Universidade Federal do Rio de Janeiro
Instituto de Matemática

Relatório do Segundo Trabalho
Computação Concorrente

Professora: Silvana
Aluno: Matheus Abrantes Gouvêa da Silva
DRE: 113170726

Introdução

O objetivo desse trabalho era de implementar uma solução para o problema dos leitores e escritores e um programa auxiliar para verificar sua corretude, em que não há inanição das threads, ou seja, não acontece a situação em que os leitores monopolizam o processador não permitindo que os escritores executem suas tarefas ou vice versa. Esse fenômeno também é conhecido como starvation e é comum de acontecer nesses tipos de problemas. Para que não aconteça starvation, as threads têm que ter prioridades iguais. As estratégias usadas na implementação da solução será discutida a seguir.

Implementação

Existem 3 arquivos na composição do programa: 2 arquivos .c, “principal.c”, que o código da função principal, e “modelo.c” que é um código que auxilia na implementação do programa de verificação de corretude do principal, e um header “funcoes_auxiliares.h”.

Para que fosse obedecida a restrição de que as threads deveriam ter prioridades iguais, foi implementada uma fila para gerenciar os pedidos de escrita e leitura. A fila funciona da seguinte maneira: existe um array circular, que possui tamanho igual a soma do número total de threads, (sendo assim no pior caso em que todas as threads façam o pedido de uma posição na fila, todas podem receber uma posição.) que funciona como um “gerenciador de senhas”, em que cada thread vai pedir uma senha de atendimento, e esperar sua vez de ser atendida. Se vários leitores pedirem uma senha, um seguido de outro, então o programa permite que todos eles possam ler ao mesmo tempo, até que um pedido de um escritor para ser atendido, seja feito. Todo escritor tem acesso exclusivo enquanto está executando sua função de escrita.

Com esse tipo de implementação, o balanceamento de prioridades é respeitado e não há monopolização do processador.

A implementação do programa auxiliar foi feita de uma maneira quase automática. Ele foi integrado ao programa principal, e será gerado como o resultado de uma execução do programa principal. Ele não é um programa separado, ele na verdade é o próprio log que já sai formatado para ser compilado e executado. Ele é criado da seguinte maneira: a função main do programa principal abre o arquivo “modelo.c”, que em seu conteúdo possui o include do header “funcoes_auxiliares”, e a definição da função main do programa auxiliar, e o copia para arquivo de log do programa principal, a partir daí, cada operação de cada thread vai ser registrada no log como uma chamada de função, seguido de comentários indicando o número do contador de operações que já foram feitas e o valor da variável compartilhada pelas threads. No fim da execução, a main adiciona uma chamada de printf pra indicar que se o programa auxiliar chegou naquele ponto sem erro, então o programa principal executou corretamente. O arquivo .c do log então é gerado.

As funções chamadas pelo código de verificação do log, estão definidas no header “funcoes_auxiliares.h”. São quatro funções: entra_leitura(), entra_escrita(), sai_leitura() e sai_escrita().

A função entra_leitura() verifica se tem algum escritor escrevendo, caso tenha então o programa acusa um erro, pois não pode ter um leitor lendo se houver um escritor escrevendo. Se não tiver, então o leitor que chamou a função tinha permissão de fazer a leitura, e a variável contadora do número de leitores lendo é incrementada.

A função entra_escrita() verifica se há algum leitor ou escritor executando, caso haja então, o programa acusa um erro, pois quando um escritor vai escrever, não podem haver nem leitor lendo nem outro escritor escrevendo. Caso contrário, então o escritor tinha permissão de executar sua escrita, e a variável contadora de escritores é incrementada.

As funções sai_leitura() e sai_escrita() apenas decrementam o contador de threads leitoras e escritoras, respectivamente.

Todas as funções desse header incrementam o contador de operações.

Algoritmo

Os passos executados pelo programa são os seguintes:

- A função main inicializa as variáveis usando as entradas do usuário e o arquivo de log.
- Dispara as threads.
- As threads leitoras:
  - Define as variáveis locais.
  - Abre o arquivo de saída.
  - Verifica se o arquivo foi aberto.
  - Pega um número na fila de atendimento.
  - Verifica se pode fazer a leitura.
  - Quando puder, incrementa o contador de atendimento e sinaliza os leitores, se o próximo na fila for um leitor, ele também poderá fazer a leitura, e assim por diante.
  - Incrementa o contador sinalizando quantos leitores estão escrevendo.
  - Lê, depois faz uma espera de um microsegundo.
  - Verifica se é o último leitor de algum grupo que estava lendo.

        - Se for, então quer dizer que o próximo a ser atendido é um escritor, então sinaliza os escritores.
        - Repete até se esgotar o número de leituras.
    - As threads escritoras:
        - Define as variáveis locais.
        - Pega um número para esperar atendimento.
        - Verifica se já pode ser atendido.
        - Quando puder, escreve na variável compartilhada.
        - Espera por um microsegundo.
        - Incrementa o contador de atendimento.
        - Sinaliza todas as threads.
        - Repete até que esgotar o número de escritas.
    - A função main espera as threads terminarem de executar
    - Finaliza a criação do código de log.
    - Imprime uma mensagem de finalização.
    - Termina o programa.

Testes e Observações

Pela maneira como a solução foi implementada, a condição de não acontecer starvation é naturalmente respeitada, então não foi preciso tomar nenhum tipo de medida especial para evitar starvation.

O programa foi testado primeiramente sem usar as funções de sleep, porém quando começava sua execução, a primeira thread a ser criada, geralmente conseguia fazer muitas operações seguidas pois não precisava competir pelo processador por um tempo, e quando as operações eram poucas (<10), então a thread terminava antes de que outras threads fossem criadas. Mas quando eram um número grande de operações, e tinham muitas threads competindo então as operações se estabilizavam. Nesse caso também (sem usar a função de sleep), haviam mais mudanças no valor lido pelas threads leitoras.

Quando foi adicionado a chamada à função de sleep após as operações de escrita e leitura, as operações foram feitas de em uma ordem mais “comportada”, e curiosamente, havia pouquíssimas mudanças no valor lido pelas threads leitoras, porque a ordem de execução era estabalecida desde o começo do programa, então acontecia de uma mesma thread leitora, sempre ler o valor escrito por uma mesma thread escritora, pelo ordem de execução.