Universidade Federal Fluminense
Disciplina: Sistemas de Computação
Professor: Leandro Santiago
Projeto - Semáforos

Produtor Consumidor

Faça um programa em C, com pthreads, para linux, usando o modelo de produtor consumidor, que siga uma cadeia de execução P → CP1 → CP2 → CP3 → C. A comunicação entre os pares de classes de threads será feita por estruturas compartilhadas (um vetor de nome shared, sendo shared[0] usado por P e CP1, shared[1] por CP1 e CP2, shared[2] usado por CP2 e CP3 e shared[3] usado por CP3 e C). Cada elemento de shared possui os seguintes itens:

- Um buffer com 5 ponteiros para struct (chamaremos de S), cada uma contendo:
  - Nome - Nomes do arquivo de entrada associado a A e B.
  - A, B e C - Matrizes quadradas de ordem 10 (10 linhas e 10 colunas) com elementos double de cada coluna separados por vírgulas
  - V - Vetor com 10 elementos double
  - E - Variável double
- semáforos full, empty e mutex
- indices in e out do buffer

Considere as seguintes classes de threads e suas funções:

P - Thread produtora - Lê um arquivo (entrada.in) contendo uma lista de 50 arquivos de entrada (um nome de arquivo por linha), cada um contendo duas matrizes quadradas de ordem 10 de doubles. A cada arquivo lida, a thread produtora cria dinamicamente uma estrutura S, preenche o nome do arquivo de entrada, além de A e B e coloca o ponteiro para a estrutura S em shared[0].buffer[in] para ser processada pela etapa seguinte. Só teremos 1 instância desta thread.

CP1 - Thread Consumidora & Produtora 1 - Move shared[0]→buffer[out] para um ponteiro temporário, calcula C=A*B no elemento temporário e move o ponteiro temporário para shared[1]→buffer[in]. Teremos 5 instâncias desta thread.

CP2 - Thread Consumidora & Produtora 2 - Move shared[1]→buffer[out] para um ponteiro temporário, calcula V como a soma das colunas de C. (teremos 4 instâncias desta thread)

CP3 - Thread Consumidora & Produtora 3 - Move shared[2]→buffer[out] para shared[3]→buffer[in], calcula E como a soma dos elementos de V e move o ponteiro temporário para shared[2]→buffer[in]. Teremos 3 instâncias desta thread.

C - Thread consumidora - Escreve um arquivo (saida.out) contendo Nome, A, B, C, V e E para cada um dos arquivos de entrada, obedecendo o seguinte formato (substituir <variável> pelo valor da variável). Teremos 1 instância desta thread:

==========
Entrada: Nome;

A (formatado com linhas e colunas, usando espaços para separar elementos em uma linha)

B (formatado com linhas e colunas, usando espaços para separar elementos em uma linha)

C (formatado com linhas e colunas, usando espaços para separar elementos em uma linha)

V (cada elemento em uma linha)

E
==========

Dica: Deve haver um mecanismo para que cada thread saiba que não há mais tarefas a serem processadas. Uma forma fácil de fazer isso é criar um contador local na thread C que é inicializado com 0 e incrementado a cada elemento processado por C. Quando o contador chega em 50, a thread C "se mata". A thread pai, depois de criar todas as threads, faz join na thread C e depois do join mata todas as demais threads, que certamente já terminaram.

Dica 2: Código para multiplicação de matrizes (C = A*B)

```
for (i = 0; i < 10; i++) {
    for (j = 0; j < 10; j++) {
        c[i][j] = 0;
        for (k = 0; k < 10; k++) {
            c[i][j] += a[i][k] * b[k][j];
            /*Exemplo: c[0][0] = a[0][0]*b[0][0]+
                               a[0][1]*b[1][0]+
                               a[0][2]*b[2][0];
            */
        }
    }
}
```

Este trabalho corresponde a 20% (2 pontos) da média final. As seguintes funcionalidades extras podem ser implentadas para obter mais pontos (até 10 pontos extras - 1 ponto extra na média final)

1. (1,5 points) Tranformar a aplicação em um daemon, isto é, uma aplicação que inicia, cria um processo filho, se torna filho de outro processo na árvore do SO para que possa rodar em background, mude permissões de acesso e mate o processo pai. Fazer syslog de todo o sistema, inclusive no inicio e fim de cada estágio, informando sempre o nome do arquivo de origem que está sendo trabalhado. Fazer a parte de monitoramento de sinais. Em caso de recebimento de um sinal SIGTERM, o programa deve parar de monitorar novos arquivos e só terminar depois de finalizar todas as tarefas existentes. Em caso de SIGKILL, logar o evento e finalizar a aplicação fechando os arquivos abertos para que não sejam corrompidos. Você deve fazer isso através de variáveis compartilhadas com todas as threads dos estágios. Um exemplo será fornecido no site.

2. (1,5 points) Necessita do item 1 - Criar um programa de controle do deamon onde um menu é exibido ao usuário para que o mesmo possa alterar a quantidade de threads de cada tipo. O deamon recebe a mensagem, cria ou destrói threads e responde com o status da solicitação (OK, NOK, MAX)

   - OK - sucesso
   - NOK - Erro
   - MAX - numero de threads solicitado maior que o máximo definido – então serão criadas até MAX threads

   Se o número de threads for menor que o atual, threads serão avisadas para finalizar o trabalho corrente, não buscar mais nada no buffer e finalizarem.
   O programa de controle também permite desligar o deamon, enviando SIGTERM para o mesmo.
   O programa de controle também permite alterar o mapa de afinidade para cada classe

de threads.
Grupos interessados em implementar esta funcionalidade devem contactar o professor para obter dicas de como isto pode ser feito

3. (1,0 points) Fazer com que cada thread do tipo CP1 calcule a multiplicação em paralelo, com openMP, em 2 sub-threads.

4. (1,0 points) Fazer com que cada thread do tipo CP2 calcule a soma dos vetores em paralelo, com openMP, em 2 sub-threads.

5. (1,5 points) Permitir que a lista de arquivos de entrada tenha um número indeterminado de arquivos de entrada. Você precisa modificar o mecanismo de detecção de terminação para considerar esse número de elementos dinâmico. Para isto, use um contador global e uma flag protegida por um mutex. A thread P incrementa o contador sempre que um arquivo é processado e modifica a flag para 1 (usando mutex) quando termina de processar arquivos. A Thread C testa o flag (usando mutex) a cada execução e quando o flag é 1 ela passa a verificar se o contador local é igual ao local. Quando isto ocorre, C se mata e o pai pode matar as demais threads. Você deve escolher entre fazer esta implementação ou a implementação do item 6 (que vale mais pontos e é mais complexo).

6. (3,0 points) Permitir que a lista de arquivos de entrada tenha um número indeterminado de arquivos de entrada e que o controle de terminação seja distribuído. Você precisa encontrar uma forma de não usar a thread pai para matar todas as filhas, pois isto é um mecanismo centralizado e com menor escalabilidade. Pense em uma maneira da thread P, quando acabar de processar, notificar todas as threads CP1 que elas podem terminar. Depois que as threads CP1 terminarem de processar, elas devem notificar as threads CP2 e assim por diante. Você deve escolher entre fazer esta implementação ou a implementação do item 5 (que vale menos pontos e é menos complexo).

7. (2,0 points) Modifique o programa para que a thread P monitore um diretório, cujo caminho é fornecido pelo usário via linha de comando, procurando por arquivos com a extensão ".ready". Cada arquivo processado por P deve ter a extensão alterada para ".processing"no direrório de entrada. A thread C quando processa este arquivo deve mudar a sua extensão para ".done"no direrório de entrada e gerar o arquivo de saida como a extensão ".out"em outro diretório fornecido pelo usuário via linha de comando. Desta forma, se a aplicação travar ou se o computador desligar teremos como reiniciar o processamento dos arquivos com extensão ".processing"executando a aplicação com a opção -resume em linha de comando (a opção -normal será usada para uma inicialização normal).