Sistemas Distribuídos – COS470

2020/1

Prof. Daniel R. Figueiredo

Trabalho Prático 1

1 Objetivos

O objetivo deste trabalho é se familiarizar com os principais mecanismos de IPC (Interprocess Communication) baseados em troca de mensagens. Para cada parte, você deve desenvolver um programa na linguagem de programação de sua preferência mas que tenha suporte direto aos mecanismos de IPC. 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 poder de fogo.

Além da implementação, você deve testar seu programa, rodando os estudos de casos. Você deve preparar um relatório, com no máximo 4 páginas, com as decisões de projeto e implementação das funcionalidades especificadas, assim como a avaliação dos estudos de caso. O trabalho pode ser realizado individualmente ou em dupla.

2 Sinais

Nesta tarefa você tem que escrever dois programas.

- O primeiro programa deve ser capaz de enviar um sinal a qualquer outro processo. Este programa recebe como parâmetros o número do processo destino e o sinal que deve ser enviado. Seu programa deve verificar se o processo passado por parâmetro existe, e retornar um erro em caso caso negativo. Caso contrário, seu programa deve enviar o sinal desejado.

- O segundo programa deve ser capaz de receber alguns sinais específicos. Para isto, você vai precisar definir *signal handlers*. Seu programa deve capturar e reagir a três sinais diferentes (de sua escolha), ou mais, imprimindo no terminal uma mensagem diferente para cada sinal. Além disso, um dos sinais sendo capturados deve terminar a execução do programa, ou seja, sua *signal handler* deve terminar o processo. Repare que após estipular as *signal handlers* seu programa fica aguardando a chegada de sinais. Você deve implementar duas formas de esperar, *busy wait* e *blocking wait* (passado como parâmetro para o programa). Descubra como implementar cada um destas formas de esperar!

- Teste seus programas fazendo com que um envie sinais para o outro. Use também o programa *kill* para enviar sinais para o seu segundo programa.

3 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: $N_i = N_{i-1} + \Delta$, onde $N_0 = 1$ e Δ é um valor aleatório entre 1 e 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 duas respectivas pontas do pipe (*write end* e *read end*). O programa 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 execução.

Cuidado com a representação numérica através do pipe! Dica: converta o número para uma string de tamanho fixo, ex. 20 bytes. Teste o seu programa mostrando seu funcionamento para alguns casos.

4 Sockets

Implemente um programa Produtor-Consumidor *com resposta* com dois processos e *sockets* para fazer a comunicação. Novamente, o programa produtor deve gerar números inteiros aleatórios e crescentes (conforme acima) e o programa consumidor deve receber o número e verificar se o mesmo é primo. Diferentemente do exercício acima, o programa consumidor deve enviar uma mensagem ao produtor informando se o número recebido é ou não primo, com este último imprimindo o resultado no terminal. Implemente um *protocolo* bem simples, onde o produtor gera um número, envia ao consumidor, e aguarda a resposta de forma bloqueante. O programa consumidor deve se conectar ao programa produtor, e aguardar a chegada de um número, de forma bloqueante, para então determinar se o mesmo é primo, enviar o resultado ao produtor, e voltar a aguardar a chegada do próximo número. Utilize *sockets* to tipo TCP. Por fim, os programas devem terminar como no exercício acima (enviando zero para terminar o consumidor, tendo como parâmetro o número de números a serem gerados).

Novamente, cuidado com a representação numérica! Dica: converta o número para uma string de tamanho fixo, ex. 20 bytes. Teste o seu programa mostrando seu funcionamento para alguns casos. Teste seu programa utilizando computadores diferentes, se possível.