Universidade de Brasília
Departamento de Ciência da Computação
Disciplina: Programação Concorrente
Professor: Eduardo Adilio Pelinson Alchieri

Programação Concorrente

24 de outubro de 2021

Bruno Esteves Dalla Costa Filho, 17/0100863

1 Introdução

Este trabalho tem como objetivo aplicar a programação concorrente como principal conceito na implementação de um projeto com tema de escolha de cada aluno para aprofundar os conhecimentos nas suas ferramentas.

As ferramentas da programação concorrente tem como principal objetivo manter uma sincronização e concordância entre vários processos executados concorrentemente. Dentre várias dessas ferramentas e conceitos temos:

Locks: É um mecanismo de sincronização de processos/threads, em que processos/threads devem ser programados de modo que seus efeitos sobre os dados compartilhados sejam equivalentes serialmente.

Variáveis de Condição: Utilizadas em conjunto com os Locks, as Variáveis de Condição utilizam de conceitos como "Sleep"e "Wake Up"para manter a sincronização entre processos.

Semáforos: Um tipo de variável que tem 2 operações básicas DOWN e UP (generalização das primitivas sleep e wakeup), e ficam associadas a um recurso compartilhado, indicando quando o recurso está sendo acessado por um dos processos concorrentes.

Barreiras: Como o próprio nome sugere, uma barreira é um mecanismo de sincronização que determina um ponto na execução de uma aplicação onde vários processos ou threads esperam uns pelos outros.

2 Linguagem e Sistema Operacional

A linguagem de programação C foi utilizada na implementação.

O projeto foi compilado e executado no Windows utilizando o WSL (Subsistema do Windows para Linux).

3 Tema do Projeto

O tema do projeto foi escolhido com o intuito de simular a execução de operações dentro do mercado de ativos de criptomoedas com operações básicas de compra e venda, e acompanhar seus próprios valores sendo modificados à medida que tais operações são realizadas.

4 Implementação

Na implementação do projeto existem os "negociadores"que são simulados pelas threads do programa, onde vão poder realizar as operações básicas de compra e venda por meio da função "negociate()".

```
/* criando negociadores */
for (i = 0; i < NEG ; i++) {

    id = (int *) malloc(sizeof(int));
    *id = i;
    pthread_create(&negociators[i], NULL, negociate, (void *) (id));

}
```

Os ativos escolhidos foram 3 criptomoedas dentre as milhares existentes, sendo elas Bitcoin, Ethereum e AXS. Os valores dos respectivos ativos serão as variáveis compartilhadas pelas threads que terão seu acesso controlado pelos locks declarados no código.

```
//1 lock para cada ativo

pthread_mutex_t lock_BTC = PTHREAD_MUTEX_INITIALIZER;
pthread_mutex_t lock_ETH = PTHREAD_MUTEX_INITIALIZER;
pthread_mutex_t lock_AXS = PTHREAD_MUTEX_INITIALIZER;

//valor inicial dos ativos
double valor_BTC = 200000;
double valor_ETH = 16000;
double valor_AXS = 700;
```

No código não é realizado um join em nenhuma thread pois após a inicialização das threads é chamada a função "gerador()", que entra em loop infinito enquanto durar a execução do programa.

A função "gerador()"faz o papel de gerar as permissões que as threads negociadores consomem para realizar as operações de compra ou venda.

Essas permissões são representadas pela variável compartilhada "permissions"inicializada com o valor zero.

As variáveis condições foram utilizadas para haver uma comunicação e sincronização entre o gerador de permissões e as threads. O gerador dorme enquanto permissions ¿ 0, é acordado quando as permissões acabam e volta a dormir após gerar as mesmas, fazendo um broadcast para todos os negociadores dizendo que existem novas permissões para serem consumidas.

```c
void *gerador(){

  while(1){

    pthread_mutex_lock(&lock_PERMISSAO);

        while(permissions > 0){
            pthread_cond_wait(&cond_gerador, &lock_PERMISSAO);
        }

        printf("----------------------------\nGERADOR: Acordando!\n");

        int new_permissions = rand()%10;
        permissions += new_permissions;

        printf("GERADOR: Gerei %d permissões!\n----------------------------\n", new_permissions);

        sleep(5);

        if(permissions > 0) pthread_cond_broadcast(&cond_negociadores);

    pthread_mutex_unlock(&lock_PERMISSAO);

   }

}
```

```c
pthread_cond_t cond_negociadores = PTHREAD_COND_INITIALIZER;
pthread_cond_t cond_gerador = PTHREAD_COND_INITIALIZER;
```

Após uma thread consumir sua permissão, ela libera o lock responsável pela exclusão mútua da variável "permissions"e parte para a operação de compra ou venda do ativo escolhido aleatóriamente.

Cada ativo possui um lock responsável pela exclusão mútua da variável compartilhada que representa seu valor, para que mais de um ativo possa ser negociado ao mesmo tempo, desde que sejam distintos.

5 Conclusão

As ferramentas que a programação concorrente nos proporcionam provam-se ser um método muito confiável de implementação da concorrência entre processos. Promovendo a sincronização necessária entre threads/processos, e garantindo com segurança a exclusão mútua para áreas que são críticas no funcionamento do programa.

As threads, locks e variáveis de condição utilizadas no projeto mostraram-se efetivas para que suas execuções em paralelo não se conflitem em nenhum momento. Promovendo concorrência ao mesmo tempo é promovido um dinamismo e fluidez na execução dos processos concorrentes.

6 Referências

1. https://cic.unb.br/ alchieri/disciplinas/graduacao/pc