Relatório Experimento 5
Sistemas Operacionais 1 – Turma 3 Grupo 8

Professor Tobar

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

Introdução

O experimento visa entender o que são threads, como elas funcionam e no que elas são diferentes dos processos comuns.

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.

Para este experimento foi adotado o problema clássico do Produtor-Consumidor implementado com threads, para melhor entendimento das mesmas e análise de condições de disputa de recursos entre elas, que assim como os processos precisam ser sincronizados.

Programa Exemplo

O programa exemplo é uma implementação com threads do Problema do Produtor-Consumidor, onde basicamente, um processo (no caso thread) denominado de Produtor “produz” um item a ser colocado num buffer (no caso do experimento um vetor circular) e um outro processo (thread) denominado de Consumidor, “consome” esse item produzido desse mesmo buffer.

Existem três ponteiros que fazem o controle desse buffer: (rp) que aponta para a posição do buffer onde deve ocorrer a próxima retirada (reading pointer); (wp) que aponta para a posição do buffer onde deve ocorrer o próximo armazenamento (writing pointer) e (start) que aponta para o inicio do buffer (vetor).

Quando os dois primeiros são incrementados e quando realizada ação que cada um representa eles são incrementados e quando ultrapassam o limite do buffer, são apontados para o mesmo endereço em que o ponteiro start aponta garantindo assim, a circularidade do buffer.

Inicialmente, o produtor e consumidor são criados por uma thread, que é a própria função main () do programa, um numero

determinado de vezes (no caso 1 thread para Produtor e 1 para Consumidor). A thread criadora (main()) é então terminada independentemente das outras threads.

Resultados do Programa Exemplo

Número de Threads	Número Tentativas Mal Sucedidas Produtor	Número de Tentativas Mal Sucedidas Consumidor	Tempo Threads(s)
1	38168	274686	0.128584027290

Tarefas

a) Descrever o que ocorre com as threads criadas, se a rotina principal (main()) terminar sem ser através de pthread_exit.

Quando a função main () é terminada sem o pthread_exit(null) , o programa termina antes que as duas threads Produtor e Consumidor sejam criadas.

Programa Modificado

Foram formuladas as seguintes perguntas para serem respondidas através da modificação do programa exemplo:

1) Porque algumas das tentativas de armazenar em produce() e de retirar em consume() não dão certo?

2) a) Quantas vezes não foi possível armazenar valores no buffer?
b) E quantas vezes não foi possível retirar?

3) Quanto tempo demorou para todas as threads realizarem seus processamentos?

Modificações Realizadas

As seguintes modificações foram realizadas para solucionar cada um dos itens anteriores:

1) Foi aumentado o tamanho do Buffer em relação ao numero de iterações em que a função produce() e consume() realizam. O tamanho foi aumentado de SIZEOFBUFFER igual ao NO_OF_ITERATIONS mais 1. A ordem de criação das threads também foi invertida na implementação, colocando-se primeiro a criação do Produtor e depois, a criação do Consumidor;

2) Foi colocada uma variável contadora dentro de cada função (consume() e produce()) para contar cada vez que essas NÃO conseguissem realizar suas respectivas operações de remoção e inserção.

3) Foi usada uma tomada de tempo através da função getimeofday() antes da criação das threads e ao final de cada uma das threads criadas (Produtor, Consumidor e Main()).O tempo dessas ultimas, foram comparados e o maior dentre elas foi usado para calcular a diferença entre o tempo de criação e termino de TODAS as threads.

Resultados do Programa Modificado
Parte 1 (SEM EXLCUSÃO)

```
#define SIZEOFBUFFER    100001
#define NO_OF_ITERATIONS 100000
#define MICRO_PER_SECOND 1000000
```

No de Threads	Número de Tentativas Mal Sucedidas Produtor	Número Tentativas Mal Sucedidas Consumidor	Tempo Threads (s)
1	0	0	0.077728003263
5	172115	194665	0.323312997818
10	291231	335022	0.733669042587
15	381811	289251	0.742079973221
20	573209	430549	0.082639992237
25	676746	625918	0.908218979836
30	832126	609702	0.236937999725
35	532630	497146	0.030726969242
40	393312	621256	0.673270940781
45	571272	607022	0.961423993111

Respostas das Perguntas e Analise das Modificações

1) Porque algumas das tentativas de armazenar em produce() e de retirar em consume() não dão certo?

Podemos supor que o as tentativas de colocar do buffer que não deram certo são devido ao tamanho do buffer ser menor (metade no caso) em relação ao numero de iterações (numero de vezes) em que é realizado a função myadd().

As tentativas más sucedidas de remoção são devido ao fato também de o tamanho do buffer ser menor que o numero de iterações (numero de vezes) em que é realizada a função myremove().

Sendo assim, o Consumidor estaria tentando retirar um número maior que o possível de itens armazenados no buffer. Porem é perceptível que essas tentativas más sucedidas sejam pelo fato do Consumidor ter sido criado primeiro pela implementação, o que resulta numa tentativa de consumo de itens que ainda não foram produzidos até ao momento da criação da thread Produtor.

2) a) Quantas vezes não foi possível armazenar valores no buffer?

As tentativas mal sucedidas foram muito maiores que as bem sucedidas no caso de armazenamento do item (ordem de 5 casas decimais);

b) E quantas vezes não foi possível retirar?

As tentativas mal sucedidas foram muito maiores que as bem sucedidas no caso de remoção do item (ordem de 5 casas decimais);

Comparando as duas situações, pode-se perceber que as tentativas não sucedidas do Consumidor em consumir o item foi maior que as tentativas não sucedidas do Produtor e produzir (quase 3 vezes maior) o item no buffer devido a razão dita anteriormente de que o Consumidor é criado primeiro.

3) Quanto tempo demorou para todas as threads realizarem seus processamentos?

Podemos perceber claramente que a thread main () é a primeira a ser finalizada. Sendo a mesma, a thread que calcula os demais tempo, foi necessário que ele durmisse um tempo determinado para que desse tempo das outras threads serem finalizadas e seus respectivos tempos de termino fossem tomados.

Percebe-se que não há uma linearidade com relação ao tempo das threads já que a execução é de forma concorrente. O que pode se perceber foi o aumento do número de tentativas mal sucedidas (tanto no produtor quanto no consumidor) sendo que isso deve-se ao fato da implementação estar SEM um mecanismo de exclusão mutua que permita que só um thread possua manipular o buffer por vez.

Resultados do Programa Modificado
Parte 1 (COM EXLCUSÃO)

```
#define SIZEOFBUFFER    100001
#define NO_OF_ITERATIONS 100000
```

#define MICRO_PER_SECOND 1000000

No de Threads	Número de Tentativas Mal Sucedidas Produtor	Número de Tentativas Mal Sucedidas Consumidor	Tempo Threads (s)
1	0	0	0.110571002960
2	102575	76029	0.273699998856
3	34300	97232	0.876002013683
5	35188	265327	0.885074019432
7	98628	6284	0.973594009876
9	39710	17081	0.893594009876
10	67200	1150	0.989959001541
12	52181	9855	0.955000136832
14	110241	14851	0.970201201748
15	98521	11233	0.950605988503

Analise das Modificações – Parte 2

Notamos que não há uma sequência lógica de tentativas sem êxito tanto do produtor quanto do consumidor ao realizar sua tarefa proposta. Entretando o tempo das threads são aparentemente sequênciais, o que nos leva a crer que quanto mais threads na máquina, maior será o uso do processador, levando uma maior concorrência de acesso ao buffer, acarretando em um atraso nos produtores e consumidores, que estarão concorrendo entre si, o processamento se torna mais lento e então o término de todas threads aumenta.

Conclusão

Podemos concluir que thread é uma ótima ferramenta para se trabalhar quando há necessidade de execução de código paralela, com a grande vantagem de poder usar além do código os dados compartilhados, como variáveis e macro sem a necessidade de um mecanismo para isso.

A desvantagem a nosso ponto de vista, é o desconforto em controlar o que está sendo processado no momento, bem como o controle a alteração os dados em variáveis usadas em várias threads ao mesmo tempo, porém já estudando mecanismo de exclusão mútua, esse problema se resolve.

Threads são uma ótima alternativa ao uso de vários processos ao nosso ponto de ver, simplificando vários pontos de programação.