A IMPORTÂNCIA DE THREADS NO DESEMPENHO DE APLICAÇÕES

Euzébio da Costa Silva[1], Victor Pereira Ribeiro[2], Susana Brunoro Costa de Oliveira[3]

[1]Instituto Federal do Espírito Santo - Campus de Alegre, Rod Br 482, Km 47, s/n - Rive, Alegre - ES, 29520-000, euzebioprogramacao@gmail.com
[2]Instituto Federal do Espírito Santo - Campus de Alegre, Rod Br 482, Km 47, s/n - Rive, Alegre - ES, 29520-000, victor3ifes@gmail.com
[3]Instituto Federal do Espírito Santo - Campus de Alegre, Rod Br 482, Km 47, s/n - Rive, Alegre - ES, 29520-000/Departamento de Informática, sbrunoro@ifes.edu.br

Resumo - Executar mais de um programa no computador é uma prática que tem se tornado mais comum a cada dia, mas como isso é possível? O que esta por trás disso? Com a finalidade de mostrar a importância e a presença de *threads* para os usuários de computadores, e assim despertar o interesse, principalmente de estudantes das áreas relacionadas, este trabalho realiza, nos seus testes, a implementação de códigos em C++ usando bibliotecas da própria linguagem, e também algumas comuns da C, para demonstrar como são eficientes e apresentar resultados do beneficio deste método de execução.

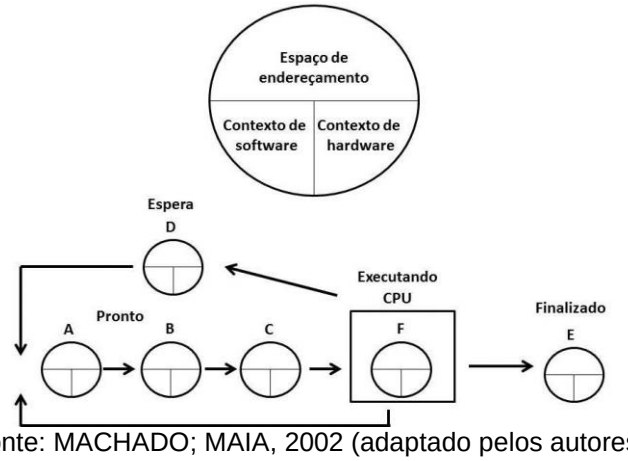
Palavras-chaves: Técnicas de desempenho, Gerenciamento de processos, Implementação de códigos.
Área do conhecimento: Sistemas Operacionais.

Introdução

No bloco onde o processo é armazenado na memória, encontra-se o seu contexto de hardware, que é o local onde se encontram as informações necessárias para o processamento, tais como os valores dos conteúdos dos registradores do processador e resultado de operações.

O contexto de software é onde são armazenadas as informações referentes ao processamento desses processos, tais como: espaço de memória necessário, números de subprocessos e tempo de uso do processador. Por fim o espaço de endereçamento que é o espaço onde estão as instruções dos processos e variáveis usadas por ele. Como é mostrado na figura 1, logo após ser criado o processo entra na fila de pronto onde ele espera para pela sua vez de usar o processador, após o termino do tempo de execução, ele pode ir para espera, como o nome diz o processo aguardo uma operação (E/S), pode ir para o fim da fila, esperar pela sua vez novamente, pode também ir para o estado de finalizado, e encerrar suas tarefas (MACHADO; MAIA, 2002).

Figura 1- Ciclo de vida do processo.



Fonte: MACHADO; MAIA, 2002 (adaptado pelos autores).

Um thread, como descrito por Silberschatz *at al* (2008) é uma unidade básica de utilização da CPU, mas podem ser entendidas como parte da execução de um programa. Elas têm o mesmo ciclo de vida que um processo, porém compartilham o mesmo espaço de endereçamento do seu processo "pai" isso causa um ganho de desempenho, comparado a criação de novos processos, além disso, *threads* podem ser executados em paralelo, no caso de múltiplos núcleos e ou múltiplos processadores. (MACHADO; MAIA, 2002).

Dessa forma, o presente trabalho objetiva mostrar a importância de *threads* no desempenho de aplicações e despertar o interesse, principalmente de alunos das áreas relacionadas, no aprofundamento desse tema.

Metodologia

Para implementação dos *threads,* foram utilizados os algoritmos de ordenação *Bubble Sort, Insertion Sort* e o *Selection Sort*. O *Bubble Sort* segundo Laureano (2012) é um método simples de ordenação por troca inicia percorrendo a lista da esquerda para a direita, comparando pares de elementos consecutivos, trocando de lugar os que estão fora de ordem. O *Selection Sort* de acordo com Laureano (2012) consiste em trocar o menor elemento (ou maior) de uma lista com o elemento posicionado no início da lista, depois o segundo menor elemento para a segunda posição e assim sucessivamente . No *Insertion Sort* segundo Laureano (2012) é feito a ordenação os dois primeiros membros da lista, em seguida o algoritmo insere o terceiro membro na sua posição ordenada com relação aos dois primeiros membros. Para cada um desses algoritmos foram feitos dois testes com vetores de 100000 (cem mil) valores, sendo que o primeiro teste foi feito com 5 (cinco) e o segundo com 9 (nove) vetores. Cada um deles foi implementado com e sem a utilização do *thread* em modo usuário. Os algoritmos foram escolhidos para representar aplicações com tempos variados de execução. Deste modo pode se ter resultados mais variados entre os códigos.

Para implementação dos algoritmos, foi usada a linguagem C++ e as seguintes bibliotecas da linguagem C: *stdlib.h* que contém a função de *srand*() e *rand*(), para gerar números aleatórios que ocuparam as posições do vetor, *time.h,* para calcular aproximadamente o tempo necessário para criação e ordenação do vetor e *pthread.h,* para criar *threads* em modo usuário.

Figura 2- Código do programa que implementa a ordenação de 9 vetores com 100.000 posições cada usando o algoritmo de ordenação *Bubble Sort* implementado *thread*.

```
# include <iostream>
# include <stdlib.h>
# include <time.h>
# include <pthread.h>

using namespace std;
void *Bubble(void *arg){
        const int tamanho = 100000;
        int vetor[tamanho];
        int cont, auxilio;
        bool troca;
        srand(1);
        for(cont = 0; cont <= tamanho; cont ++){
                vetor[cont] = (rand() % tamanho); }
        // Bubblesort:
        do{
                troca = false;
                for (cont = 0; cont <= tamanho; cont++){
                        if(vetor[cont] > vetor[cont + 1]){
                                auxilio = vetor[cont + 1];
                                vetor[cont + 1] = vetor[cont];
                                vetor[cont] = auxilio;
                                troca = true;}
                }
        }while(troca == true);
}
int main (){
        int  Tinicial, Tfinal,  numero = 9;
        pthread_t  th[numero];
        int arg, r;

        Tinicial = time(NULL);
        for (r = 0; r < numero; r++){
                pthread_create(&th[r], NULL, *Bubble,(void *) arg);
        }
        for (r = 0; r < numero; r++){
                pthread_join(th[r],NULL);
        }
        Tfinal = time(NULL);
        cout << "Tempo gasto: " << difftime(Tfinal, Tinicial);
}
```

Fonte: Elaborado pelos autores.

Figura 3- Código do programa que implementa a ordenação de 9 vetores com 100.000 posições cada usando o algoritmo de ordenação *Insertion Sort* implementado *thread*.

```
# include <iostream>
# include <stdlib.h>
# include <time.h>
# include <pthread.h>

using namespace std;

void *Insertion(void *args){
        int tamanho = 100000;
        int vetor[tamanho];
        int posicao = 1, auxilio;

        for(int cont = 0; cont <= tamanho; cont ++){
                vetor[cont] = rand() % tamanho;
        }

        // Insertion sort:
        do{
                for(int t = posicao; t > 0; t--){
                        if (vetor[t - 1] > vetor[t]){
                                auxilio = vetor[t - 1];
                                vetor[t - 1] = vetor[t];
                                vetor[t] = auxilio;
                        }
                }
                posicao++;
        }while(posicao <= tamanho);
        // fim do Insertio sort.
}

int main (){
        int  Tinicial, Tfinal,  numero = 9;
        pthread_t  th[numero];
        int arg, r;

        Tinicial = time(NULL);
        for (r = 0; r < numero; r++){
                pthread_create(&th[r], NULL, *Insertion,(void *) arg);
        }
        for (r = 0; r < numero; r++){
                pthread_join(th[r],NULL);
        }
        Tfinal = time(NULL);
        cout << "Tempo gasto: " << difftime(Tfinal, Tinicial);
        return 0;
}
```

Fonte: Elaborado pelos autores.

Figura 4- Código do programa que implementa a ordenação de 9 vetores com 100.000 posições cada usando o algoritmo de ordenação *Selection Sort* implementado *thread*.

```
# include <iostream>
# include <stdlib.h>
# include <time.h>
# include <pthread.h>

using namespace std;

void *Selection(void *arg){
        int vetor[100000];
        int cont, selecao, auxilio;
        srand(1);

        for(cont = 0; cont <= 100000; cont ++){
                vetor[cont] = rand() % 100000;
        }
        // Selectio sort:
        for(cont = 0; cont < 100000; cont++){
                for(selecao = cont + 1; selecao <= 100000; selecao++){
                        if (vetor[cont] > vetor[selecao]){
                                auxilio = vetor[cont];
                                vetor[cont] = vetor[selecao];
                                vetor[selecao] = auxilio;
                        }
                }
        }
        // Fim do Selectio sort.
}

int main (){
        int  Tinicial, Tfinal,  numero = 9;
        pthread_t  th[numero];
        int arg, r;

        Tinicial = time(NULL);
        for (r = 0; r < numero; r++){
                pthread_create(&th[r], NULL, *Selection,(void *) arg);
        }
        for (r = 0; r < numero; r++){
                pthread_join(th[r],NULL);
        }
        Tfinal = time(NULL);
        cout << "Tempo gasto: " << difftime(Tfinal, Tinicial);
        return 0;
}
```

Fonte: Elaborado pelos autores.

Os testes foram realizados em 2 (dois) computadores. O primeiro foi o notebook Samsung RV415 com o sistema operacional Windows 7 Ultimate 64 bits, processador AMD E-300 com 2 CPUs de 1.3GHz e com 4096MB de RAM e o segundo foi o computador Desktop Dell com sistema operacional Windows 7 Professional 64 bits, processador Intel® Core ™ 2 Quad com 4 CPUs de 2.3GHz e com 4096MB de RAM. Assim, é possível avaliar o desempenho da utilização de *threads* em diferentes sistemas computacionais.

Resultados

Em cada computador, foram realizados os testes utilizando 5 e 9 vetores para cada um dos algoritmos de ordenação, de duas formas: com e sem *thread*, totalizando 24 execuções. O tempo de execução foi apresentado ao final de cada operação. A tabela 1 apresenta os resultados obtidos dos dois computadores. No computador 1, com 2 (dois) núcleos, o ganho médio do processamento com *thread* em relação ao processamento sem *thread* foi de 48,34%, enquanto no segundo computador, com 4 (quatro) núcleos, o ganho de desempenho médio atingiu 67,46%.

Tabela 1- Tempo gasto por algoritmo para ordenar os vetores de 5 e 9 elementos nos computadores 1 e 2.

Computador	Algoritmo	Nº de vetores	Sem thread	Com thread	Ganho
1	**Bubble**	5	978	550	43,76%
		9	2202	1193	45,82%

	Insertion	5	333	161	51,65%
		9	740	363	50,95%
	Selection	5	299	158	47,16%
		9	671	331	50,67%
Ganho Médio					**48,34%**
2	**Bubble**	5	187	77	58,82%
		9	420	124	70,48%
	Insertion	5	143	44	69,23%
		9	257	73	71,60%
	Selection	5	132	49	62,88%
		9	237	67	71,73%
Ganho Médio					**67,46%**

Fonte: Elaborado pelos autores

Discussão

Ao analisar as tabelas, percebe-se que todos os processos com *threads* tiveram ganho expressivo de desempenho no computador 1 (um) e ainda maior no computador 2 (dois). Isso se deve ao paralelismo de processamento proporcionado pela utilização de mais de um *thread*, mas, é importante notar que comparando os resultados do computador 1 (um) com o 2 (dois) houve uma grande diferença de desempenho, isso se dá devido às tecnologias dos processadores. O processador do primeiro computador possui 2 (dois) núcleos e do segundo, 4 (quatro). Isso indica que o processador do segundo computador pode executar até 4 (quatro) *threads* em paralelo, aumentando a concorrência, ou seja, ele executa duas vezes mais *threads* comparado com o primeiro que pode executar até 2 (dois) *threads* simultaneamente. É importante ressaltar que os processadores AMD funcionam de modo diferente que os da Intel, embora a base seja a mesma, além disso, as outras especificações técnicas dos computadores também podem influenciar nos resultados, um exemplo é a velocidade do processador do segundo computador, que sendo mais rápido que a do primeiro, executou os programas sem *threads* em menor tempo que o primeiro computador usando *threads.*

Conclusão

Threads são fluxos de execução de um processador. Várias *threads* de um mesmo processo podem ser executados em paralelo. No presente trabalho, foi feita a análise comparativa de desempenho do processamento de três algoritmos de ordenação de vetor com e sem a utilização de *threads* em dois ambientes computacionais distintos. Em todos os casos, houve ganho de desempenho no processamento que implementa *thread* em relação ao processamento que não implementa. Os resultados obtidos, apresentam ganho de processamento entre 43,76% e 71,73%, dessa forma, observa-se a vantagem da utilização dessa técnica quando há a possibilidade de se paralelizar as rotinas de um processo.

Referências

LAUREANO, M., Ordenação. In LAUREANO, M.; **Estrutura de Dados com Algoritmos e C.** 1.ed.Curitiba: Brasport, 2012, p.100-107. Disponível em: http://www.mlaureano.org/livro/livro_estrutura_conta.pdf

MACHADO, F.B; MAIA, L.P.,Thread. In: MACHADO, F.B; MAIA, L.P.; **Arquitetura de Sistemas Operacionais.** 3.ed. Rio de Janeiro: LTC, 2002, p.83-94.

SILBERSCHATZ, A.; GAGNE, G.; GALVIN, P.B.,Treads. In: SILBERSCHATZ, A.; GAGNE, G.; GALVIN, P.B.; **Sistemas Operacionais com Java**. 7.ed.Rio de Janeiro: Elsevier, 2008, p.97 - 100.

TANENBAUM, A.S., Threads. In: TANENBAUM, A.S.; **Sistemas Operacionais Modernos.** 3.ed.São Paulo: Pearson, 2012, p.57-67.