Nome: Eduardo Fernando dos Santos Araujo DRE:115021397

Renan Fasolato Basilio DRE: \_\_\_\_\_\_\_\_

**Sistemas Distribuídos**

*Relatório - Trabalho Prático I*

Os programas criados nesse trabalho foram implementados nas linguagens C/C++ e testados em Sistemas Operacionais Windows. Na codificação foi utilizada a ide Notepad++ e os respectivos programas foram executados e compilados com o auxílio do Command Prompt (cmd.exe) e a biblioteca Cygwin.

**1) Sinais**

Na implementação dos sinais utilizamos dois processos, um que envia e outro que recebe sinais.

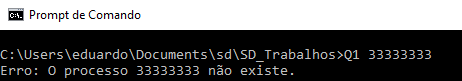
O primeiro é bem simples, em resumo, apenas realiza um comando kill, que recebe como parâmetros o identificador do processo (PID) e o tipo de sinal a ser enviado. O parâmetro referente ao PID é obtido via linha de comando (CLI), ao se executar o programa, e o tipo de sinal foi escolhido por nós (o SIGHUP). A função retorna 0 em caso de sucesso e 1 no caso de o processo não existir, sendo o último caso tratado como especificado no trabalho; há a exibição de uma mensagem de erro. 

Figura 1. Tentativa de enviar um sinal para um processo não existente.

O segundo processo ao ser executado via CLI requisita um modo de execução, que deverá ser busy (busy wait) ou block (blocking wait). O modo busy wait foi implementado com um simples while infinito que fica apenas aguardando algum tipo de sinal ser recebido. Enquanto que o modo blocking wait é bastante semelhante, sendo que dentro do while há uma função chamada pause, que suspende a execução do programa até que o mesmo receba um sinal.

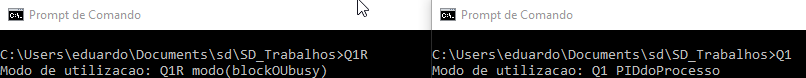
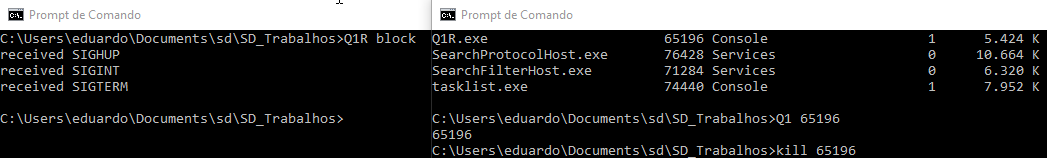


Figura 2: Execução via CLI do programa exigindo os parâmetros de entrada

O programa responde a 3 tipos diferentes de signal handlers, o SIGINT, o SIGHUP e o SIGTERM. Nos nossos testes, o SIGINT foi um sinal enviado pelo teclado (ctrl+c), o SIGHUP foi enviado pelo primeiro processo e o SIGTERM por comando via linha de comando, que encerrava o processo.

Figura 3. Receptor de sinal em execução junto às tentativas de envio de sinal

Esse envio de sinais por meio de outros processos ou pela linha de comando é bem interessante, mas para isso tornava-se necessário identificar o número do respectivo processo por meio de comandos na CLI (no Windows o comando tasklist) além do comando que envie um determinado sinal a um processo (o comando kill).

**2) Pipes**

A geração de números aleatórios crescentes foi feita somando-se a cada iteração um delta aleatório positivo, obtido por meio da função da rand e da srand com semente dada em função do tempo. A verificação do número ser ou não primo foi realizada por meio de uma função que averigua se o número é divisível apenas por 2 números, “varrendo” os números menores que ele até o número 1, onde a complexidade do algoritmo não foi priorizada, já que não era o objetivo central do estudo.

Na implementação propriamente do pipe, primeiramente criamos um pipe dando como argumento um vetor de dois inteiros, que fazem referência a leitura e a escrita do pipe, dois files descriptors que são correspondentes a entrada e a saída do tubo. Utilizamos também a função fork, que cria um novo “processo filho” e por meio de estruturas condicionais definimos funções que rodariam em cada um dos dois processos, a depender do valor do PID. Assim, definimos uma função produtor e outra consumidor, para serem executadas em cada um dos processos.

Basicamente a função produtor é responsável por escrever na pipe, através da função write, que recebe como parâmetro o file descriptor de escrita, o endereço da variável que contém o numero a ser enviado e o tamanho do buffer necessário. Já a função consumidor é responsável por ler os números gerados na função produtor e verificar se estes são primos. A leitura é feita por meio do comando read, semelhante ao write, sendo que para leitura utiliza-se o file descriptor de leitura. Ambos os processos aguardam um tempo aleatório de 1 a 5 segundos entre cada ação de leitura ou escrita. E é provável que os itens sejam produzidos mais rápidos do que consumidos porque há um processamento mais demorado em saber se o número é primo ou não no consumidor.

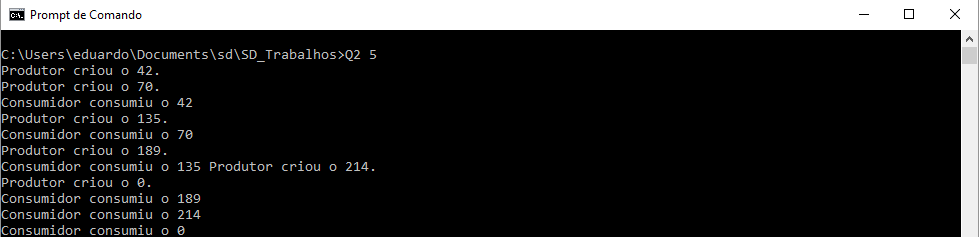


Figura 3. Execução do Pipe com 5 itens produzidos.

Mantendo o tempo de espera do Produtor e diminuindo o do consumidor (para 1 segundo de espera entre cada ação de consumo), notamos que há a possiblidade dessa relação sucessiva de consumo e produção ser praticamente de um para um.

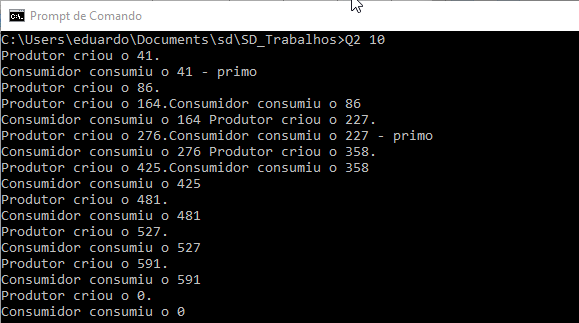


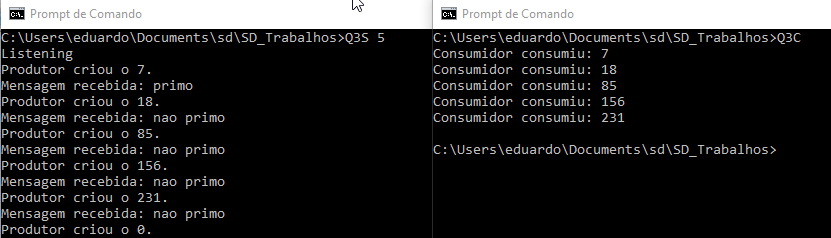
Figura 4. Execução da pipe com consumidor com tempo reduzido de espera.

**3) Sockets**

As funcionalidades são muito semelhantes ao do programa acima, aproveitamos boa parte da estrutura do código, tanto de gerar números aleatórios quanto a verificação do número primo, por exemplo. Criamos dois programas, o servidor e o cliente, que se comunicam por meio de sockets. Primeiramente há uma configuração e inicialização do socket, especificando estilo de comunicação, tipo de protocolo, porta, e endereço ip a ser conectado.

No servidor, há uma função (a listen) que ouve a porta e aguarda que uma conexão seja iniciada, enquanto que o cliente possui uma função de conexão com o outro socket via TCP, a partir da porta e do número do ip do socket servidor. No servidor há a geração dos números aleatórios que são transformados em um array de caracteres (buffer) de 20 bytes e enviados ao cliente (função send) e ele então aguarda uma mensagem ser recebida de forma bloqueante e a exibe na tela. No cliente, espera-se receber uma mensagem do servidor para lê-la (função recv), verifica se a mensagem é “0”, caso seja termina o programa. Se não for, a converte para inteiro, passa para a função que verifica se é primo e envia o resultado de volta para o cliente. O número de itens que o produtor consome tem de ser informado via CLI.

Nesse modelo bloqueante de IPC, nota-se que o produtor só produz se o consumidor consumir e ações são feitas de forma ordenada. Enquanto que no pipe um produtor poderia produzir vários itens até que o consumidor consumisse um deles, a depender do intervalo de espera entre as ações.

Figura 5. Teste cliente-servidor com sockets TCP.