Fundamentos de Sistemas Operacionais
Trabalho 02

Prof. Tiago Alves

Processos e Threads

Introdução

A disciplina de Fundamentos de Sistemas Operacionais trata de diversos tópicos desses sistemas que provêem uma forma intuitiva de se utilizar as funcionalidades de computadores digitais sem que seja necessário ao usuário ou programador ter profundo conhecimento das interações entre os diferentes *hardwares* que compõem um computador.

Para construir ou adicionar funcionalidades a esses sistemas computacionais, é necessário conhecimento de linguagens de programação e ferramentas de desenvolvimento. Em nosso curso, o domínio da linguagem C é um pré-requisito para o devido acompanhamento das atividades da disciplina.

Objetivos

1) Exercitar conceitos da linguagem de programação C, especialmente aqueles referentes à programação de sistemas operacionais.
2) Exercitar aspectos de programação de sistemas operacionais referentes ao gerenciamento de threads e processos.

Referências Teóricas

Capítulos 3, 4 e 5 de Mitchell, Mark, Jeffrey Oldham, and Alex Samuel. Advanced linux programming. New Riders, 2001.

Material Necessário

- Computador com sistema operacional programável
- Ferramentas de desenvolvimento GNU/Linux ou similares: compilador GCC, depurador, editor de texto.

Roteiro

1) Revisão de técnicas e ferramentas de desenvolvimento usando linguagem C.

Colete o material acompanhante do roteiro do trabalho a partir do Moodle da disciplina e estude os princípios e técnicas de desenvolvimento de aplicações usando linguagem C e sistema operacional Linux.

2) Realizar as implementações solicitadas no questionário do trabalho.

Nas referências bibliográficas que acompanham o trabalho, há uma breve apresentação de ferramentas de depuração para o ambiente Linux.

ATENÇÃO: Suas implementações deverão ser construídas a partir de um Makefile.

Implementações e Questões para Estudo

1) Escreva um programa em C que implemente uma aplicação que simula o alarme de relógio. A aplicação alarme.c deverá disparar um processo filho que, logo após sua execução, dorme por 5 segundos antes de enviar um SIGALRM ao processo pai. O processo pai, por sua vez, prepara-se para receber o sinal do tipo SIGALRM, aguarda por esse sinal; quando o sinal é recebido, imprime na tela a ocorrência do evento e, em seguida, se encerra. Faça o uso da função pause(void) para aguardar pelo sinal enviado pelo processo filho.
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?**

- 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:

Thread	Comparação	Saída
T01	Compara x0=3 e x1=1	Grava 0 em w1
T02	Compara x0=3 e x2=7	Grava 0 em w0
T03	Compara x0=3 e x3=4	Grava 0 em w0
T12	Compara x1=1 e x2=7	Grava 0 em w1
T13	Compara x1=1 e x3=4	Grava 0 em w1
T23	Compara x2=7 e x3=4	Grava 0 em w3

  - 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
```

- **Questões 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.
    - Para essa solução, será necessário usar esquemas de bechmarking para comparar os tempos de execução.
  - O que você sugeriria para otimizar essa implementação?

3) Escreva programas em C que implementem a multiplicação de matrizes de coeficientes inteiros de acordo com os requisitos abaixo:
   - Implemente o programa q03a que realize a multiplicação de matrizes de forma sequencial, ou seja, execute todas as operações em uma única thread (a thread principal do processo).
   - Implemente o programa q03b que realize a multiplicação de matrizes de forma concorrente, lançando uma thread para o cálculo simultâneo de cada célula da matriz de saída.
     - Considere o produto de duas matrizes inteiras do seguinte suporte: 3x2 e 2x3. A matriz de saída deverá ter dimensões 3x3; logo, seu programa deverá calcular as 9 células da matriz de saída disparando 9 threads simultaneamente, cada uma responsável pelo cálculo da sua respectiva célula de saída.
   - Implemente o programa q03c que realize a multiplicação de matrizes de forma concorrente, porém, lançando uma thread para o cálculo de cada célula da matriz de saída.
     - O número máximo de threads disparadas simultaneamente deverá ser igual ao inteiro que resulta do seguinte comando:
       ```
       cat /proc/cpuinfo | grep -c processor
       ```
     - Se o número retornado pelo comando acima for igual a 4, por exemplo, e a matriz de saída possuir 9 células, será necessário o disparo de 3 lotes de threads: o primeiro e o segundo lotes com 4 threads e, no lote final, sobrará uma thread para ser lançada e calcular o valor do produto da célula remanescente.
   - **Questão de análise:**
     - Compare o desempenho em termos de tempo de execução das três aplicações e justifique eventuais diferenças de desempenho obtidas. Em sua resposta, indique qual foi a implementação mais rápida em termos de tempo de execução.
       - Para essa solução, será necessário usar esquemas de bechmarking para comparar os tempos de execução das aplicações.

Instruções e Recomendações

A submissão das respostas aos problemas dos trabalhos deverá ser feita através do Moodle da disciplina.

Cada Problema do Trabalho 02 deverá ser entregue em um pacote ZIP. A dupla de alunos deverá nomear o pacote ZIP da seguinte forma: nome_sobrenome_matricula_nome_sobrenome_matricula_**trab02.**zip.

Entre os artefatos esperados, listam-se:

- códigos-fonte C das soluções dos problemas;
- documentação mínima da aplicação:
  - qual sistema operacional foi usado na construção do sistema;
  - qual ambiente de desenvolvimento foi usado;

- quais são as telas (instruções de uso)
- quais são as limitações conhecidas

Não devem ser submetidos executáveis.

Códigos-fonte C com erros de compilação serão desconsiderados (anulados).

Os trabalhos poderão ser realizados em duplas; a identificação de cópia ou plágio irá provocar anulação de todos os artefatos em recorrência.