Pontifícia Universidade Católica do Rio de Janeiro

Lab 3 - Threads

Sistemas Operacionais

Alunos	Tharik Lourenço
	Suemy Inagaki
Professor	Luiz Fernando Seibel

Rio de Janeiro, 23 de abril de 2021

Conteúdo

1 Problema 1

1) Implemente um programa que crie 2 threads:

- Uma delas (contCrescente) contará de 1 a N=60 (com pausas de T1=2 segundos) imprimindo o valor de contCrescente. A outra (contDecrescente) de M=60 a 1 (com pausas de T2=1 segundo) imprimindo o valor de contDecrescente.
- Acrescente agora uma variável global inteira que é inicializada com zero e é incrementada e impressa na tela por cada thread. Verifique que as tarefas manipulam a mesma variável. Compile com a opção –lpthread

1.1 Código Fonte

```
/*
Tharik Lourenço
Suemy Inagaki
Lab 03 Threads, questão 1
Entrega Final
Sistemas Operacionais
*/


#include <pthread.h>
#include <stdio.h>
#include<unistd.h>
#define NUM_THREADS 2

int count = 0;

void *contCrescente(void* threadid){
    int c = 0;
    for(int i=1;i<=60;i++){
        sleep(2);
        c+=5;
        printf("Thread 0:%d\n",c);
    }
    pthread_exit(NULL);
}
```

```
void *contDecrescente(void* threadid){
        int d = 300;
    for(int j= 1;j<=60;j++){
        sleep(1);
        d-=5;
        printf("Thread 1:%d\n",d);
        }
    pthread_exit(NULL);
}

void *incrementa(void* threadid){
    for(int i=1;i<=2000;i++){
        sleep(0.1);
        count+=5;
        printf("Thread 0:%d\n",count);
        }
    pthread_exit(NULL);
}
void *decrementa(void* threadid){
     for(int j= 1;j<=2000;j++){
        sleep(0.1);
        count-=5;
        printf("Thread 1:%d\n",count);
        }
    pthread_exit(NULL);
}


int main()
{
        pthread_t threads[NUM_THREADS];
        int t;
        /*Verificando concorrencia*/
        printf("Creating thread 0\n");
        pthread_create(&threads[0], NULL, contCrescente, (void *)0);
        printf("Creating thread 1\n");
        pthread_create(&threads[1], NULL, contDecrescente, (void *)1);
        for(t=0; t < NUM_THREADS; t++)
                pthread_join(threads[t],NULL);
```

```
        /*Manipulando variavel global*/
        printf("Creating thread 0\n");
        pthread_create(&threads[0], NULL, incrementa, (void *)0);
        printf("Creating thread 1\n");
        pthread_create(&threads[1], NULL, decrementa, (void *)1);
        for(t=0; t < NUM_THREADS; t++)
                pthread_join(threads[t],NULL);
        printf("Variavel global: %d\n", count);
}
```

1.2 Resultado da Execução

```
$ gcc ex1.c -l pthread -o ex1
$ ./ex1

Creating thread 0
Creating thread 1
Thread 1:295
Thread 0:5
Thread 1:290
Thread 1:285
Thread 0:10
...
Thread 1:5
Thread 0:150
Thread 1:0
Thread 0:155
...
Thread 0:285
Thread 0:290
Thread 0:295
Thread 0:300
Creating thread 0
Creating thread 1
Thread 0:5
Thread 0:10
...
Thread 0:560
Thread 0:555
Thread 1:550
Thread 1:555
...
Thread 1:445
Thread 0:560
Thread 1:440
Thread 0:445
...
Thread 0:440
Thread 1:435
Thread 0:440
Thread 0:440
...
Thread 0:660
Thread 1:435
```

```
Thread 1:660
...
Thread 1:565
Thread 0:665
Thread 0:565
...
Thread 1:-5
Thread 1:-10
Thread 1:-15
Thread 1:-20
Thread 1:-25
Variavel global: -25
```

1.3 Perguntas

- **Houve concorrência entre as threads em a. e b.? Justifique sua resposta. Os valores impressos foram os esperados? Justifique sua resposta.**

  Sim, houve concorrencia, nos dois itens as duas threads executaram as operações simultaneamente. No item a, uma thread não interveriu no resultado da outra, pois usamos variáveis locais. Isso pode ser comprovado pelo fato de que a thread 0 terminou com a contagem 300 (fizemos um loop de 1 a 60 e somamos 5 a cada iteração) e que a thread 1 terminou com a contagem 0 (inicializamos a contagem com 300 e fizemos um loop de 1 a 60, subtraindo 5 a cada iteração).

  Entretanto, no item b, com a variável global, as duas threads usam a mesma variável para contabilizar a soma ou a subtração. Como uma thread pode ser interrompida pela outra, por exemplo, enquanto lê a variável, o valor a ser somado ou subtraido pode ser perdido nessas interrupções. Sendo assim, o valor final da variável global pode não ser zero.

  Para verificar isso, fizemos um loop de 1 a 2000, na função incrementa somamos 5 a cada iteração e na função decrementa subtraimos 5 em cada iteração. O resultado que obtivemos foi de -25 e não de 0, o que mostra que houve concorrencia e também houve algum valor que foi perdido em alguma interrupção.

  No resultado acima, é possível reparar alguns pontos onde há "confusão"e a thread que deveria subtrair acaba somando, ou a thread que está recomeçando não usa o valor atualizado da variável global e sim o valor que era antes de ser interrompida.

2 Problema 2

Implemente um programa que, dado um vetor de 4K posições inicializado com valores inteiros, crie 4 processos para paralelizar a busca do maior valor armazenado no vetor. O processo coordenador vai criar os 4 processos trabalhadores e consolidar a resposta indicando o maior valor armazenado no vetor. Contabilize o tempo da execução do processo. Faça o mesmo programa utilizando tarefas e compare os tempos de execução dos dois programas. Explique os resultados apresentados.

2.1 Código Fonte

```
/*
Tharik Lourenço
Suemy Inagaki
SO - Lab3 - Entrega Final
processos.c
*/

#include <stdio.h>
#include <stdlib.h>
#include <sys/shm.h>
#include <sys/stat.h>
#include <unistd.h>
#include <sys/wait.h>
#include <time.h>

#define TAM_TOTAL 4000
#define NUMPROC 4

int main( void ){
    clock_t begin = clock();
    int segmento,*p;

    segmento=shmget(IPC_PRIVATE, (TAM_TOTAL*sizeof(int))
    ,IPC_CREAT | IPC_EXCL | S_IRUSR | S_IWUSR);

    if(segmento<0){
        printf("Shmget erro!\n");
        exit(1);
    }
```

```
//criando o vetor que vai armazenar max na memória compartilhada
int* max = (int *)shmat(segmento, 0, 0);
for(int t = 0; t < NUMPROC; t++){
    max[t] = -1;
}

//criando o vetor que vai armazenar os valores na memoria compartilhada
p = (int*)shmat(segmento,0,0);
for(int t=0;t<TAM_TOTAL;t++){
    if (t == 500)
        p[t] = 5000;
    else
        p[t]= rand()%1000;
}


for (int i = 0; i < NUMPROC; ++i)
{
    if (fork() == 0){
        printf("começo do processo filho %d\n",getpid());

        /*separa igualmente a qtd de elementos para cada processo*/
        int start = 0;
        for(int j=0;j<i;++j){
            start += (TAM_TOTAL+1-start) /(NUMPROC - j);
        }
        int tam = (TAM_TOTAL+1-start) /(NUMPROC - i);
        int end = start + tam;

        for(int n = start ;n < end;n++){
            sleep(0.1);
            printf("processo %d: p[%d]=%4d\n" , getpid(),n,p[n]);
            if (p[n] > max[i]){
                max[i] = p[n];
            }
        }

        exit(EXIT_SUCCESS);
    }
}
```

```
    for (int i = 0; i < NUMPROC; ++i)
        wait(NULL);
    printf("processo pai vai terminar !!!\n");
    printf("Maiores valores em cada segmento:\n");
    printf("segmento1: max = %d\n", max[0]);
    printf("segmento2: max = %d\n", max[1]);
    printf("segmento3: max = %d\n", max[2]);
    printf("segmento4: max = %d\n", max[3]);
    int m = -1;
    for(int i = 0; i < NUMPROC; i++)
        if(max[i] > m)
            m = max[i];
    printf("Valor maximo do vetor: %d\n",m);
    clock_t end = clock();
    double time_spent = (double)(end - begin) / CLOCKS_PER_SEC;
    printf("Tempo gasto na execucao: %.4fms\n", time_spent*1000);
    exit(0);
}


/*
Tharik Lourenço
Suemy Inagaki
Lab 03 Threads, questao 2
Entrega Final
Sistemas Operacionais
*/


#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/shm.h>
#include <sys/stat.h>
#include <unistd.h>
#include <sys/wait.h>
#include <time.h>

#define NUM_THREADS 4
#define TAM_TOTAL 4000
```

```c
int count = 0;
int max = -1;

int p[TAM_TOTAL];

void* percorre(void* threadid){
        int ini, fim;
    if(threadid == 0){
        ini = 0;
        fim = 1000;
        }
    else if(threadid == 1){
        ini = 1000;
        fim = 2000;
    }
    else if(threadid == 2){
        ini = 2000;
        fim = 3000;
    }
    else{
        ini = 3000;
        fim = 4000;
    }
    for(int i=ini;i<fim;i++){
        sleep(0.1);
        if(p[i] > max)
            max = p[i];
        printf("Processo %d:p[%d]=%3d\n", threadid, i,p[i]);
    }
    pthread_exit(NULL);
}


int main()
{
    clock_t begin = clock();
    pthread_t threads[NUM_THREADS];
    int t;
    /*
     Preenche vetor com numeros entre 0 e 999
     Coloca o valor 4000 na posicao 200
```

```
     O programa deve retornar max = 4000
    */

    for(int t=0;t<TAM_TOTAL;t++){
        p[t]= rand()%1000;
        if(t == 200)
            p[t] = 4000;
    }

    printf("Creating thread 0\n");
    pthread_create(&threads[0], NULL, percorre, (void *)0);
    printf("Creating thread 1\n");
    pthread_create(&threads[1], NULL, percorre, (void *)1);
    printf("Creating thread 2\n");
    pthread_create(&threads[2], NULL, percorre, (void *)2);
    printf("Creating thread 3\n");
    pthread_create(&threads[3], NULL, percorre, (void *)3);

    for(t=0; t < NUM_THREADS; t++)
        pthread_join(threads[t],NULL);

    printf("max = %d\n", max);

    clock_t end = clock();
    double time_spent = (double)(end - begin) / CLOCKS_PER_SEC;
    printf("Tempo gasto na execucao: %.4fms\n", time_spent*1000);
}
```

2.2 Resultado da Execução

```
$gcc processos.c -o processos
$./processos

come o do processo filho 100
processo 100: p[3000]=  72
processo 100: p[3001]= 327
...
processo 100: p[3149]= 257
processo 100: p[come o do processo filho 99
processo 99: p[2000]= 833
...
processo 98: p[1998]= 161
processo 98: p[1999]= 134
processo pai vai terminar !!!
Maiores valores em cada segmento:
segmento1: max = 5000
```

```
segmento2: max = 997
segmento3: max = 999
segmento4: max = 999
Valor maximo do vetor: 5000
Tempo gasto na execucao: 1.1520ms

$gcc threads.c -o threads -l pthread
$./threads

Creating thread 0
Creating thread 1
Creating thread 2
Processo 0:p[0]=383
Creating thread 3
Processo 1:p[1000]=693
Processo 0:p[1]=886
Processo 3:p[3000]=327
...
Processo 1:p[1998]=134
Processo 1:p[1999]=833
max = 4000
Tempo gasto na execucao: 91.3230ms
```

2.3 Comentários

Para a execução da busca em paralelo usando processos, foi necessário criar um vetor na memória compartilhada para armazenar os valores máximos de cada trecho do vetor.Já para a execução da busca em paralelo usando threads, bastou utilizarmos uma variável global. Nos dois casos houve concorrencia, os processos foram executados paralelamente.

Com relação ao tempo de execução, a paralelização com processos executou a tarefa em um tempo bem menor que a paralelização com threads. Fizemos alguns testes alterando o clock de lugar e o resultado foi o mesmo.

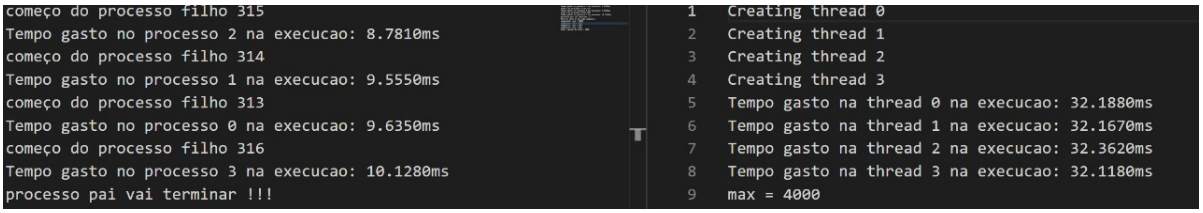

Figura 1: tempo gasto em cada processo X tempo gasto em cada thread