Sistemas Operacionais

Relatório do Trabalho: Produtor Consumidor em Pthreads

Prof. Gerson Cavalheiro

Thiago Heron Albano de Ávila

1. Descrição do Trabalho

Implementar um programa do tipo Produtor / Consumidor em Pthreads. Sendo que os produtores e os consumidores devem ser implementados com threads e compartilham um buffer de tamanho limitado. Os produtores inserem no buffer e os consumidor retiram. A política de manipulação do buffer é **FIFO (first in, first out)**

2. Dados de Entrada

A configuração dos dados de entrada do programa é feita através de **constantes** que podem ser configuradas no início do arquivo ***main.c***

- **Buffer Size:** Tamanho do Buffer
- **Number of Producer:** Número de Produtores
- **Number of Consumer:** Número de Consumidor
- **Iterations of Producer:** Número de iterações que será feita por cada thread produtor.

3. Configurações do Trabalho

O seguinte trabalho foi implementado através de uma lista encadeada que representa um ***buffer*** de tamanho limitado, no qual possui métodos básicos para a manipulação do mesmo, sendo que em relação a inserção e remoção de elementos é feita através da manipulação FIFO (first in, first out).

Os demais parâmetros de entrada, tanto do produtor quando do consumidor podem ser configurados no início do código, onde são definidas as constantes, como explicado no **tópico 2.**

Para a compilação do código, basta executar a seguinte linha de comando no seu respectivo termina:

gcc main.c -o main -lpthread

4. Funcionamento do Produtor

Conforme a descrição do trabalho, os produtores recebem dois valores que representam o número de iterações que será feito por cada *producer thread* e o endereço de memória do *buffer*. Sendo que, o número de iterações é passada como parâmetro para a função do produtor, enquanto o *buffer* como foi criado na memória estática não é necessário. A passagem desse parâmetro é feito através do ***thread_arg*** denominado ***argumentsProducer***.

Logo, a função do produtor realiza um laço com o número de iterações passado, em cada uma dessas, utiliza ***pthread_mutex_lock*** para acessar a sessão crítica e fazer a verificação se o *buffer* está cheio. Enquanto o buffer estiver cheio, utiliza-se ***pthread_cond_wait***, para bloqueá-lo até que haja espaço para inserir um elemento no *buffer*.

Caso contrário, é gerado um número através da função *drand48()* que por sua vez é inserido no buffer, em seguida é emitido um sinal para a Variável de Condição do Consumidor através da função ***pthread_cond_signal(&canConsume)*** para sinalizar que há algum elemento no buffer para consumir. Pois os consumidores caso estejam vazio, estarão esperando a condição ser satisfeita.

Para finalizar, é liberado o mutex através da função **pthread_mutex_unlock** para as demais threads acessar a sessão crítica.

5. Funcionamento do Consumidor

De acordo com a descrição do trabalho, os consumidores devem receber um parâmetro que é o endereço de memória do *buffer*, como explicado anteriormente, não foi necessário uma vez que o *buffer* está na memória estática.

Os consumidores executam um laço infinito onde primeiramente pega-se o mutex através da **pthread_mutex_lock** para iniciar a verificação, uma vez que, enquanto o buffer estiver vazio, o consumidor é bloqueado através da função **pthread_cond_wait** que aguarda a condição ser satisfeita e o sinal dado pelo produtor.

Caso contrário, ou seja, o *buffer* não está vazio, ele irá retirar o primeiro elemento do buffer através da função **pop()** que foi implementada utilizando a manipulação **FIFO.** Logo, é emitido um *sinal* através da **pthread_cond_signal** para o produtor e liberado o mutex. Portanto, é feito a verificação do número removido, caso este seja igual à -1 é feito um *break* para terminar a thread em questão.

Por fim, também é verificado se o respectivo número é primo através da função implementada **isPrimeNumber,** é retornado o id da thread e respectivo número primo no formado que foi solicitado.

6. Comunicação das Variáveis de Condição

A alternância da manipulação da sessão crítica e para manter as threads bloqueadas do produtor e consumidor deve-se a utilização das variáveis de condição, como a **pthread_cond_wait** para bloquear a *thread* que está utilizando a sessão crítica passando como parâmetro a variável de condição seja **canConsume** ou **canProducer**. E por fim, utilizando para **pthread_cond_signal()** passando como parâmetro **canConsume** ou **canProducer** para indicar que há um elemento no *buffer* para ser consumido ou o *buffer* está vazio para inserção de novos elementos.

7. Visualização das Threads através do HTOP

Para a visualização das threads foi utilizado o HTOP que permite identificar o número de threads que está sendo utilizado, além da porcentagem utilizada por cada CPU.

Para ter como base, foi feito um ***htop*** antes de executar o código do produtor consumidor, dado pela **Imagem 1.**

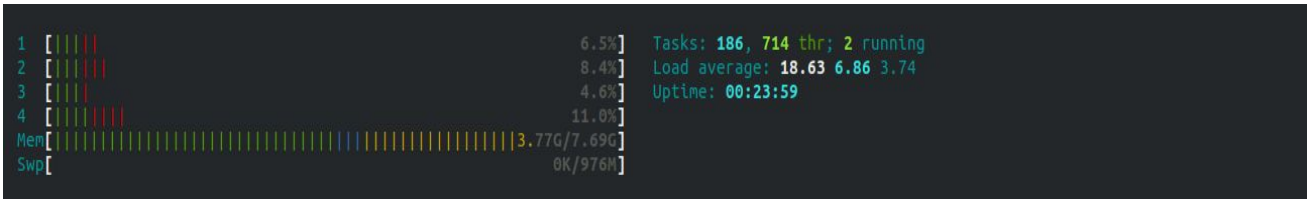

Imagem 1. Antes do produtor / consumidor

O primeiro teste foi feito com os dados da **Tabela 1** abaixo, que o mesmo é referente a **Imagem 2**, onde é possível visualizar a porcentagem usada pelos CPU e a quantidade de número de threads passando de 714 para 2715, sendo 1000 dado ao produtor e 1000 ao consumidor.

Buffer Size	100
Number of Producer	1000
Number of Consumer	1000
Iterations of Producer	10000

Tabela 1. Dados do primeiro teste do produtor consumidor

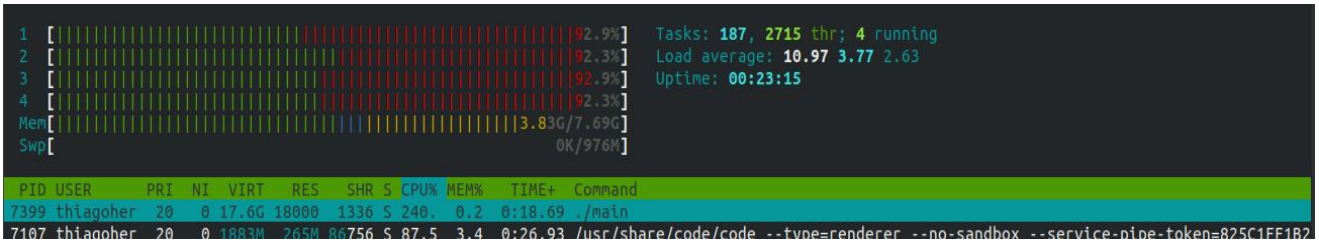

Imagem 2. Primeiro teste de produtor e consumidor com dados da Tabela 1.

O segundo teste foi feito com os dados da **Tabela 2,** que é possível visualizar pela **Imagem 3.** O mesmo é possível visualizar a diferença no número de threads o que já era esperado.

Buffer Size	**100**
Number of Producer	**2000**
Number of Consumer	**2000**
Iterations of Producer	**10000**

Tabela 2. Dados do primeiro teste do produtor consumidor

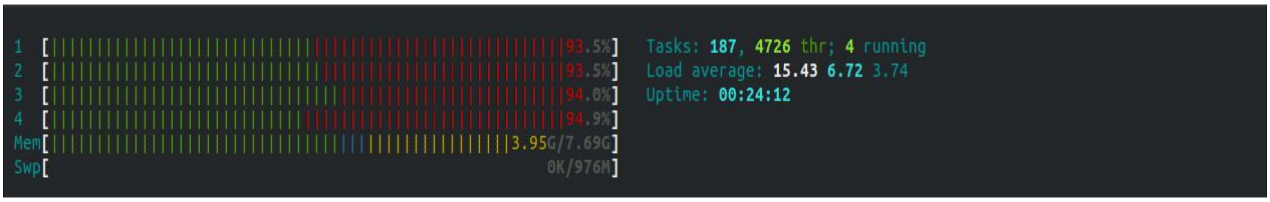

Imagem 3. Segundo teste de produtor e consumidor com dados da Tabela 2.