PRÁTICA 02 – PROGRAMAÇÃO MULTITHREAD

CONTEÚDO

1. INTRODUÇÃO

1.1. Objetivo

- Fortalecer os aspectos teóricos e práticos relacionados a programação envolvendo múltiplos threads por processo.
- Entender as primitivas básicas para manipulação de threads do Padrão Posix Threads (PThreads).

1.2. Abordagem Metodológica

- Formação de equipes de trabalho.
- Usar uma linguagem de programação C, compilador padrão C99. Compilar com /usr/bin/gcc, usando opção *–std=c99 -Wall*
- Pesquisar aspectos teóricos e práticos relacionados a programação concorrente com múltiplos threads.
- Realizar a implementação de exemplos simples para entendimento inicial da abordagem de programação, seguida pela implementação de exemplos complexos, que permitam consolidar os aspectos técnicos.
- Promover discussão das diferentes implementações -- considerando decisões de projeto, dificuldades encontradas e soluções implementadas.

1.3. Considerações Gerais

- As atividades devem ser realizadas por equipes.
- Durante o semestre, serão agendados encontros online nos quais os estudantes realizarão a apresentação das implementações desenvolvidas. Nestas apresentações, os estudantes serão avaliados individualmente.
- As notas serão concedidas de acordo com o desempenho em termos dos aspectos teóricos e práticos de cada estudante nas apresentações.
- Como as apresentações serão feitas a posteriori, é interessante que os relatórios solicitados nas atividades tenham o nível de detalhamento adequado para permitir que os conceitos e decisões de projetos utilizadas sejam recordados para serem discutidos durante as apresentações.

2. PROBLEMAS DE PROGRAMAÇÃO

Os problemas de programação descritos a seguir foram extraídos ou adaptados de Silberschatz (2015).

2.1. Problemas Básicos de Programação

Nesta seção serão considerados programas básicos, envolvendo múltiplos thread em um mesmo processo. Para tanto, os estudantes devem usar funções implementadas pela biblioteca padrão PThreads (POSIX Threads) do Linux.

Obs: Compilar os programas a seguir com o -lpthread (threads Posix).

Conjectura de Collatz multithread

A *conjectura de Collatz* preocupa-se com o que acontece quando tomamos um inteiro positivo n qualquer e aplicamos o seguinte algoritmo:

$$n = \begin{cases} \frac{n}{2}, & se\ n\ é\ par \\ 3n+1, & se\ n\ é\ impar \end{cases}$$

A conjectura estabelece que, quando esse algoritmo é aplicado continuamente, todos os inteiros positivos acabam chegando a 1. Por exemplo, se $n = 35$, a sequência é:

$$35, 106, 53, 160, 80, 40, 20, 10, 5, 16, 8, 4, 2, 1$$

Escreva um programa em C que gere essa sequência em um thread-filho, usando a função $pthread\_thread()$. O número inicial será fornecido na linha de comando. Por exemplo, se 8 for passado como parâmetro na linha de comando, o thread-filho exibirá $8, 4, 2, 1$. Faça o thread-pai invocar a chamada $pthread\_join()$ para esperar que o thread-filho seja concluído, antes de sair do programa. Execute a verificação de erros necessária para assegurar que um inteiro positivo seja passado na linha de comando.

Fibonacci Multithread

A sequência de Fibonacci é a série de números $0, 1, 1, 2, 3, 5, 8, \ldots$ Formalmente, pode ser expressa como:

$$fib_0 = 0$$
$$fib_1 = 1$$
$$fib_n = fib_{n-1} + fib_{n-2}$$

Escreva um programa com múltiplos threads que gere a sequência de Fibonacci. Esse programa deve funcionar da seguinte forma: Na linha de comando, o usuário dará entrada em quantos números de Fibonacci o programa deve gerar. O programa criará, então, um thread separado que gerará os números de Fibonacci, colocando a sequência em dados que possam ser compartilhados pelos threads (um array é, provavelmente, a estrutura de dados mais conveniente). Quando o thread encerrar a execução, o thread-pai exibirá a sequência gerada pelo thread-filho. Já que o thread-pai não pode começar a exibir a sequência de Fibonacci até o thread-filho ser concluído, ele terá que esperar o encerramento do thread-filho.

2.2. Problemas Avançados de Programação

O objetivo destes problemas é compreender melhor o conceito de thread e como eles podem ser usados para utilizar melhor os recursos de processamento disponíveis nos computadores modernos.

Obs: Compilar os programas a seguir com o -lpthread (threads Posix).

Validador de Solução de Sudoku Multithread

Um quebra-cabeça Sudoku usa uma grade de 9×9, em que cada coluna e cada linha, assim como cada uma das nove subgrades 3×3, devem conter todos os dígitos $1 \ldots 9$. A Figura 1 apresenta um exemplo de Sudoku válido.

6	2	4	5	3	9	1	8	7
5	1	9	7	2	8	6	3	4
8	3	7	6	1	4	2	9	5
1	4	3	8	6	5	7	2	9
9	5	8	2	4	7	3	6	1
7	6	2	3	9	1	4	5	8
3	7	1	9	5	6	8	4	2
4	9	6	1	8	2	5	7	3
2	8	5	4	7	3	9	1	6

Figura 1 - Solução para quebra-cabeça Sudoku 9x9

Esse projeto consiste na elaboração de uma aplicação com múltiplos threads que determine se a solução para um quebra-cabeça Sudoku é válida.

Existem várias maneiras diferentes de tornar essa aplicação multithreaded. Uma estratégia sugerida é criar threads que verifiquem os critérios a seguir:

- Um thread para verificar se cada coluna contém os dígitos de 1 a 9.
- Um thread para verificar se cada linha contém os dígitos de 1 a 9.
- Nove threads para verificar se cada uma das subgrades 3×3 contém os dígitos de 1 a 9.

Isso resultaria em um total de onze threads separados para a validação de um quebra-cabeça Sudoku. No entanto, você é incentivado a criar ainda mais threads para esse projeto. Por exemplo, em vez de criar um thread para verificar todas as nove colunas, você poderia criar nove threads separados e fazer cada um verificar uma coluna.

Passando Parâmetros para Cada Thread

O thread-pai criará os threads de trabalho, passando para cada um a localização que ele deve verificar no grid do Sudoku. Esse passo demandará a passagem de vários parâmetros para cada thread. A abordagem mais fácil é criar uma estrutura de dados com o uso de *struct*. Por exemplo, uma estrutura para passar a linha e a coluna que um thread deve começar a validar teria essa aparência:

```
/* estrutura para passagem de dados aos threads */
typedef struct{
        int row;
        int columm;
} parameters;
```

O programa usando Pthreads criará os threads de trabalho usando uma estratégia semelhante à mostrada abaixo:

```
parameters *data = (parameters *) malloc(sizeof (parameters));
data->row = l;
data->columm = l;
```

O ponteiro data será passado à função $pthread\_create()$ que, por sua vez, o passará como parâmetro à função a ser executada como um thread separado.

Retornando Resultados para o Thread-Pai

Cada thread de trabalho determinará a validade de uma região específica do quebra-cabeça Sudoku. Uma vez que um thread de trabalho tenha executado essa verificação, ele deve retornar seus resultados para o thread-pai. Uma boa forma de manipular isso é criando um vetor de valores inteiros que seja visível a cada thread. O i-ésimo índice desse vetor corresponde ao i-ésimo thread de trabalho. Se um thread de trabalho estabelecer seu valor como 1, indicará que sua região do quebra-cabeça Sudoku é válida. Um valor igual a 0 indicaria o contrário. Quando todos os threads de trabalho tiverem terminado, o thread-pai verificará cada entrada no vetor de resultados para determinar se o quebra-cabeça Sudoku é válido.

Ordenação (Classificação) Multithread

Escreva um programa de ordenação com múltiplos threads que funcione da seguinte forma: Uma lista de inteiros é dividida em duas listas menores de tamanho igual. Dois threads separados (que chamaremos de threads ordenadores) ordenam cada sublista usando um algoritmo de ordenação de sua escolha. As duas sublistas são, então, mescladas por um terceiro thread - um thread de mesclagem - que as combina em uma única lista ordenada.

Já que dados globais são compartilhados por todos os threads, talvez a maneira mais fácil de configurar os dados

seja criar um array global. Cada thread de ordenação trabalhará em uma metade desse array. Um segundo array global, com o mesmo tamanho do array de inteiros desordenados, também será definido. O thread de mesclagem combinará as duas sublistas nesse segundo array. Graficamente, esse programa tem a estrutura mostrada na Figura 2.

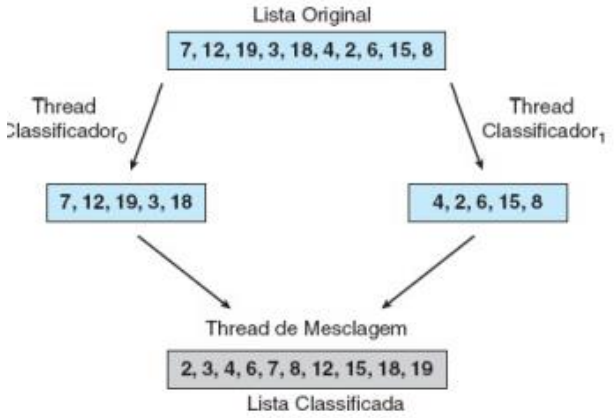

Figura 2 - Ordenação com múltiplos threads

Esse projeto de programação demandará a passagem de parâmetros para cada um dos threads de ordenação. Especificamente, será necessário identificar o índice inicial a partir do qual cada thread deve começar a ordenar. Consulte as instruções no "Problema do Validador Sodoku Multithread" para ver detalhes da passagem de parâmetros para um thread.

O thread-pai exibirá o array ordenado, uma vez que todos os threads de ordenação tenham terminado.

3. Entregáveis

- Códigos fontes devidamente comentados.
- Relatório contendo documentação dos códigos-fontes.
- Relatório técnico descrevendo todos os passos do desenvolvimento, incluindo descrições dos conceitos relacionados, testes realizados, procedimentos de compilação e implantação do código etc. [PONTUAÇÃO EXTRA]

4. Pontuação

Item	Pontuação
Implementação e documentação dos programas básicos Conjectura de Collatz e Fibonacci)	20%
Implementação e documentação dos programas avançados (Validador de solução sudoku e ordenação multithread)	70%
Verificação e manipulação correta de todas as condições de erro	5%
Bom estilo de codificação (e.g., boa formatação do código, nomes de variáveis significativos, comentários adequados e boa documentação do código fonte etc.)	5%
Relatório técnico (Pontuação extra)	5%

5. Referências Bibliográficas

Barney, B. **POSIX Threads Programming**. *Lawrence Livermore National Laboratory*. Disponível em: https://computing.llnl.gov/tutorials/pthreads/. Acesso em: 21/09/2020

Silberschatz, A.; Galvin, P. B.; Gagne, G. **Fundamentos de Sistemas Operacionais**. 9a ed., 2015, Willey.

Tanenbaum, A. S. Sistemas Operacionais Modernos. Prentice-Hall, 3ª edição, 2008.

Stallings, W. **Operating System: Internals and Design Principles**. 7th ed., 2012, Prentice Hall.

Mazeiro, C. A. **Sistemas Operacionais: Conceitos e Mecanismos**. 2019, Disponível em: http://wiki.inf.ufpr.br/maziero/doku.php?id=so:start. Acessado em: 17/09/2019

Exemplos de programas com múltiplos processos e threads

- http://wiki.inf.ufpr.br/maziero/doku.php?id=so:criacao_de_processos
- http://wiki.inf.ufpr.br/maziero/doku.php?id=so:criacao_de_threads