Escola de Artes, Ciências e Humanidades (EACH)

Prática 02 - ACH2044 Sistemas Operacionais
Relatório do experimento

Nome: Guilherme Cavalanti Gomes
NºUSP: 11844788

O PROBLEMA

O objetivo desse EP, é implementar um programa que possua dois processos que compartilhem de uma mesma variável global, utilizando POSIX Threads em C. Dessa forma, será usada uma seção crítica a partir de um algoritmo de espera ociosa baseado nos pseudocódigos passado:

Algoritmo 1

P0

```
meu_id = 0;
outro = 1;
while(TRUE){

        while(vez != meu_id)
                /*laço*/ ;

        secao_crítica();

        vez = outro;

        secao_nao_critica;
}
```

P1

```
meu_id = 1;
outro = 0;
while(TRUE){

        while(vez != meu_id)
                /*laço*/ ;

        secao_crítica();

        vez = outro;

        secao_nao_critica;
}
```

O objetivo é implementar esse mesmo algoritmo em C usando POSIX Threads, e analisar a sua funcionalidade. Após isso, será verificado se ele segue as 3 condições para que haja exclusão mútua.

IMPLEMENTANDO O ALGORITMO:

A lógica usada nessa solução é a da espera ocupada, ou seja, as threads irão consultar a seção crítica constantemente buscando acessá-la, e, se ela já estiver ocupada, elas aguardam até ela ficar livre.

Para isso seria utilizada uma variável para controlar o acesso de cada um dos processos, que alterna entre as duas threads no acesso à região crítica, criando um sistema de turnos. Assim, o mesmo processo não pode utilizar a sessão crítica

duas vezes seguidas. Além de uma variável global inteira ‘count’, que será modificada pelos 2 processos.

O CÓDIGO EM C:

```c
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>

//Variáveis globais do programa
int count;
int vez;

//Função executada dentro da seção crítica
void secao_critica (int threadid){
    int id = (int) threadid;
    printf("(%d) Secao Critica, count = %i\n",id, count);
    count += 1;
    //pthread_exit(NULL);
}

//Função de espera
void secao_nao_critica (int threadid) {
    int id = (int) threadid;
    printf("(%d) Esperando\n",id);
    //pthread_exit(NULL);
}

void *p0 (void *threadid){
    int id = (int) threadid;
    int outro = 1;
    while (1) {
        while(vez != 0) ;
        secao_critica(id);
        vez = outro;
        secao_nao_critica(id);
    }
}
void *p1 (void *threadid){
    int id = (int) threadid;
    int outro = 0;
    while (1) {
        while(vez != 1) ;
        secao_critica(id);
```

```
        vez = outro;
        secao_nao_critica(id);
    }
}

void main(){
    vez = 0;
    count = 0;
    int num; //indetificador da thread
    pthread_t t1, t2;
    int numThread; // armazena o retorno da pthread_create
    num = 0;
    numThread = pthread_create(&t1, NULL, p0, (void *)num);
    num = 1;
    numThread = pthread_create(&t2, NULL, p1, (void *)num);
    pthread_join(t1,NULL); //Espera a t1 acabar
    pthread_join(t2,NULL); // Espera a t2 acabar
}
```

Resultados:

```
C:\Users\W10\Desktop\EP2-OAC>programa2.exe
(0) Secao Critica, count = 0
(0) Esperando
(1) Secao Critica, count = 1
(1) Esperando
(0) Secao Critica, count = 2
(0) Esperando
(1) Secao Critica, count = 3
(1) Esperando
(0) Secao Critica, count = 4
(0) Esperando
(1) Secao Critica, count = 5
(1) Esperando
(0) Secao Critica, count = 6
(0) Esperando
(1) Secao Critica, count = 7
(1) Esperando
(0) Secao Critica, count = 8
(0) Esperando
(1) Secao Critica, count = 9
(1) Esperando
(0) Secao Critica, count = 10
(0) Esperando
```

Análise:

Como é possível observar, a solução é funcional, já que os 2 processos trabalham concorrentemente sem interferências e sem distorcer o valor da variável global ‘count’. Dito isso, é preciso analisar se ele atende as 3 condições necessárias para que haja exclusão mútua:

1- Exclusão Mútua: Atende, pois quando uma das threads está na seção crítica, a outra necessariamente não pode estar devido ao valor da variável ‘vez’, ela não pode ser igual a 1 e igual a 0 ao mesmo tempo.

2-Progesso: Ela é atendida pois, dentro das funções ‘p1’ e ‘p0’ que são executadas nas threads, há um loop infinito então elas nunca ficarão ociosas ao mesmo tempo, visto que o valor de ‘vez’ tem que favorecer uma das duas.

3-Espera limitada: Também é cumprida, pois, sempre ao fim da tarefa na seção crítica, o turno muda para o próximo processo, ou seja, sempre chegará a vez daquele processo não há como ele ficar na fila para sempre.

Logo essa solução é válida e atinge aos 3 pontos da exclusão mútua, um único ponto fraco seria que a ordem dos processos é definida por turnos, então se há 1 processo muito rápido e 1 muito lento, o rápido vai ter que esperar o lento terminar toda vez que seu turno acabar, gerando uma perda no desempenho se comparado à outras soluções.