Programação Concorrente - Trabalho 1

Alice da Silva de Lima - 18/0112601

Universidade de Brasília

1 Introdução

Ao decorrer do semestre, a disciplina de Programação Concorrente abordou os conceitos desse paradigma de programação, bem como a resolução de problemas com comunicação de processos que fazem uso de memória compartilhada [1].

Para tratar desses problemas, foram utilizados vários recursos da biblioteca POSIX *pthread*[1], que disponibiliza mutex, variáveis de condição e suas respectivas funcionalidades. Um outro recurso também utilizado foi o semáforo, da biblioteca POSIX *semaphore*[2].

A fim de consolidar os conhecimentos da disciplina, este trabalho apresenta um problema de programação concorrente e a sua respectiva solução implementada na linguagem C, com utilização dos recursos anteriormente citados.

2 Formulação do Problema Proposto

O problema do presente trabalho consiste em uma situação onde alguns alunos de computação, após o término de suas aulas vão até uma sala de estudos do departamento para estudar. Essa sala possui mesas individuais, porém em uma quantidade limitada. A única prioridade para adentrar a sala é a ordem de chegada, e quando um aluno não consegue uma mesa ele vai para a BCE, biblioteca da universidade. Os alunos que entram, estudam por um tempo determinado e depois vão embora, liberando uma mesa para algum aluno que deseja entrar.

Além disso, dentro da sala existe um frigobar com alguns energéticos. Caso um aluno deseje uma lata, este pode deixar o dinheiro em cima do eletrodoméstico e pegar a bebida. Um mesmo aluno não pode pegar mais de uma lata por vez, e quando resta apenas uma lata disponível o aluno que pegá-la deve chamar um robô que repõe o estoque de energéticos. Este fica aguardando enquanto existem latas disponíveis.

3 Descrição do Algoritmo Desenvolvido

3.1 Descrição geral

Para resolver este problema foi utilizado um semáforo com 7 permissões representando as mesas, duas variáveis de condição (uma para o robô e outra para os

[1] https://pubs.opengroup.org/onlinepubs/7908799/xsh/pthread.h.html
[2] https://pubs.opengroup.org/onlinepubs/7908799/xsh/semaphore.h.html

alunos), e também um *mutex* para garantir exclusão mútua em algumas regiões críticas do programa. Foram implementadas quatro funções: `sala`, onde as *threads* que representam os alunos entram; `estudo`, que é invocada pela primeira função e representa a permanência dos alunos na salinha; `frigobar`, que simula um estudante comprando um energético quando há disponibilidade e a função `f_robo`, onde a *thread* que representa o robô entra e realiza a reposição das latas quando necessário.

3.2 Algoritmo

Constantes

Algumas constantes foram adicionadas ao programa para facilitar a compreensão do código. Das três, uma representa o número de alunos (13), outra o número de mesas (7) e outra o número máximo de latas que cabem no frigobar (4).

```
#define N_ALUNOS 13
#define N_MESAS 7
#define L 4          /*numero de latas*/
```

Variáveis

O semáforo `sem_mesas` representa as mesas da sala. As variáveis de condição `alunos` e `robo` foram adicionadas para realizar o controle da situação onde os alunos pegam um energético no frigobar e a reposição das bebidas pelo robô. Também foi utilizado um `mutex` para exclusão mútua em regiões críticas. A variável global `latas` é um contador auxiliar que decrementa as latas a medida que os alunos vão comprando, e incrementa quando o robô enche o frigobar; como observado, o programa inicia com 4 latas disponíveis.

```
sem_t sem_mesas;
pthread_cond_t alunos = PTHREAD_COND_INITIALIZER;
pthread_cond_t robo = PTHREAD_COND_INITIALIZER;
pthread_mutex_t lock = PTHREAD_MUTEX_INITIALIZER;

int latas = 4;
```

O programa possui 14 *threads*, sendo 13 representando os alunos e uma representando o robô, a inicialização se dá como mostra o código abaixo:

```
pthread_t alunos[N_ALUNOS], robo;

for (i = 0; i < N_ALUNOS; i++) {
    id[i] = i;
    pthread_create(&alunos[i], NULL, sala, (void*) &id[i]);
}

pthread_create(&(robo), NULL, f_robo, (void*) (a));
pthread_join(robo,NULL);

sem_init(&sem_mesas, 0, N_MESAS);
```

O semáforo foi iniciado com 7 permissões, que representam as mesas.

Função sala

A função `sala` possui um `sem_trywait`, que faz o controle de mesas da sala. No condicional mostrado abaixo, um aluno só adentra a sala caso ainda tenha permissão disponível no semáforo, isto é, caso ainda tenha alguma mesa disponível na sala:

```
if(sem_trywait(&sem_mesas) == 0){
    printf("Aluno %d entrou na sala \n",id);
    ...
```

Caso não tenha nenhuma mesa disponível, o aluno vai embora:

```
else{
    printf("Sala cheia, aluno %d vai ter que andar ate a BCE
    ...\n",id);
    sleep(2);
}
```

Função estudo

Quando um aluno consegue uma mesa, este vai estudar na função `estudo`:

```
void estudo(int id){
  int aux = rand() % 10;
  printf("Aluno %d estudando...\n", id);
  if(aux%2 == 0){
    frigobar(id);
  }
  sleep(rand() % 4);
}
```

Neste problema a ideia é que nem todos os alunos que estão dentro da sala comprem um energético. Para isso, foi adicionado um condicional na função, onde toda vez que esta é chamada gera-se um número aleatório *aux*, e então a *thread* do aluno só entra na função frigobar caso este seja par. Além disso, para simular o tempo que cada aluno permanece na sala é dado um *sleep* que tem uma duração variável.

Função frigobar

Na função `frigobar` acontece a compra de um energético pelo aluno. Primeiramente, é verificado se existem latas disponíveis, em caso positivo o aluno compra uma lata, a variável *latas* é decrementada e é verificado se as latas acabaram. Caso as latas tenham acabado, o aluno da vez chama o robô:

```
if(latas == 0){
    printf("%d: Vou chamar o robo\n", id);
    pthread_cond_signal(&robo);
}
```

E assim, quando um novo aluno deseja comprar, é verificado que o frigobar está vazio, e o aluno precisa esperar que o robô reponha as latas:

```
while(latas == 0){
    printf("Aluno %d esperando o robo repor as latas\n", id);
    sleep(1);
    pthread_cond_wait(&alunos,&lock);
}
```

Função f_robo

A lógica da função `f_robo` é semelhante a da função anterior: há um condicional que verifica se as latas acabaram, caso não, a variável de condição do robô aguada até que um aluno a libere. Caso tenham acabado, o robô repõe as latas e acorda os alunos:

```
printf("Robo: vou levar as latas e recolher o dinheiro\n");
sleep(4);
latas = L;
pthread_cond_broadcast(&alunos);
```

Como observado, o robô é acordado com o `pthread_cond_signal`, por se tratar de uma *thread* apenas. Já os alunos são acordados com `pthread_cond_broadcast`, pois mais de um podem estar aguardando para comprar um energético.

Execução do programa

Para uma melhor visualização dos resultados ao rodar o programa, foi necessário configurar os *outputs* com cores diferentes, uma vez que a saída continha muitas informações por conta das atuações das *threads*.

Ao executar o programa, a saída mostrará a ocorrência de diversos eventos:

1. `Aluno x entrou na sala` : a princípio 7 alunos entram na sala de uma vez, ao decorrer da execução isso varia, pois o *sleep* presente na função `estudo` é diferente para cada aluno, logo alguns alunos vão embora antes que outros;
2. `Sala cheia, aluno y vai ter que andar até a BCE`: quando todas as permissões do `sem_mesa` foram pegas, um aluno que tentar entrar não conseguirá, podendo este entrar uma outra vez em caso de tentar em um momento que tenha mesa disponível;
3. `Aluno x: vou comprar uma latinha, ainda tem z`: na função `estudo` temos `"Aluno estudando..."`, e pode ocorre de querer comprar um energético, quando isso ocorre ele estará na função `frigobar`, e caso tenha latas no frigobar, esta será a mensagem;

4. `Aluno x esperando o robo repor as latas`: caso onde o aluno deseja comprar o energético mas o frigobar está vazio. Ele aguarda até que a reposição seja feita.
5. `x jogando a latinha fora`: o aluno acabou de beber seu energético;
6. `Robo: vou levar as latas e recolher o dinheiro`: o robô foi acordado e vai realizar a reposição. Em caso contrário a mensagem será `"Robo dormindo, ainda tem lata no frigobar"`.

O vídeo com a explicação do trabalho pode ser acessado **neste link**.

4 Conclusão

Através deste trabalho foi possível utilizar os conhecimentos adquiridos ao longo da disciplina e fazer a utilização simultânea de vários recursos das bibliotecas *pthread* e *semaphore*. A princípio, foi desafiador formular um problema uma vez que o tema era aberto, além disso, uma vez que as regras de resolução também dependiam do problema a ser resolvido, foi necessária muita atenção na implementação para atender a estes requisitos. Apesar disso, foi possível desenvolver a solução do problema sem maiores dificuldades.

Referências

1. Slides da disciplina. Disponível em: https://cic.unb.br/ alchieri/disciplinas/graduacao/pc/