COS 470 – Sistemas Distribuídos

Trabalho 1

Universidade Federal
do Rio de Janeiro

- Danyel Clinário dos Santos - DRE 119045123
  - danyel.clinario@poli.ufrj.br

- Luiz Gustavo Costa Marques de Oliveira - DRE 119063888
  - luizgustavo.marques@poli.ufrj.br

Rio de Janeiro - RJ
05 de março de 2022 (2021.2)

1 Introdução

Esse trabalho foi realizado em C, pela maior familiaridade que possuímos com essa linguagem de baixo nível. Também aproveitamos trabalhos desempenhados na disciplina de Sistemas Operacionais.

2 Sinais

Realizamos o trabalho pedido, utilizando apenas um código para ambas partes. Isso foi feito pela maior facilidade de implementação e para encapsular o problema em apenas um lugar.

Fizemos um programa bem humorado. É o sistema de avaliação de uma loja de meias, um dos sistemas assume o papel de receber as notas e o outro as envia. As notas são os sinais.

O signal handler implementado possui a variável *toRun*, que indica qual o comportamento esperado. Também há a função *mustRun*, que implementa a busy wait, o código fica no loop enquanto a *toRun* não for 0 (um SIGTERM), e roda as rotinas quando o *mustRun* for setado em 1. Para blocking wait, usamos a função *pause()*, que coloca o processo para dormir até detectar um sinal, sem ocupar ciclos de CPU e sendo acordado pelo sistema operacional.

Os processos informam os PIDs no começo da execução, facilitando disparar os sinais para o processo certo. O código tenta capturar alguns erros e informar de forma clara, como o de PID inválido, implementado com o código antes de enviar as notas testa se o PID é referente a um processo rodando, usando o sinal 0.

Os parâmetros devem ser passados como argumentos na hora de rodar o código. Essa forma foi mais rápida para implementar e achamos que faria mais sentido num sistema do que esperar o input do teclado.

Abaixo estão o output de exemplo dos códigos. O signal handler foi capaz de receber os sinais enviados tanto pelo outro processo quanto rodando o programa kill no terminal.

Signal handler:
"~/TRABSD-1$ ./sinais.out rating_receiver 0
O PID do processo é 6520.
Pode-se escolher o tipo de espera passando 0 (busy) ou 1 (blocking) para a função
O tipo de espera escolhido foi o de: 0
Recebemos sua avaliação 1 e estamos encaminhando para o setor responsável.
Ficamos muito decepcionados que sua experiência tenha sido tão ruim a ponta de fazer você chorar.
Recebemos sua avaliação 2.
Obrigado por usar nossos serviços, mas na próxima iremos prover meias de melhor qualidade.
Recebemos sua avaliação 3.
Ficamos muito felizes que gostou de nosso serviço! Estamos um pouco preocupados que amou nossos serviços mais que seu conjugê, mas agradecemos.
Finalizando o disparador..."

Enviador de sinais:
"~/TRABSD-1$ ./sinais.out rating_signaler 6416 SIGUSR1
O PID do processo é 6520.
Olá, eu sou o processo que vai enviar as avaliações que você tiver da loja.
Lembrando que as notas são: SIGUSR1 (ruim), SIGUSR2 (aceitável), SIGHUP (excepcional) da nossa loja. Também é possível enviar SIGKILL para terminar o outro processo. O PID do processo handler recebido foi o 6416, e a nota enviada (o sinal) foi o 0.

Checamos que o PID é válido.
Nota enviada”

Enviador de sinais com PID inválido:
”~/TRABSD-1$ ./sinais.out rating_signaler 7000 SIGUSR1
O PID do processo é 6706.
Olá, eu sou o processo que vai enviar as avaliações que você tiver da loja.
Lembrando que as notas são: SIGUSR1 (ruim), SIGUSR2 (aceitável), SIGHUP (excepcional) da nossa loja. Também é possível enviar SIGKILL para terminar o outro processo. O PID do processo handler recebido foi o 7000, e a nota enviada (o sinal) foi o 0.
PID inválido.”

3 Pipes

Tomamos muitas decisões semelhantes. Criamos a função que gera números aleatórios com uma biblioteca baseado na hora que a função roda, e uma função simples que checa se é primo. Lemos do argumento passado a quantidade de números randômicos e fazemos nosso loop. Criamos um buffer pipe para comunicação

Usando a função fork(), fazemos um fork do processo, gerando um filho. O processo pai (produtor) fecha a ponta de leitura e o filho (consumidor) fecha a ponta de escrita, já que estamos implementando uma comunicação half-duplex.

O processo pai coloca no buffer os números produzidos e o filho checa se é impar. Tentamos intercalar os processos, fazendo o produtor produzir parte da produção e o consumidor consumir parte dela. Para isso, usamos a função sleep, que coloca o processo para dormir por alguns instantes enquanto a outra roda (ou não, cada rodada os processos têm 50% de chance de dormir ou não).

Abaixo segue o output:
”./pipes.out 5
Serão gerados 5 números hoje
Objeto ’1’ na linha de montagem
Consumindo objeto ’1’...
O 1 é primo. WOW!
Objeto ’10’ na linha de montagem
Objeto ’19’ na linha de montagem
Consumindo objeto ’10’...
O 10 não é primo.
Objeto ’28’ na linha de montagem
Objeto ’37’ na linha de montagem
Consumindo objeto ’19’...
O 19 é primo. WOW!
PARANDO A PRODUÇÃO
Consumindo objeto ’28’...
O 28 não é primo.
Consumindo objeto ’37’...
O 37 é primo. WOW!
PARANDO O CONSUMO”

4 Sockets

O código de sockets é de ideia semelhante ao código de pipes, continuamos usando as mesmas decisões. A diferença é que em vez de um buffer pipe, estamos abrindo sockets no lado de cada processo.

O funcionamento é dado por: o processo pai (consumidor e servidor), irá abrir seu socket e esperar conexão pelo filho (produtor e cliente). O filho irá se conectar e enviará o número para ser checado; o consumidor então irá avaliar e enviar a resposta. Tivemos muita dificuldade em realizar essa tarefa, e devemos agradecimentos ao blog https://www.educative.io/edpresso/how-to-implement-tcp-sockets-in-c pelo tutorial.

Por uma limitação física, não rodamos em máquinas separadas.

Exemplo de saída do código:
"./sockets.out 5
Serão gerados 5 números hoje
SERVIDOR: Socket criado com sucesso!
SERVIDOR: Conectado a porta.
SERVIDOR: Esperando conexão...
Socket criado com sucesso!
SERVIDOR: Cliente conectado no IP: 127.0.0.1 e porta: 56862
CLIENTE: Conectado ao servidor!
CLIENTE: Número para enviar ao servidor: 71
SERVIDOR: Número recebido: 71
CLIENTE: Mensagem enviada
CLIENTE: Mensagem do servidor: Esse número é primo. WOW!
CLIENTE: Número para enviar ao servidor: 90
SERVIDOR: Número recebido: 90
CLIENTE: Mensagem enviada
CLIENTE: Mensagem do servidor: Esse número não é primo.
CLIENTE: Número para enviar ao servidor: 10
SERVIDOR: Número recebido: 10
CLIENTE: Mensagem enviada
CLIENTE: Mensagem do servidor: Esse número não é primo.
CLIENTE: Número para enviar ao servidor: 2
CLIENTE: Mensagem enviada
SERVIDOR: Número recebido: 20
CLIENTE: Mensagem do servidor: Esse número não é primo.
CLIENTE: Número para enviar ao servidor: 4
CLIENTE: Mensagem enviada
SERVIDOR: Número recebido: 40
CLIENTE: Mensagem do servidor: Esse número não é primo.
CLIENTE: Máximo de iterações, finalizando...
CLIENTE: Mensagem enviada
SERVIDOR: Recebido 0, finalizando..."