Péricles J. E. S. Moreira Borges

THREADS E SUA UTILIZAÇÃO EM PROGRAMAÇÃO NA LINGUAGEM C

Resumo

A escolha entre o uso de threads ou processos depende das necessidades específicas do programa e dos recursos disponíveis. Threads são mais adequadas para tarefas que envolvem compartilhamento de dados e sincronização frequente, enquanto processos são mais indicados quando o isolamento de recursos é necessário ou quando as tarefas são independentes e não precisam compartilhar dados.

Palavras-chave: Threads, programa, programação.

Programação Sistemas Computacionais

Numero de palavras: 883

Universidade do Mindelo

2022/2023

Parte 1 - Threads

1.1 O que é uma thread? Uma thread, também conhecida como "linha de execução" ou "fluxo de controle", é uma sequência de instruções que pode ser executada concorrentemente com outras threads em um programa. Cada thread representa uma unidade básica de processamento que pode realizar tarefas de forma independente. Threads compartilham o mesmo espaço de endereçamento e recursos do processo pai, permitindo a execução paralela de tarefas e o aproveitamento eficiente dos recursos do sistema.

1.2 Por que usar threads? Há várias razões para utilizar threads em programação:

1.2.1 Responsividade: Ao dividir uma tarefa em threads separadas, é possível manter o programa responsivo, mesmo quando ocorrem operações bloqueantes, como entrada/saída de dados. Enquanto uma thread está bloqueada, outras threads podem continuar executando, garantindo que o programa não fique parado aguardando a conclusão de uma única tarefa.

1.2.2 Compartilhamento de recursos: As threads podem compartilhar recursos do processo, como memória, arquivos abertos e conexões de rede. Isso facilita a comunicação e a cooperação entre diferentes partes do programa. Além disso, o compartilhamento de recursos evita a duplicação desnecessária de dados e ajuda a otimizar o uso dos recursos do sistema.

1.2.3 Modularidade: Ao dividir um programa em threads, é possível organizar a lógica em módulos independentes. Isso facilita a compreensão e a manutenção do código, além de permitir a reutilização de componentes em diferentes contextos.

1.3 Implementações de threads:

Existem várias implementações de threads disponíveis, cada uma com suas características e suporte em diferentes sistemas operacionais. Algumas das implementações mais comuns são:

1.3.1 POSIX threads (pthreads): As POSIX threads são uma biblioteca de programação de threads disponível em sistemas operacionais Unix-like, como Linux e macOS. A API das pthreads oferece funções para criar, sincronizar e gerenciar threads. Ela fornece uma interface consistente e amplamente utilizada para a programação de threads em C.

1.3.2 Win32 threads: Para programação em ambientes Windows, a biblioteca de threads do Win32 é amplamente utilizada. Ela oferece funcionalidades semelhantes às pthreads, permitindo a criação e sincronização de threads.

1.3.3 Java threads: A linguagem de programação Java possui suporte embutido para threads. Através das classes Thread e Runnable, é possível criar e controlar threads em programas Java.

1.3.4 Outras implementações: Além das implementações mencionadas, existem diversas outras bibliotecas e frameworks que fornecem suporte a threads em diferentes linguagens de programação, como C++11 threads, .NET threads, entre outras.

Parte 2: Threads vs. Processos

2.1 Threads: As threads são unidades de execução leves que compartilham o mesmo espaço de endereçamento e recursos do processo pai. Elas são adequadas para tarefas que exigem compartilhamento de dados e sincronização frequente entre diferentes partes do programa. As threads são criadas mais rapidamente e consomem menos recursos do sistema em comparação com os processos.

2.2 Processos: Os processos são unidades de execução independentes com espaços de endereçamento separados. Cada processo possui seu próprio espaço de memória e recursos, o que garante o isolamento entre eles. Os processos são mais adequados para tarefas que não requerem compartilhamento de dados ou que exigem isolamento de recursos. A criação de um processo é mais demorada e consome mais recursos do sistema em comparação com a criação de uma thread.

2.3 Comparação: A escolha entre o uso de threads e processos depende das necessidades específicas do programa. Alguns pontos de comparação entre threads e processos são:

2.3.1 Compartilhamento de recursos: Threads compartilham o mesmo espaço de endereçamento e recursos do processo pai, o que facilita o compartilhamento de dados e a comunicação entre elas. Processos, por outro lado, possuem espaços de endereçamento separados e requerem mecanismos de comunicação interprocessual, como pipes ou sockets, para compartilhar informações.

2.3.2 Criação e término: A criação de uma nova thread é mais eficiente em termos de recursos do sistema em comparação com a criação de um novo processo. Da mesma forma, o término de uma thread é mais rápido do que o término de um processo.

2.3.3 Sincronização: Threads compartilham a mesma memória, o que facilita a sincronização e a comunicação entre elas. Por outro lado, processos requerem mecanismos de sincronização mais complexos, como semáforos, memória compartilhada ou sinais, para cooperar e compartilhar informações.

2.3.4 Isolamento: Threads compartilham o mesmo espaço de endereçamento, o que significa que uma falha em uma thread pode afetar todo o processo. Em contraste, processos têm espaços de endereçamento separados, o que proporciona um maior nível de isolamento. Isso pode ser vantajoso em situações em que a falha de um componente não deve afetar o restante do sistema.

Referências Bibliográficas:

- Kerrisk, M. (2010). The Linux Programming Interface: A Linux and UNIX System Programming Handbook. No Starch Press.
- Butenhof, D. R. (1997). Programming with POSIX threads. Addison-Wesley Professional.
- Love, R. (2018). Linux System Programming: Talking Directly to the Kernel and C Library. O'Reilly Media.