referências de função para criar threads passando seu valor por parâmetro, e vários erros para chamar a função que executava o controle das threads (Enter\_Region), porém já funcionava perfeitamente mesmo apresentando aviso de declaração incorreta.

Primeiro o algoritmo apresentando falha na referencia mas funcionando.



Segundo o algoritmo funcionando normalmente:

Lamport

A técnica do mutex de lamport foi a que apresentou maiores problemas, pois não havia tanto conteúdo na internet para termos como orientação. Utilizamos duas bibliotecas adicionais para fazer a execução no Dos do windows, são elas <dos.h> <conio.h>. tivemos que usar um gerador de ID aleatório para os usuários, e fizemos o vetor dos assentos atualizar todas as vezes que o usuário fizesse um registro, assim ficou mais realista e automática a escolha dos locais e por quem escolheu.

