
 INSTITUTO DE INFORMÁTICA UFG	<b>Universidade Federal de Goiás</b> <b>Instituto de Informática</b> <b>Sistemas Operacionais</b>	 <b>UFG</b>
<b>Professor:</b> Diego Américo Guedes		<b>Data:</b> 05/11/2020
<b>Lista de Exercícios - Threads</b> <b><u>Vale nota (64h e 96h) e presença (96h)</u></b>		

1. A seguir são apresentadas três versões de um mesmo programa: uma serial, uma “paralela” com múltiplos processos e uma “paralela” com múltiplas *threads*. O programa possui duas funções, uma que usa a CPU de forma intensiva (*CPU bound*) e outra que simula o uso intensivo de operações de E/S (*I/O bound*). Compile e execute cada uma das versões conforme as orientações que seguem. Anote e compare o desempenho das três versões. Explique as diferenças observadas.

```

/* prog1.c - serial */
#include <stdio.h>
#include <unistd.h>
#include <math.h>
#include <string.h>
#define CALLS 10

void CPU_bound (int id)
{
    int i;
    double result=0.0;

    /* Consumo de CPU */
    for (i=0; i<10000; i++)
        result = result + sin(i) * tan(i) * sqrt(result);

    printf("%d: CPU intensiva terminada\n", id);
}

void IO_bound (int id)
{
    /* Simula operações de I/O, as quais levam a bloqueio */
    sleep(1);

    printf("%d: E/S intensiva terminada\n", id);
}

int main()
{
    int i;

    for(i=0; i<CALLS; i++)
    {
        if(i%2)
            CPU_bound(i);
        else
            IO_bound(i);
    }

    printf("\n*** Tarefas concluídas ***\n");
    return 0;
}

```

Compilação do prog1.c:

```
gcc -Wall -lm prog1.c -o prog1
```

Execução do prog1:

```
time ./prog1
```

```

/* prog2.c - múltiplos processos */
#include <stdio.h>
#include <unistd.h>
#include <math.h>
#include <string.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/wait.h>
#define NPROCESS      10

void CPU_bound (int id)
{
    int i;
    double result=0.0;

    /* Consumo de CPU */
    for (i=0; i<10000; i++)
        result = result + sin(i) * tan(i) * sqrt(result);

    printf("%d: CPU intensiva terminada\n", id);
}

void IO_bound (int id)
{
    /* Simula operações de I/O, as quais levam a bloqueio */
    sleep(1);

    printf("%d: E/S intensiva terminada\n", id);
}

int main()
{
    int i, status;
    pid_t children[NPROCESS];

    for(i=0; i<NPROCESS; i++)
    {
        if(i%2)
        {
            children[i]=fork();
            if(children[i]==0)
            {
                CPU_bound(i);
                exit(0);
            }
        }
        else
        {
            children[i]=fork();
            if(children[i]==0)
            {
                IO_bound(i);
                exit(0);
            }
        }
    }
    for(i=0; i<NPROCESS; i++)
        waitpid(children[i], &status, 0);

    printf("\n*** Tarefas concluídas ***\n");
    return 0;
}

```

Compilação do prog2.c:

```
gcc -Wall -lm prog2.c -o prog2
```

Execução do prog2:

```
time ./prog2
```

```

/* prog3.c - múltiplas threads */
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <math.h>
#include <string.h>
#define NTHREADS      10

void *CPU_bound(void *threadid)
{
    int i;
    double result=0.0;

    /* Consumo de CPU */
    for (i=0; i<10000; i++)
        result = result + sin(i) * tan(i) * sqrt(result);

    printf("%ld: CPU intensiva terminada\n", (long)threadid);
    pthread_exit(NULL);
}

void *IO_bound(void *threadid)
{
    /* Simula operações de I/O, as quais levam a bloqueio */
    sleep(1);

    printf("%ld: E/S intensiva terminada\n", (long)threadid);
    pthread_exit(NULL);
}

int main()
{
    pthread_t threads[NTHREADS];
    int rc;
    long thId;
    void *status;

    for(thId=0; thId<NTHREADS; thId++)
    {
        if(thId%2)
            rc = pthread_create(&threads[thId], NULL, CPU_bound, (void *)thId);
        else
            rc = pthread_create(&threads[thId], NULL, IO_bound, (void *)thId);

        if (rc)
        {
            printf("ERRO: código de retorno de pthread_create() é %d\n", rc);
            exit(-1);
        }
    }

    for(thId=0; thId<NTHREADS; thId++)
    {
        rc = pthread_join(threads[thId], &status);
        if (rc)
        {
            printf("ERRO: código de retorno de pthread_join() é %d\n", rc);
            exit(-1);
        }
    }

    printf("\n*** Tarefas concluídas ***\n");
    pthread_exit(NULL);
}

```

Compilação do prog3.c:

```
gcc -Wall -lm -pthread prog3.c -o prog3
```

Execução do prog3:

```
time ./prog3
```

2. Implemente um programa que crie uma cadeia de  $N$  *threads* (além da *thread* principal). A cadeia deve obedecer a seguinte sequência: 1ª *thread* cria o 2ª *thread*, 2ª *thread* cria o 3ª *thread*, ..., (N-1)ª *thread* cria a Nª *thread*. Cada *thread* deve imprimir seu ID e o ID da *thread* que a criou. Garanta que a informação exibida na tela ocorrerá na ordem inversa da criação das *threads*, ou seja, inicialmente aparecem as informações da Nª *thread*, depois da (N-1)ª, ..., depois da 2ª e por fim da 1ª.

3. Implemente um programa que crie três *threads*, ou seja, três funções. Uma função deve escrever na tela "AAAAA", a segunda "BBBBB" e a terceira "CCCCC". Execute as três *threads*, garantindo que seja sempre escrito na tela "AAAAABBBBBCCCCC", nessa ordem.

4. A seguir é apresentada versão serial de um programa para cálculo de todos os números primos até um determinado valor. Implemente uma versão *multithread* para esse programa. Avalie o desempenho do seu programa dois valores de entrada: 10.000.000 e 100.000.000. Ao fazer a comparação, não imprima o resultado na tela, pois a concorrência para realização de E/S tornará o programa muito mais lento e afetará a avaliação.

```
/* primos.c - identifica todos os numeros primos ate um certo valor*/
#include<stdio.h>
#include <math.h>
#include <stdlib.h>

int verifica_se_primo(long int);

int main(int argc, char *argv[])
{
    long int numero = 0;
    short int result, imprimir;

    if (argc != 3) {
        printf ("Uso correto: %s <numero> <imprimir=1,nao_imprimir=0>\n\n", argv[0]);
        return 1;
    }

    numero = atol(argv[1]);
    imprimir = atoi(argv[2]);

    for (long int num_int = 1; num_int <= numero; num_int++) {
        result = verifica_se_primo(num_int);

        if (imprimir == 1)
            if (result == 1)
                printf("%ld eh primo.\n", num_int);
    }

    return 0;
}

int verifica_se_primo(long int numero)
{
    long int ant;

    for (ant = 2; ant <= (long int)sqrt(numero); ant++) {
        if (numero%ant == 0)
            return 0;
    }
    if (ant*ant >= numero)
        return 1;
}
```

5. Implemente um programa que crie duas *threads* adicionais, ou seja, chama duas funções diferentes. A primeira grava em um arquivo números de 1 a 10. A segunda grava em um arquivo letras de A a Z. A *thread* principal (que criou as demais), após cada *thread* terminar sua execução, deve listar o conteúdo dos arquivos criados na tela.