Sistemas Distribuídos - CEFET - MG
Professora: Michelle Hanne
Trabalho Prático 1

2025/2

1 Objetivos

O objetivo deste trabalho é se familiarizar com os principais mecanismos de IPC (Interprocess Communication) baseados em troca de mensagens, threads e mecanismos de sincronização. Para cada parte, você deve desenvolver um programa na linguagem de programação de sua preferência mas que tenha suporte a threads e mecanismos de sincronização, como instruções atômicas e semáforos. A sugestão é utilizar C ou C++, tendo em vista a proximidade das bibliotecas destas linguagens com o sistema operacional, oferecendo ao desenvolvedor (você) um maior controle.

Além da implementação, você deve testar seu programa, rodando os estudos de casos. Você deve preparar um relatório, com no máximo 5 páginas, com as decisões de projeto e implementação das funcionalidades especificadas, assim como a avaliação dos estudos de caso. O relatório deve conter a URL para o código-fonte da sua implementação. O trabalho pode ser realizado em dupla.

2 Pipes

Implemente o programa Produtor-Consumidor como vimos em aula com dois processos que utilizam *pipes* (*anonymous pipes*, para ser mais preciso) para fazer a comunicação.

O programa produtor deve gerar números inteiros aleatórios e crescentes, da seguinte forma:

$$N_i = N_{i-1} + \Delta, \quad N_0 = 1, \quad \Delta \in [1, 100].$$

O programa consumidor deve receber o número e verificar se o mesmo é primo, imprimindo o resultado no terminal. Seu programa deve primeiramente criar um *pipe* e depois fazer um `fork()` para duplicar o processo, de forma que os dois processos (pai e filho) tenham as duas respectivas pontas do *pipe* (*write end* e *read end*).

O processo consumidor deve terminar quando receber o número 0. O programa produtor tem como parâmetro o número de números a serem gerados (ex.: 1000), depois do qual o número zero é enviado, e o produtor termina sua execução.

Cuidado com a representação numérica ao escrever no *pipe*!
Dica: converta o número para uma string de tamanho fixo, por exemplo, 20 bytes. Escreva e leia do *pipe* este mesmo número de bytes para cada mensagem.

Teste o seu programa mostrando seu funcionamento para alguns casos.

3 Produtor-Consumidor com Semáforos

Implemente um programa Produtor-Consumidor multithreaded com memória compartilhada. Assuma que a memória compartilhada é um vetor de números inteiros de tamanho N. O número de threads do tipo produtor e consumidor são parâmetros do programa dados por N_p e N_c, respectivamente. A thread produtor deve gerar números inteiros aleatórios entre 1 e 10^7 e colocar o número em uma posição livre da memória compartilhada. A thread consumidor deve retirar um número produzido por um produtor da memória compartilhada, liberar a posição do vetor, e verificar se o mesmo é primo, imprimindo o resultado no terminal.

Repare que a memória compartilhada será escrita e lida por várias threads, então o acesso deve ser serializado, evitando efeitos indesejáveis da condição de corrida. Utilize semáforos para serializar o acesso à memória compartilhada. Repare ainda que quando a memória compartilhada estiver cheia ou vazia, as threads produtor ou consumidor devem aguardar bloqueadas, respectivamente. Ou seja, uma thread produtor aguarda até que haja uma posição de memória livre, e uma thread consumidor aguarda até que haja uma posição de memória ocupada. Utilize semáforos contadores para esta coordenação, como visto em aula.

Para o estudo de caso, considere que o programa termina sua execução após o consumidor processar $M = 10^5$ números. Considere ainda os valores $N = 1, 10, 100, 1000$, com os seguintes combinações de número de threads produtor/consumidor:

$$(N_p, N_c) \in \{(1,1), (1,2), (1,4), (1,8), (2,1), (4,1), (8,1)\}.$$

Para cada combinação de parâmetros, obtenha o tempo de execução do seu programa, rodando o programa 10 vezes para calcular o tempo médio de execução. Apresente um gráfico mostrando o tempo médio de execução em função do número de threads produtor/consumidor para cada valor de N (cada N deve ser uma curva no gráfico). Analise o comportamento observado.

Para cada cenário, trace um gráfico representativo com a ocupação do buffer compartilhado ao longo do tempo. Para gerar o gráfico, utilize um vetor que armazena a ocupação do buffer após cada operação de produção ou consumo. Ao final, use o vetor para gerar o gráfico (ou arquivo).

O que você pode concluir em cada um dos casos?