Relatório EP 2 · SO

Prof. Dra. Gisele da Silva Craveiro

Mikael Gi Sung Shin, 10843441

Código (com comentários):

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

#define count 100000000 // variável global count
#define tam 2 // número de processos/threads

int vez = 0; // declarando e definindo de quem é a vez da execução

void secao_critica(int thread){ // função que simula a execução da região crítica

    int i;
    for (int i = 0; i < count; i++) { // uma execução qualquer que demande um tempinho
        i = i;
    }
    printf("P%d: Entrou na Seção Crítica\n", thread); // mensagem de entrada na seção crítica
}

void secao_nao_critica(int thread){ // função que simula a execução da seção não crítica

    int i;
    int num = 10000000;
    for (int i = 0; i < num; i++) { // uma execução qualquer que demande um tempinho
        num = num;
    }
    printf("P%d: Saiu da Seção Crítica\n", thread); // mensagem de entrada na seção crítica
}
```

```c
void *run (void *arg){ // função que irá servir de execução para cada uma dos processo

    int* n;
    n = arg; // necessário para os seguintes if's
    int meu_id;
    int outro;

    if(*n == 0){ // se o argumento do processo for 0, ele entra para a execução

        meu_id = 0; // declarando o ID do processo 0
        outro = 1; // armazenando o ID do outro processo

        printf("P0 chegou no método 'run'\n"); // mensagem de sucesso

        while(1){ // enquanto for verdade

            while(vez != meu_id); // enquanto vez é diferente do id, executa (no caso, nada)
            secao_critica(meu_id); // processo entra na seção crítica
            vez = outro; // dá a vez para o outro processo
            secao_nao_critica(meu_id); // processo executa algo que está fora da seção crítica
        }
    }

    else if(*n == 1){ // bloco semelhante ao anterior, só que para a processo 1

        meu_id = 1;
        outro = 0;

        printf("P1 chegou no método 'run'\n");

        while(1){

            while(vez != meu_id);
            secao_critica(meu_id);
            vez = outro;
            secao_nao_critica(meu_id);
        }
    }
}
```

```c
int main()
{

    pthread_t p[tam]; // declarando as threads posix
    int i, arg[tam]; // declarando as variáveis inteiras

    for(i = 0; i < tam; i++){ // for: 0 e 1

        arg[i] = i; // array de argumento, posição 0 armezena 0, posição 1 armazena 1
        pthread_create(&p[i], NULL, run, (void*)&arg[i]); // instanciação das threads 0 e 1
    }

    for(i = 0; i < tam; i++){

        pthread_join(p[i], NULL); // aguarda o término de cada uma das threads
    }

    pthread_exit(NULL); // finaliza a execução das threads

    return 0;
}
```

Saída:

```
T1 chegou no método 'run'
T0 chegou no método 'run'
P0: Entrou na Seção Crítica
P0: Saiu da Seção Crítica
P1: Entrou na Seção Crítica
P1: Saiu da Seção Crítica
P0: Entrou na Seção Crítica
P0: Saiu da Seção Crítica
P1: Entrou na Seção Crítica
P1: Saiu da Seção Crítica
P0: Entrou na Seção Crítica
P0: Saiu da Seção Crítica
```

Explicação do código:

O método main() faz as instanciações das threads posix necessárias para o resolução do exercício programa em questão.

Foi criado um método run() a fim de executar as threads que foram criadas. A thread que recebe o argumento 0, tem o ID igual 0. Da mesma forma, a thread que recebe o argumento 1, tem o ID igual 1. Cada um desses cenários, é armazenado na variável “outro”, o ID da outra thread. Além disso, é executada uma mensagem de sucesso pela entrada dos threads neste método. Em seguida, apenas o thread cujo o id é igual a variável “vez”, declarada nas primeiras linhas de comando, entra no “while” a fim de simular a execução condicionada da seção crítica (um processo por vez). Depois do método secao_critica() ser finalizado, é passada a vez para o outro processo que, agora sim, poderá entrar neste método e ser executado.

Como podemos ver na saída do programa, primeiro é exibida as mensagens de sucesso citada em linhas anteriores, e em seguida, enxergamos que a execução do processo da seção crítica ocorre uma por vez. Esta implementação respeita as regras de que dois ou mais processos não podem estar simultaneamente dentro dessa região e que o processo não espera infinitamente até um outro conseguir entrar na região crítica. Por outro lado, ele bloqueia a entrada de outros processos nesta região, visto que a variável “vez” comanda a entrada de um em detrimento do bloqueio de outro.

Instruções para compilação:

1) Usar alguma IDE que suporte a linguagem C, e execute (“run”) o código descrito acima. (obs: utilizei o compilador online https://www.onlinegdb.com/online_c_compiler).
2) Executar as seguintes linhas de código em algum terminal, dentro do diretório do programa:
   `gcc “nomedoarquivo”.c -o “nomedoarquivo”`
   `./“nomedoarquivo”`

Referências para o desenvolvimento do código:

1) https://www.youtube.com/watch?v=6wFCoa7x9BY&ab_channel=UNIVESP
2) http://people.sabanciuniv.edu/ysaygin/documents/lectures/CS307_lecture_2.pdf