Relatório - Projeto 1

Thiago Veras Machado 16/0146682

June 3, 2019

1 Introdução

1.1 Descrição

O trabalho foi realizado com o objetivo da aplicação de algoritmos que resolvem problemas de comunicação entre processos. O problema elaborado consiste na simulação de um laboratório de informática. Existem 15 alunos e 15 veteranos, onde os alunos disputam uma das 20 vagas disponíveis no laboratório para estudar, no qual o aluno entra, estudar por um tempo e depois sai. A diversidade do problema se da a partir de que quando não há vagas, é criado uma fila do lado de fora do laboratório. Como os veteranos estão a mais tempo na universidade, eles querem se formar os mais rápido possível, os mesmo possuem prioridade para entrar, pois precisam estudar para passar nas matérias e se formarem. Caso não tenha vaga e não há veteranos querendo entrar, então a vaga é concebida para um calouro.

Outro ponto do problema é que, eventualmente, ocorre uma chuva muito forte na região, causando uma enchente no laboratório, e quando este evento acontece, o estudo de todos os alunos é interrompido e eles são obrigados e evacuar o laboratório. No desespero, todos os alunos começam a sair desesperadamente mas a porta só tem a capacidade de 1 aluno por vez, ou seja, eles ficam disputando quem consegue sair primeiro (sem prioridade). Após o laboratório ser evacuado, espera-se um pequeno tempo, representando que o laboratório secou e que os alunos podem voltar a entrar, fazendo assim, um ciclo infinito.

1.2 Visão Geral do Algoritmo

O programa consiste em 31 threads, das quais 15 são calouros, 15 veteranos e 1 thread que representa a enchente.

```
pthread_t threads[31];

for(int i = 0; i < 15; i++)
    pthread_create(&threads[i], NULL, veteranos, (void * )(long)i);

for(int i = 15; i < 30; i++)
    pthread_create(&threads[i], NULL, calouros, (void * )(long)i);

pthread_create(&threads[30], NULL, alagar, NULL);
```

As 31 threads rodam dentro do seu respectivo laço de repetição, onde as threads dos alunos esperam um tempo aleatório e chamam a função para entrar no laboratório.

```
void * veteranos(void * _id){
    int id = (int)(long)_id;
    while (true){
        sleep(rand()%espera + 1);
        entrar_veterano(id);
    }
}
```

A idéia por trás da função de entrar é estabelecida de seguinte forma, se for um veterano, ele incrementa a variável que marca a quantidade de veteranos que quer entrar no laboratório, depois dorme enquanto não tem vaga para entrar ou está tendo enchente. Após essa parte, o veterano decrementa a variável que marca quantos veteranos querem entrar no laboratório, pois agora ele vai entrar no laboratório. Após entrar o veterano simula um tempo de espera, como se estivesse estudando, caso no meio do estudo aconteça uma enchente, ele manda um sinal para todos os alunos estudando, fazendo com que ele pare e execute a parte de sair.

A saída dos alunos é definida por uma variável enchente, se ela estiver marcada, é porque está tendo uma enchente, logo o aluno deve sair correndo desesperadamente, caso contrário ele só sai normalmente.

A thread responsável por causar a enchente fica adormecida durante um tempo maior, pois a enchente acontece eventualmente de tempos e tempos. Sua função é acordar todos alunos que estão estudando e fazer com que eles evacuem o laboratório.

2 Implementação

2.1 Variáveis

As principais variáveis da memória compartilhada consiste em : 1 mutex, 4 condicionais e 3 inteiros.

```
pthread_mutex_t linf = PTHREAD_MUTEX_INITIALIZER;
pthread_cond_t cond_veterano_entrar = PTHREAD_COND_INITIALIZER;
pthread_cond_t cond_calouro_entrar = PTHREAD_COND_INITIALIZER;
pthread_cond_t fim_estudo = PTHREAD_COND_INITIALIZER;
pthread_cond_t fim_evacuar = PTHREAD_COND_INITIALIZER;
int veterano_entrar = 0, enchente = 0, vagas = 20;
```

O mutex linf é responsável por criar exclusão mútua da região crítica, no qual se tem a variável que marca a quantidade de vagas do laboratório, a variável que marca se há enchente ou não e a variável que marca quantos veteranos querem entrar, sendo assim, cada aluno irá acessar individualmente essas variáveis.

A variável condicional `cond_veterano_entrar` e `cond_calouro_entrar` é responsável para que o calouro e veterano possa dormir enquanto ele não pode entrar no laboratório, quando é liberado, um sinal é mandado para que a thread possa acordar e entrar no laboratório.

A variável `fim_estudo` está ligada com o tempo de espera dos alunos no linf, quando o aluno entra no laboratório, ele dorme durante um tempo aleatório ou caso aconteça uma enchente, pois neste caso é mandado um sinal para a variável `fim_estudo`, interrompendo o estudo do aluno para que ele possa evacuar o laboratório.

```
struct timespec tempo_estudando = cond_set_time(rand()%6 + 5);
pthread_cond_timedwait(&fim_estudo, &linf, &tempo_estudando);
```

A última variável de condição `fim_evacuar` é responsável para que a thread responsável por causar enchente possa dormir enquanto os alunos não tenham evacuado o laboratório.

```
if(vagas == 20){
    printf("LINF JA ESTAVA VAZIO!!!!!\n");
} else {
    pthread_cond_wait(&fim_evacuar, &linf);
}
```

2.2 Funções

```
void entrar_veterano(int id)
```

Essa função é responsável por fazer o aluno com um determinado id entrar no laboratório, ele segue os seguintes passos. Pega o lock do laboratório, incrementa a variável que indica quantos veteranos querem entrar, dorme enquanto a quantidade de vagas é igual a 0 ou a variável enchente é igual a 1. Depois dessa etapa o veterano já consegue entrar, logo ele decrementa a variável de que o veterano quer entrar, pois agora ele vai entrar, decrementa a variável de vagas, pois agora ela vai ser ocupada.

Próxima etapa é determinar quanto tempo o veterano irá dormir para poder colocar dentro da função `pthread_cond_timedwait()`, no qual o veterano irá dormir e acorda somente depois de receber um sinal ou passar um determinado tempo, estipulado anteriormente.

A ultima etapa dessa função é liberar o laboratório, para isso deve-se incrementar a variável de vagas, pois ele liberou a vaga, em seguida verifica se a variável de enchente está setada, se ela estiver, então o veterano foi acordado para poder evacuar o laboratório, então imprime a mensagem que o veterano saiu correndo, caso tenha chegado em 20 vagas, logo todos os alunos evacuaram o laboratório e assim mandar um sinal para a thread da enchente que acabou a evacuação. Caso não tenha enchente, ele imprime na tela que o veterano saiu andando, e caso chegou em 1 vaga, foi porque não havia vaga e o aluno liberou a vaga, logo ele acorda os outros que estavam esperando para entrar. No final de tudo ele libera o lock

do laboratório.

```
void entrar_calouro(int id)
```

Essa função é equivalente do veterano entrar, porém a única diferença é que o aluno dorme enquanto não tem vaga, está tendo enchente ou algum veterano quer entrar.

```
void * alagar()
```

Essa função é responsável por estabelecer quando acontecerá enchente ou não. Primeiro ele pega o lock do laboratório, seta a variável enchente como 1, verifica se existe aluno no laboratório, se sim, acorda todos os alunos que estavam estudando, para que possam sair, depois ele dorme enquanto o laboratório não for evacuado. Quando recebe o sinal de que acabou de evacuar o laboratório, ele imprime a mensagem que foi evacuado e dorme por um tempo simulando que o laboratório está secando, depois acorda todas as threads que estavam dormindo para entrar no laboratório.

3 Conclusão

3.1 Comentários Gerais

Podemos concluir que, com esse trabalho, pudemos colocar em prática o desenvolvimento de um algoritmo que resolva problema de comunicação entre processos, porém com uma complexidade um pouco maior comparado com os estudos dirigidos realizados em sala de aula. Além disso, foi minha primeira experiência resolvendo problemas concorrentes nesse nível de dificuldade, algo que é muito interessante, já que, praticamente tudo que envolve tecnologia, ultimamente, possui algum tipo de concorrência.

Com isso, foi uma ótima experiência, que me trouxa uma ampliação no conhecimento de desenvolvimento de algoritmo para realizar problemas concorrentes. Além de mostrar a eficiência que esse tipo de solução pode trazer.

3.2 Principais Dificuldades

Na minha visão, houve algumas partes que tiveram uma certa complexidade para ser executada. Como por exemplo definir a maneira correta de estruturar as threads para que cada função seja executada corretamente, neste aspecto, tive que pensar em um jeito para fazer o aluno entrar no laboratório e que ele possa ser interrompido para que evacue o laboratório. Primeiramente minha idéia era de dividir em 2 threads para cada aluno, onde 1 iria fazer ele entrar e outra fazer ele sair, porém essa idéia não era a mais correta possível, pois o correto seria apenas 1 thread para realizar esse tipo de tarefa.

Após alguns estudos e conversa com o professor em sala de aula, cheguei em uma solução no qual deverá ser feita uma verificação na hora da saída do aluno do laboratório, caso a variável enchente esteja em 1, então o aluno deve sair correndo, caso contrário ele pode sair normalmente. Com a solução desse problema, um outro ponto apareceu, como fazer o aluno

parar de dormir para pode ser evacuado instantaneamente, e assim, a solução foi a utilização da função `pthread_cond_timedwait()`, que se encaixou perfeitamente no trabalho, fazendo com que o aluno dormisse enquanto não recebesse o sinal da enchente ou então passasse o tempo de estudo do aluno, finalizado a solução do projeto.

4 Referências

- Slides passados em sala de aula : `https://cic.unb.br/~alchieri/`
- Manual de programação de Threads : `https://computing.llnl.gov/tutorials/pthreads/`
- Artigo da IBM sobre `pthread_cond_timedwait` : `https://www.ibm.com/support/knowledgecenter/en/ssw_ibm_i_72/apis/users_77.htm`