Relatório Experimento 6
Sistemas Operacionais 1 – Turma 3 Grupo 8

Professor Tobar

Bruno de André Mazzoco RA:02150522
César Henrique Kallas RA: 02099224

Introdução

O Experimento visa o entendimento do uso de threads em conjunto com semáforos. Esse experimento é uma continuação de dois anteriores, melhor dizendo, é a combinação deles.

A idéia por trás de se juntar semáforos com threads é poder ter um maior controle sobre o trecho de código ligados as threads iniciadas. Sendo assim, os semáforos usados no experimento, irão controlar de maneira que não haja desentendimento entre as threads.

As threads são denominados de processos leves que possuem e usam os mesmos recursos dos processos que as criaram sendo que com o uso de threads é possível compartilhar dados e recursos em geral sem a necessidade de mecanismos de comunicação entre processos (IPCs) o que determina uma melhor eficiência em termos de mudança de contexto.

Nesse experimento foi implementado o famoso problema dos produtores e consumidores, aonde são criados consumidores e produtores (número de threads) que disputaram um buffer controlados por um semáforo.

Programa Exemplo

O programa exemplo implementa um solução de produtor e consumidor. O problema de produtores e consumidores surge quanto ao acesso do buffer deferido. Sendo assim, consumidores só podem ter acesso para consumir caso tenha conteúdo o buffer, esse mesmo conteúdo é dado (colocado) no buffer pelo produtor. Pode acontecer de não haver mais espaço no buffer para ser inserido pelo produtor, bem como não haja nada para ser consumido pelo consumidor, surgindo o problema.

Fora isso, pode haver a disputa interna pelos produtores e consumidores, pois no exemplo, são usados threads, sendo que podem haver vários produtores e vários consumidores. Assim deve haver um controle entre os consumidores e os produtores.

No programa exemplo são criados N produtores e consumidores, o N é dado pelo número de threads. Logo após isso, produtores e consumidores entram em condição de disputa, colocando e retirando do buffer X vezes

(número de iterações).

O programa principal (main), executa um comando de espera pelo término dos consumidores e produtores, logo que acabam, calcula o tempo gasto por eles (produtores e consumidores) e imprime na tela.

Tarefas

---

1) Executar o programa exemplo com um número diferente de produtores e consumidores. Alterando o número de threads para múltiplos de 5. Faça isso 10 vezes, sendo a primeira com NO_THREADS igual a um. Anote os tempos obtidos e coloque-os em uma tabela.

Resultados Tarefa 1 - experiment-x2

Execução	Número de threads	Total time (s)
1	1	0.0762009993
2	5	0.6563010216
3	10	1.4415140152
4	15	2.4984350204
5	20	4.1168751717
6	25	5.1759819984
7	30	7.1658082008
8	35	8.0234251022
9	40	10.7157115936
10	45	11.7016248703

Na décima execução obtivemos problemas executando na Puc, o programa exemplo original travou, consumindo todo o processamento da máquina, timeout > 2m e número de tentativas = 3. Veja que a tabela não foi gerada usando dados dos computadores da Puc, por isso há resultado na décima execução.

2) Fazer alterações no programa exemplo, de maneira a existirem dois mutex, um para produtores e outro para consumidores. Execute esse programa modificado com um número diferente de produtores e consumidores. Altere, para isso, o valor de NO_THREADS para múltiplos de cinco. Faça isso 10 vezes, sendo a primeira com NO_THREADS igual a um. Anote os tempos obtidos e coloque-os em uma tabela.

Resultados Tarefa 2 - experiment-x2TwoMutex

Execução	Número de threads	Total time(s)
1	1	0.0590850003
2	5	0.4070050120

Execução	Número de threads	Total time(s)
3	10	1.3143210411
4	15	1.8983089924
5	20	2.8302209377
6	25	4.7410078049
7	30	6.6291022301
8	35	7.2348818779
9	40	8.5050086975
10	45	11.1698627472

Nas execuções 8, 9 e 10 obtivemos também o mesmo problema da tarefa 1, o programa exemplo modificado não respondeu, ficando em provável loop ocupando todo o processamento da máquina.

A fato interessante, os resultados da tabela existem porque usamos o mesmo programa exemplo em outra versão de kernel do linux, note que o problema não ocorreu (versão testada 2.4.26).

3) Implemente a segunda solução vista na teoria; para isso, utilize mutex para obter exclusão mútua entre produtores e entre consumidores. Utilize semáforos para controlar as situações de buffer cheio e buffer vazio. O tamanho do buffer deve ser o mesmo do programa exemplo. Utilize a função módulo para fazer os índices percorrerem o buffer de maneira circular. Calcule a duração da execução da mesma maneira que no programa exemplo. Execute esse novo programa com um número diferente de produtores e consumidores. Use, para isso, o valor de NO_THREADS para múltiplos de cinco. Faça isso 10 vezes, sendo a primeira com NO_THREADS igual a um. Anote os tempos obtidos e coloque-os em uma tabela.

Resultados Tarefa 3 - experiment-x2Sem

Execução	Número de threads	Total time (s)
1	1	30/12/99
2	5	0.2111109942
3	10	0.4351620078
4	15	1.1524089575
5	20	1.9350529909
6	25	3.1009609699
7	30	4.8488459587
8	35	4.9518339634
9	40	4.7619009018
10	45	5.2406721115

Análise dos resultados

Podemos notar que as tarefas 1 e 2 resultaram em tempos praticamente idênticos, mas a condição de corrida entre os dois programas são diferentes.

No programa exemplo 2 é implementado um mutex a mais, sendo assim, pode haver um consumidor ou produtor trabalhando ao mesmo tempo, sem depender da execução do outro lado de produção ou consumação. Com isso o tempo gasto deveria ser menor, porém por ser pequeno o buffer e não há maneira exata de medir o tempo, os resultados podem ficar praticamente iguais, como aconteceu.

Na tarefa 3, notamos que o tempo caí de maneira notável, isso porque os semáforos permitem que haja vários produtores e consumidores trabalhando ao mesmo tempo, algo que não acontecia com dois mutex aonde só podia haver um produtor e um consumidor trabalhando ao mesmo tempo.

Quando há vários produzindo e vários consumindo, de apenas um fonte, há tendência do tempo diminuir, e isso foi constatado.

Conclusão

Podemos concluir que há grande benefício em usar semáforos na condição de corrida entre threads, pois o mesmo permite várias threads se disputarem entre si, não apenas um, como ocorre usando mutex.

O experimento foi de grande proveito por parte de conhecimento, principalmente por usar semáforos permitindo vários “down”.