Universidade Federal de Uberlândia

Disciplina: Sistemas Operacionais (SO)

Prática: Sincronização

Sincronização em Ambiente LINUX

Threads em Ambiente LINUX

Essa atividade está dividida em duas partes, sendo que na Parte 1 os alunos devem explorar a sincronização das threads em ambiente do Linux. São apresentados comandos para que os estudantes possam se familiarizar com as principais funções empregadas para sincronização dos threads em POSIX. Na Parte 2, será aplicado um questionário com questões objetivas sobre os tópicos abordados em nossa disciplina.

Parte 1: Sincronização entre os Threads em Recursos Compartilhados

1. Variável do tipo trava (Lock)

Uma solução para travar o acesso a região crítica é empregar a variável de impedimento (lock) em que o processo precisa consultá-la para entrar nesta região. Compile o código **variavel_lock.c** para observar o uso desse recurso entre duas funções com threads. Após compilação execute o código e observe se essa abordagem consegue garantir o impedimento na região crítica. Modifique o código para que essas funções seja repetidas 3 vezes no processo de criação e faça o mesmo com a função pthread_join. Como será o comportamento após essa modificação? Retire a opção volatile da variável compartilhada? Essa modificação permite manter o mesmo comportamento? Quais são as principais limitações dessa solução para aplicação em um sistema operacional?

2. Chaveamento Obrigatório

A estratégia chaveamento obrigatório emprega uma variável “turn” que serve para controlar a vez de quem entra na região crítica e verifica ou atualiza a memória compartilhada. Compile o código **spin_l.c** para observar o uso desse recurso entre as funções com threads. Após compilação execute o código e observe se essa abordagem consegue garantir o impedimento na região crítica. Retire o volatile da variável compartilhada? Essa execução tem o mesmo comportamento? Comente a função sleep o que isso provoca? Modifique o tempo do sleep e alguma das funções fora da região crítica impede a outra de uma execução?

3. O Problema do Produtor e Consumidor

A característica de execução para o problema é que há dois processos que compartilham um buffer de tamanho limitado. O processo “produtor” produz um dado, insere no buffer e volta a gerar dado e o processo “consumidor” consome o dado do buffer (um por vez).

a) Compile e execute o código **prod-cons-1.c.** O problema é como garantir que o produtor não adicionará dados no buffer se este estiver cheio? Como garantir que o consumidor não vai remover dados de um buffer vazio? As threads conseguem garantir uma sincronização pelo recurso compartilhado? Por que pode ocorrer a condição de corrida nesse código? Dos mecanismos estudados, há algum que garanta a exclusão múltua? Caso tenha algum mecanismo, descreva a característica desse mecanismo de sincronização. Esse codigo pode ocorrer deadlock?

b) Compile e execute o código **prod-cons-2.c.** As threads conseguem garantir uma sincronização pelo recurso compartilhado? Há algum mecanismo que garanta a exclusão mútua? Caso tenha algum mecanismo, descreva a característica desse mecanismo de sincronização. Nesse código pode ocorrer o problema de starvation?

c) Compile e execute o código **prod-cons-3.c.** Análise o código. As threads conseguem garantir uma sincronização pelo recurso compartilhado? Há algum mecanismo que garanta a exclusão mútua? Caso tenha algum mecanismo, descreva a característica desse mecanismo de sincronização. Nesse código pode ocorrer starvation ou deadlock? Qual a função da chamada sem_destroy?

d) Compile e execute o código **mult-prod-cons-4.c.** Análise o código e observe se os threads conseguem garantir uma sincronização pelo recurso compartilhado? Há algum mecanismo que garanta a exclusão mútua? Caso tenha algum mecanismo, descreva a característica desse mecanismo de sincronização em relação ao número de threads. Nesse código pode ocorrer starvation ou deadlock? Qual a função da chamada sem_destroy? Por que neste código são empregados 2 semáforos antes de entrar na região crítica? O que cada um desses semáforos estão controlando neste código.