Trabalho 2 Fundamentos de Sistemas Operacionais - Questão 2

Alunos:
Eduardo Q. Gomes 14/0137068
Miguel Pimentel 14/0156143

Material Usado

- Sistema Operacional Linux, distribuição Ubuntu 16.04 com a interface Gnome;
- Compilador gcc (5.4.0);
- Editor de texto Sublime.
- Valgrind versão 3.11.0

Conteúdo Utilizado

- Conteúdo do Moodle disponível em http://aprender.ead.unb.br/course/view.php?id=3806
- Livro Autor: Autor: SILBERSCHATZ, A.; GAGNE, G.; GALVIN, P.B. Obra: Operating System Concepts

Questão 2

Calculando o máximo de uma sequência de forma diferente (e rápida).

- **O cálculo do máximo de uma sequência de inteiros pode ser realizado com o seguinte fragmento de código:**

```
max = x[0];
for (i = 1; i < n; i++)
if (x[i] > max)
max = x[i];
```

- **Isso demandará à única CPU a execução de (n-1) comparações para o cálculo do máximo. Porém, será possível fazer melhor se dispormos de mais CPUs? Vejamos uma alternativa.**

- **Considere uma thread como a representação lógica de uma CPU. Se possuímos uma sequência de n inteiros distintos x[0], x[1], …, x[n-1], começamos pela inicialização de um array de trabalho w com n posições todas com o mesmo conteúdo, por exemplo, 1. Isso pode ser obtido com o disparo de n theads, cada uma com a responsabilidade de escrever o valor 1 em sua respectiva posição do array w. Por exemplo, seja Ti a thread que escreverá 1 na posição w[i]. Como cada thread (ou CPU) executa essa ação em um passo e como todas as threads são concorrentes (executam simultaneamente), a etapa de inicialização demadará apenas um passo na linha de tempo de execução! Depois dessa etapa, mais dois passos ainda são necessários.**

- **A Etapa 2 é contemplada da forma apresentada a seguir. Para cada par de inteiros x[i] e x[j], criamos uma thread (ou alocamos uma CPU, se você assim**

preferir) Tij. Essa thread compara x[i] e x[j] e escreve 0 em w[i] se x[i]<x[j] ou, no caso contrário, escreve 0 em w[j]. Dessa maneira, nessa etapa, necessitamos de um n(n-1)/2, em que cada thread executa uma operação de comparação e escrita. QUESTÃO: Por que precisamos n(n-1)/2 em vez de n^2 threads?

Respostas ao final do enunciado.

- **A Etapa 3 demandará n threads. A i-ésima thread deve checar o valor de w[i]. Se o valor for 0, a thread interrompe. Se o valor for 1, a thread mostra o valor de x[i], porque esse é o valor máximo! A definição de máximo é um valor que é mior do que qualquer um dos outros números. Basicamente, quando x[i] foi comparado com todos os outros valores na Etapa 2, a sua entrada em w[i] não foi zerada! Naturalmente, a Etapa 3 requer um passo de impressão do valor máximo. Dessa forma, se possuímos uma sequência de n inteiros, necessitamos apenas de 3 passos para encontrar o máximo!**

Exemplo:

- **Considere a seguinte sequência de inteiros: x[] = {3, 1, 7, 4};. O array w[4] deve ser inicializado com {1, 1, 1, 1}.**
- **Na Etapa 2, necessitamos de 4*3/2=6 threads (ou CPUs). Confira a tabela:**
- **Depois dessa Etapa, w[0], w[1] e w[3] são iguais a 0, e w[2] ainda possui seu valor de inicialização, 1. Dessa forma, na Etapa 3, a thread associada com w[2] encontra o valor 1 na célula e exibe o valor de x[2], que é 7: o máximo da sequência.**

Questão de implementação.

- **Entrada e saída. O programa a ser implementado deverá receber entradas por meio de parâmetros de linha de comando. Considerando que o seu executável tem nome q02, espera-se uma entrada da seguinte maneira:**
- **./q02 n x[0] x[1] x[2] … x[n-1]**
- **./nome_executável n_inteiros inteiro_0 inteiro_1 inteiro_2 … inteiro_n-1**
- **Considere esquema de programação defensiva e limite o seu programa para receber sequências de inteiro de até 100 elementos.**
- **Requisitos funcionais que o programa deverá atender:**
- **Imprimir o número de elementos da sequência de inteiros, a sequência de inteiros x e os valores do array w.**
- **A atividade realizada pela thread Tij na Etapa2.**
- **Os valores do array w após a Etapa 2.**
- **O valor do máximo da sequência e sua posição.**

Number of input values = 4
Input values x = 3 1 7 4
After initialization w = 1 1 1 1
.....
Thread T(1,3) compares x[1] = 1 and x[3] = 4, and writes 0 into

w[1]
.....
After Step 2
w = 0 0 1 0
Maximum = 7
Location = 2

- **Questão de Análise**

Implemente o mesmo programa, porém usando abordagem sequencial(convencional), ou seja, não use threads na sua implementação. Verifique se há diferenças de desempenho observadas na busca pelo máximo ao se comparar o comportamento da implementação com threads com a implementação sequencial. Caso haja diferenças, explique-as.

Respostas ao final do enunciado

- **Para essa solução, será necessário usar esquemas de bechmarking para comparar os tempos de execução.**

O programa foi desenvolvido na plataforma linux, desta forma, a utilização deste programa é apenas possíveis nesse Sistema Operacional (SO), pois é utilizada por exemplo a biblioteca pthread. O programa em questão foi criado a partir de um Makefile.

Desta forma para a execução do programa, realize as seguintes instruções:

1. Entre no diretório q02 onde estará disposta o código fonte e o Makefile para a execução do programa;
2. Abra um terminal(ctrl + alt + t para a maioria das distros linux), e digite:
    a. make - para compilar o programa;
    b. ./q02 <n_casos_de_teste> <caso_1> … <caso_n>.
    c. make clean - para deletar o executável do programa.

Dependências: Depende das bibliotecas <stdio.h> <stdlib.h> <pthread.h>.

Vale ressaltar que o programa tem algumas restrições, tais como:

- O número de casos de teste deve ser um número inteiro e menor ou igual a 100;
- Os casos de teste são número inteiros;

Como foi observado, as entradas ocorrem pela linha de comando por meio dos parâmetros argc e argv.

As Imagens a seguir apresentam o funcionamento do programa:

Etapa 1:

```
eduardo@Nautilus2:~/Documentos/GitE/SO_Trabalhos/T2/q02$ make run
./prog 4 3 1 7 4
Number of input values = 4

Input values x = 3 1 7 4

After initialization w = 1 1 1 1
```

Etapa 2:

```
Thread T(2,3) compares x[2] = 7 and x[3] = 4
Thread T(1,3) compares x[1] = 1 and x[3] = 4
Thread T(1,2) compares x[1] = 1 and x[2] = 7
Thread T(0,3) compares x[0] = 3 and x[3] = 4
Thread T(0,2) compares x[0] = 3 and x[2] = 7
Thread T(0,1) compares x[0] = 3 and x[1] = 1
```

Etapa 3:

```
After Step 2
w = 0 0 1 0
Maximum = 7
Location = 2
```

Visualização de um programa completo:

```
eduardo@Nautilus2:~/Documentos/GitE/SO_Trabalh
./prog 5 3 6 2 1 2
Number of input values = 5

Input values x = 3 6 2 1 2

After initialization w = 1 1 1 1 1

Thread T(0,1) compares x[0] = 3 and x[1] = 6
Thread T(0,2) compares x[0] = 3 and x[2] = 2
Thread T(0,3) compares x[0] = 3 and x[3] = 1
Thread T(0,4) compares x[0] = 3 and x[4] = 2
Thread T(1,2) compares x[1] = 6 and x[2] = 2
Thread T(1,3) compares x[1] = 6 and x[3] = 1
Thread T(2,3) compares x[2] = 2 and x[3] = 1
Thread T(1,4) compares x[1] = 6 and x[4] = 2
Thread T(3,4) compares x[3] = 1 and x[4] = 2
Thread T(2,4) compares x[2] = 2 and x[4] = 2

After Step 2
w = 0 1 0 0 0
Maximum = 6
Location = 1
```

Casos de teste:

Exemplo 1:

Entrada: ./q02 5 3 6 2 1
Saída: 0 0 0 0 1 0
.
.
.
After step 2
Maximun = 6
Location = 1

Coma a imagem acima representada

Por que precisamos n(n-1)/2 em vez de n^2 threads?

Precisamos de n(n-1)/2 pois na verdade esse é o numero de vezes seria suficiente para comparar os diferentes membros da sequencia de n, a utilização de n^2 threads na verdade não seria o ideal pois não é necessário comparar um número com ele mesmo ou compara-lo com com algum número que já tenha sido comparado numa sequencia dessa questão. Vejamos um exemplo:

Sequencia x = { 2, 3, 5, 3}
n1 = 4(4-1)/2 = 6
n2 = 4^2 = 16

Comparacoes com n1
2 com 3, 2 com 5, 2 com 3;
3 com 5, 3 com 3;
5 com 3.
Total = 6 comparações.

Comparações com n2
2 com 2, 2 com 3, 2 com 5, 2 com 3;
3 com 2, 3 com 3, 3 com 5, 3 com 3;
5 com 2, 5 com 3, 5 com 5, 5 com 3;
3 com 2, 3 com 3, 3 com 5, 3 com 3.
Total = 16 comparações.

O que você sugeriria para otimizar essa implementação?

Poderia ser uma sugestão implementar de forma sequencial essa solução, pois, por causa da simplicidade do programa ainda não compensa usar threads em detrimento do tempo da solução do problema em uma abordagem sequencial.

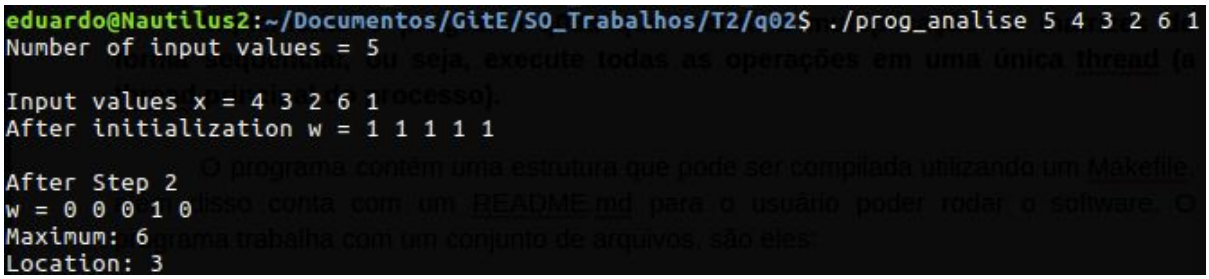
Resposta Questões de análise:

Arquivo da Questão de implementação: main_analise.c que é um arquivo independente para a realização da questão sem threads, ou seja, de forma sequencial.

Para executar o programa:
make
./q02_analise 5 4 3 2 6 1

Vale ressaltar que a resposta para a questão de análise está representada em um diretório (Implementacao_sequencial), o qual, encontra-se dentro do diretório q02.

Tela:



Analise dos tempos de execução com time ./prog e ./prog_analise:

Tempos de time ./prog 15 4 3 2 6 1 3 4 5 6 7 8 12 1 3 5
real 0m0.010s
user 0m0.000s
sys 0m0.004s

real 0m0.008s
user 0m0.000s
sys 0m0.004s

real 0m0.009s
user 0m0.008s
sys 0m0.000s

Tempos de time ./prog_analise 15 4 3 2 6 1 3 4 5 6 7 8 12 1 3 5
real 0m0.003s
user 0m0.000s
sys 0m0.000s

real 0m0.003s
user 0m0.000s

sys 0m0.000s

real 0m0.003s
user 0m0.000s
sys 0m0.000s

Tempos de times ./prog 15 4 3 2 6 1 3 4 5 6 7 8 12 1 3 5
0m0.640s 0m0.332s
0m2.236s 0m0.408s

Tempos de times ./prog_analise 15 4 3 2 6 1 3 4 5 6 7 8 12 1 3 5
0m0.628s 0m0.332s
0m2.236s 0m0.408s

Nas diferenças encontradas algumas observações podem ser feitas, dentre elas, o fato de com threads, é indicado por sys e user o aumento de tempo de execução com threads de usuário e do sistema, que já na questão sequencial não apresenta esse tempo, já que o programa é feito apenas com uma execução sequencial.

Também é possível observar que a implementação de forma sequencial foi visivelmente mais rápida que com threads, isso se deve a ausência de por exemplo a troca de contextos e a manipulação mais simples e rápida do sequencial por exemplo. Em programas com demandas maiores de processamento possivelmente essa execução com threads se tornaria mais rápida para resolver o problema do usuário.