Laboratório de Sistemas Operacionais (ECOS11A)

Prof Otávio Gomes (otavio.gomes@unifei.edu.br)

Roteiro 04

Gestão e controle de processos e threads

Ambientação

Este roteiro apresenta uma série de aplicações práticas que exploram o gerenciamento de processos e threads em sistemas operacionais, utilizando o ambiente Linux, o compilador GCC e a API POSIX, incluindo as bibliotecas unistd.h, sched.h, PThreads. O objetivo é fornecer um entendimento prático sobre como processos e threads são criados, gerenciados, e sincronizados em um sistema Linux.

Segue um passo a passo sobre como realizar estas tarefas usando o compilador GCC.

Passo 1: Instalar o GCC *– Já está instalado nas máquinas do Lab.*

```
sudo apt update
sudo apt install build-essential
```

Passo 2: Escolher um Editor de Texto

O Ubuntu vem com vários editores de texto que você pode usar para escrever código em C. Alguns editores populares incluem:

- Nano: Um editor de texto simples e fácil de usar no terminal.
- Gedit: O editor de texto gráfico padrão no ambiente de desktop GNOME.
- Vim ou Emacs: Editores de texto mais avançados para usuários que preferem uma experiência rica em recursos.

Passo 3: Criar e Editar um Arquivo Fonte em C na ferramenta Nano

- Abra o terminal.
- Navegue até o diretório onde você deseja salvar seu programa usando o comando **cd**.
- Digite **nano *meu_programa.c*** para criar e abrir um novo arquivo para edição no Nano.
- Escreva ou cole seu código em C no editor.

  Aqui está um exemplo simples de um programa em C que você pode usar:

```
#include <stdio.h>
int main() {
   printf("Olá, mundo!\n");
   return 0;
}
```

Salve o arquivo e saia do editor. No Nano, você pode fazer isso pressionando **CTRL + O**, **Enter** para salvar, e **CTRL + X** para sair.

Passo 4: Compilar o Código com GCC

Para compilar seu código-fonte em C e criar um executável, use o seguinte comando no terminal, certificando-se de estar no mesmo diretório do seu arquivo .c: **gcc meu_programa.c -o meu_programa**

Passo 5: Executar o Programa Compilado

Após a compilação, você pode executar seu programa digitando:

./meu_programa

Se tudo estiver correto, você verá a saída do seu programa no terminal, como *"Olá, mundo!"* no exemplo dado.

1) Criação e Execução de Processos com fork()

O gerenciamento eficaz de processos é um aspecto crucial do desenvolvimento de software em sistemas operacionais baseados em UNIX, como Linux. Um processo, no contexto de sistemas operacionais, é uma instância de um programa em execução. É essencial para programadores entender como criar, executar e gerenciar processos para desenvolver aplicações robustas e eficientes. A chamada de sistema fork() em C é uma ferramenta poderosa usada para criar processos. Este exercício visa introduzir o conceito de criação de processos usando fork(), explorando a relação dinâmica entre processos pai e filho.

O fork() é uma chamada de sistema que cria um novo processo duplicando o processo existente. O processo recém-criado é conhecido como processo filho, enquanto o processo original é o processo pai. Essa duplicação implica que o processo filho é quase uma cópia exata do pai, incluindo o espaço de código, variáveis globais, e o contador de programa, mas com espaços de endereçamento separados. Após a execução de fork(), ambos os processos, pai e filho, continuam sua execução a partir do ponto onde fork() foi chamado, mas podem ser diferenciados pelo valor retornado por fork().

O código 1, disponibilizado pelo professor, demonstra a utilização da chamada de sistema fork() para criar um processo filho dentro de um programa C.

2) Utilizando Threads com PThreads

No desenvolvimento de software moderno, especialmente em sistemas operacionais como Linux, a programação concorrente é um conceito crucial. Ela permite que um programa execute várias sequências de operações simultaneamente, melhorando o desempenho e a eficiência. Threads, unidades leves de execução dentro do mesmo espaço de memória de um processo, são fundamentais para alcançar a concorrência. Este exercício introduz o uso de threads POSIX (PThreads), uma API padrão para thread em sistemas Unix-like, para criar e executar múltiplas threads.

O objetivo é familiarizar-se com as funções básicas da biblioteca PThreads, como criar, iniciar e sincronizar threads. Ao concluir este exercício, você ganhará uma compreensão prática de como threads podem ser usadas para executar diferentes tarefas de forma concorrente dentro do mesmo programa.

Analise e execute o código 2, disponibilizado pelo professor

3) Sincronização com Mutex

À medida que avançamos na exploração da programação concorrente com threads, um desafio comum emerge: a sincronização. A execução concorrente de threads muitas vezes leva ao acesso simultâneo a recursos compartilhados, podendo causar inconsistências e erros difíceis de depurar, conhecidos como condições de corrida.

Para gerenciar o acesso seguro a esses recursos, utilizamos mecanismos de sincronização, sendo um dos mais fundamentais o Mutex (Mutual Exclusion). Este exercício foca na aplicação de Mutex para garantir que,

mesmo em um ambiente concorrente, o acesso a variáveis compartilhadas seja realizado de maneira ordenada e segura, evitando as condições de corrida.

Analise e execute o código 3, disponibilizado pelo professor

4) Gerenciamento de Prioridade de Processos com nice()

No ambiente multitarefa dos sistemas operacionais Linux, cada processo recebe uma parcela de tempo de CPU para executar suas operações. A distribuição desse tempo é influenciada por várias prioridades, determinando quais processos são mais urgentes ou importantes.

A função nice() em C é uma chamada de sistema que ajusta a prioridade de execução de um processo, permitindo que os desenvolvedores influenciem o agendamento de processos pelo sistema operacional. Este exercício tem como objetivo familiarizar os participantes com a manipulação de prioridades de processos usando nice(), promovendo uma compreensão prática de como as prioridades afetam a execução de processos em sistemas baseados em UNIX/Linux.

Analise e execute o código 4, disponibilizado pelo professor