Trabalho II de Computação Concorrente:

Gabriel da Fonseca Ottoboni Pinho - DRE 119043838
Rodrigo Delpreti de Siqueira - DRE 119022353

21/05/2021

1 Leitores e escritores

1.1 O problema

O modelo de leitores e escritores tem como objetivo coordenar o acesso a um certo recurso entre várias threads. No caso desse trabalho, o recurso é o array que contém as medições dos sensores.

A necessidade de coordenar os acessos a esse array existe por conta de condições de corrida que podem ocorrer quando uma thread tenta ler/escrever enquanto outra thread já estava escrevendo no array. Dessa forma, temos os seguintes requisitos:

- Se alguma thread estiver escrevedo, nenhuma outra thread pode ler ou escrever.
- Se uma ou mais threads estiverem lendo, nenhuma outra thread pode escrever.

Sabendo disso, é necessário um sistema no qual cada thread pede permissão para acessar o array e, caso o acesso não seja possível naquele momento, a thread é bloqueada.

1.2 A solução

A solução implementada no trabalho consiste em um *struct* `Rw` que armazena os números de threads atualmente lendo e escrevendo. Esses números são atualizados por meio de funções que as threads chamam antes e depois de iniciar suas tarefas.

Internamente, o controle de acesso é feito utilizando os *structs* `pthread_mutex_t` e `pthread_cond_t`. O primeiro garante exclusão mútua durante o acesso às variáveis que armazenam os números de leitores e escritores, enquanto que o segundo é utilizado para controlar o bloqueio das threads que aguardam permissão para ler/escrever.

```
Rw rw;
rw_init(&rw);

rw_get_read(&rw);
/* Ler à vontade */
rw_release_read(&rw);

rw_get_write(&rw);
/* Escrever à vontade */
rw_release_write(&rw);

rw_destroy(&rw);
```

Como mostrado no exemplo acima, a função `rw_init` inicializa os campos do *struct*, enquanto que `rw_destroy` faz o oposto. Antes de ler, a thread deve chamar `rw_get_read`, que retornará imediatamente se nenhuma outra estiver escrevendo. Caso contrário, a thread ficará bloqueada até que nenhuma outra thread esteja escrevendo. Ao acabar de ler, a thread deve chamar `rw_release_read`. As funções `rw_get_write` e `rw_release_write` funcionam de forma análoga.

Quando há alguma thread aguardando permissão para escrever, a prioridade para escrita é garantida de três formas:

- Assim que o último leitor chama `rw_release_read`, é garantido que uma thread será liberada para escrita.
- Assim que um escritor chama `rw_release_write`, é garantido que uma thread será liberada para escrita.
- Se há um escritor bloqueado, todos os novos leitores também serão bloqueados.

1.3 Testes

Os testes realizados sobre a implementação que criamos para o padrão de leitores/escritores visam atender os seguintes requisitos:

1. mais de um leitor pode ler ao mesmo tempo uma área de dados compartilhada;
2. apenas um escritor pode escrever de cada vez nessa mesma área;
3. enquanto o escritor está escrevendo, os leitores não podem ler.
4. Quando uma escritora está escrevendo e há ambas threads leitoras e escritoras bloqueadas, as escritoras devem ter prioridade.
5. Quando há leitoras lendo e uma escritora é bloqueada, futuras threads leitoras também precisam ser bloqueadas.

Estes requisitos são avaliados individualmente, no arquivo separado *rwtest.c*.

2 Monitoramento de temperatura

A solução utiliza uma struct para guardar os valores respectivos de temperatura para cada thread. Estas recebem o próprio id e um ponteiro que a solução de leitores e escritores exige. Esse ponteiro é utilizado apenas nos testes, não faz parte da lógica do programa principal.

O programa irá gerar valores aleatórios de temperatura, dentro do intervalo de 25 a 40 graus Celsius, utilizando a função `get_temperature_rand`. Os valores são gerados com uma casa decimal, conforme orientação do trabalho. Intuitivamente, entendemos que esta função não condiz com uma situação real, onde a variação da temperatura ocorreria de forma contínua. Todavia, mantivemos este método por ser de fácil imlementação.

A função `check_array` é executada pela thread leitora (atuador). Ela verifica se haverá um caso de sinal vermelho, amarelo ou condição normal, com base nos requisitos do trabalho. Além disso, ela exibe a média

A função `add_temp` é executada pela thread escritora (sensor). Ela insere o valor medido no espaço de memória compartilhado.

É permitido ao usuário interagir com a aplicação definindo o número de threads. Após mandar rodar, a aplicação executará por tempo indefinido, até ser encerrada por Ctrl+C.

3 Discussão

Durante a execução, observamos que o Alerta Vermelho é raramente acionado. Um dos motivos identificados vem do que discutimos anteriormente: a função que gera os valores de temperatura não trata o intervalo de valores como contínuo. Isso torna muito mais improvável a ocorrência de 5 leituras seguidas com temperatura superior a 35 graus.

Além disso, o alerta amarelo também demora algumas execuções para aparecer, pois no início não há escritas o suficiente para que tais condições sejam atendidas.