Software Básico 2008.2
Trabalho Prático 1: programação de E/S, uso de sinais
Prática de programação voltada a eventos
Trabalho individual ou em dupla
Data de entrega: 01/10/2008

1 O Objetivo

Utilizando as primitivas de manipulação de sinais, iniciação de processos filho, criação de arquivos e acesso a diretórios, implementar um programa de *chat* de mensagens baseado em comunicação por diretórios.

2 O Problema

Durante o curso da disciplina de Software Básico 2008.2 a professora Jussara decidiu fazer uma prova prática em laboratório de desenvolvimento em linguagem de montagem. Espertamente dois alunos da turma, Bob e Alice, tiveram a idéia de colar um do outro nesta prova fazendo uso de um programa de *chat*, afinal todo o trabalho seria feito em um dos laboratórios do DCC com o acesso a rede liberado. Porém, a professora Jussara prevendo o comportamento malicioso dos alunos da turma requisitou que o CRC bloqueasse o acesso de redes de todas as máquinas, liberando as mesmas para se comunicar apenas com o servidor no qual os exercícios deveriam ser submetidos. Jussara também requisitou que não fosse permitido a abertura de portas de rede no servidor de entrega, por fim foi requisitado que um *kernel* específico fosse utilizado pra o bloqueio de chamadas ao sistema de comunicação entre processos.

Bob e Alice resolveram buscar a ajuda de Vader e Maul, dois alunos veteranos do DCC experts na arte de colar. Vader e Maul decidiram ajudar Bob e Alice desenvolvendo um sistema de chat com base na comunicação entre arquivos e sinais, dado que o servidor bloqueava qualquer outra forma de comunicação inter-processos.

Os requisitos de Bob e Alice eram de que o sistema fosse capaz de:

- Permitir a troca de mensagens de texto entre os dois alunos;
- Permitir que um aluno enviasse para o outro uma cópia de prova;
- Gravar um log global de tudo que ocorreu para poderem revisar eventuais erros;
- E claro, executar no sistema restritivo da professora Jussara fazendo uso de arquivos e sinais.

Neste trabalho prático a dupla de alunos fará o papel de Vader e Maul desenvolvendo o sistema de chat com os requisitos acima[1].

[1]Embora Vader e Maul tenham sido excluídos do curso, a professora garante que o desenvolvimento deste TP não implicará em exclusão dos alunos que o fizerem, desde que os mesmos não façam como Vader e Maul, utilizando o sistema construído para atos excusos

3 Contextualização

Comunicação entre processos, na maioria dos casos, pode ser tratada como um problema de troca de mensagens, o que pode ser feito através de uma rede de computadores ou de chamadas do sistema operacional. Entretanto, em muitos casos essas soluções são mais complexas que o desejado, no caso deste trabalho são um requisito.

3.1 Comunicação Através de Arquivos e Diretórios

Uma forma comum de garantir a comunicação entre processos, sem exigir comandos especiais do sistema operacional, nem protocolos de redes, é através do uso de comunicação por arquivos e diretórios. Esse tipo de técnica é usada em diversos sistemas, pela sua simplicidade. Exemplos de sistemas que usam esse recurso são o spooler de impressão do Unix (arquivos a serem impressos são colocados em um diretório específico e tratados pelo programa lpd), o sistema de correio eletrônico (mensagens enviadas por qualquer usuário são colocadas em uma fila no disco para serem enviadas pela rede pelo agente de envio de mensagens do sistema) etc.

Como esse o processo de escrita e cópia de arquivos pode levar algum tempo, é preciso evitar que o receptor identifique o arquivo no sistema antes que toda a informação tenha sido gravada. Para evitar esse problema, programas frequentemente se valem do recurso de renomear arquivos: essa operação é comumente definida como atômica pelo sistema operacional (a troca de nome/diretório é percebida pelos processos como sendo instantânea). Sendo assim, é comum os programas utilizarem um outro diretório de trabalho onde um arquivo é criado e preenchido, para só depois ser renomeado (movido) para os diretórios de troca de mensagens e transferência.

3.2 Envio de Sinais

Processos podem fazer uso do conceito de sinais para o tratamento de eventos do usuário, de relógio ou de outros processos. Sinais atuam de forma similar às interrupções, quando recebidas por um processo sua execução é pausada, e a função de tratamento do sinal é executada. No Unix, o comando kill, apesar de seu nome, não serve apenas para terminar um processo. Ele pode ser usado para enviar um sinal qualquer a um processo; por exemplo, "kill -USR1 1234" envia o sinal denominado SIGUSR1 ao processo de número 1234. Existem diversos sinais predefinidos pelos sistemas operacionais. Para este trabalho, estamos interessado nos sinais SIGUSR1 e SIGUSR2, que são reservados para uso do usuário.

Para mais detalhes sobre o uso de sinais leia a página do The GNU C Library sobre tratamento de sinais[2]. Revisem atentamente as seções de espera por sinais. O programa teimoso.c, na página do curso de Software Básico, mostra exemplos de manipulação de sinais variados. Estude o código e confira as páginas de manual para as funções utilizadas.

[2]http://www.gnu.org/software/libtool/manual/libc/Signal-Handling.html

3.3 Execução de processos filho

As duas formas mais comuns de se iniciar um processo novo na linguagem C são as chamadas **fork** e **exec**. A primeira inicia um processo novo (filho) que é um cópia idêntica do processo original (pai). Por cópia idêntica, estamos nos referindo a uma cópia de: código, dados, variáveis e símbolos. A segunda chamada, **exec**, executa um outro executável pré-compilado sem nenhum cópia de informação do processo pai. Para mais informações sobre o uso destas chamadas leia a página referente do The GNU C Library[3].

4 Arquitetura e Implementação

Vader e Maul desenharam o sistema de chat de forma a ser composto de quatro processos, um *sender*, um *receiver*, um *logger* e um processo *chat*. O princípio de operação é bastante simples. Apenas um processo *logger* é executado no servidor, este é o primeiro a ser iniciado. Os processos *sender* e *receiver* são iniciados automaticamente pelo processo *chat*. Cada usuário executa um processo *chat* (por consequência, executam também um processo *sender* e um *receiver*).

Se Vader deseja se comunicar com Maul, o processo *sender* de Vader precisa se comunicar com o processo *receiver* de Maul. Isto ocorre da seguinte maneira. O processo *sender* (Vader) que deseja se comunicar com o *receiver* (Maul) deposita em um arquivo, de caminho previamente definido, a mensagem a ser enviada. Para a transferência de arquivos, os arquivos a serem transferidos são depositados em um diretório compartilhado também de caminho previamente definido. Por sua vez, os processos *receivers* lêem o arquivo de mensagens, imprimem o conteúdo do mesmo na tela e notifica o processo *logger*, que deverá então salvar a mensagem no log global. Este log global deverá apresentar as mensagens em ordem cronológica.

Para que processo *receiver* saiba quando pode ler o diretório, processo *logger* saiba quando pode logar e o processo *sender* saiba que pode enviar novas mensagens as primitivas de sinais são utilizadas. Sinais só podem ser enviados entre processos do mesmo usuário. Logo, Vader e Maul sugerem que Alice e Bob devam compartilhar um usuário único para que o sistema de chat funcione. O comportamento do programa para outros sinais pode seguir o padrão do sistema operacional (isto é, não é preciso tratar o Ctrl-C, por exemplo).

Vader e Maul, por serem inexperientes em tratamento de condições de corrida, arquitetaram o sistema de forma a evitar tais condições nas trocas de mensagens. Eles se baseiam no fato de que os sinais recebidos por um processo são enfileirados, isto é, sinais recebidos durante o tratamento de um outro sinal do mesmo tipo são bloqueados e enfileirados para posterior tratamento. Também, foi arquitetado que o diretório que dois processos *senders* não escrevem em um mesmo diretório. No exemplo acima, o *sender* de Maul deve escrever em um diretório X que é lido pelo *receiver* de Vader. Por sua vez, o processo *sender* de Vader deve escrever em um diretório diferente, Y, que é lido pelo *receiver* de Maul.

[3] http://www.gnu.org/software/libc/manual/html_node/Processes.html

4.1 Inicialização

A sequência de inicialização do sistema se dá nos seguintes passos. O processo *logger* deve ser o primeiro a ser iniciado, recebendo como argumento um diretório para salvar o log.

- Um usuário o *logger* dever ser iniciado. Seu *PID* deve ser salvo no arquivo ./logger.txt. O *logger* deve saber onde vai ser mantido os logs dos dois processos *chat.*

Após o *logger* ser iniciado, podemos iniciar o processos *chat*. O processo *chat* recebe como parâmetros um diretório temporário, um diretório para salvar arquivos, o nome do usuário, um diretório compartilhado de escrita e um diretório compartilhado de leitura. Os diretórios compartilhados são usados pelos dois processos da seguinte forma: O processo *sender* do usuário Alice tem como diretório de escrita o diretório de leitura do *receiver* de Bob. Por consequência, o diretório de escrita do processo *sender* de Bob escreve no diretório de leitura do *receiver* de Alice.

- Iniciar o processo *chat*
  O chat deve a) Usar a chamada **fork** ou **exec** para iniciar *receiver*. Este processo fará uso do diretório de salvar arquivos e do compartilhado;
  O chat deve b) Usar a chamada **fork** ou **exec** para iniciar *sender*. Este processo fará uso do diretório temporário e do compartilhado.

4.2 logger

Este processo deve apenas ler os arquivos de log locais dos dois *receivers* e montar um arquivo de log próprio com as *mensagens* trocadas **ordenadas pelo tempo**.

4.3 chat

O *chat* deve apenas iniciar os dois outros processos como descrito acima. Se em algum momento for recebido o sinal SIGUSR2 o chat deve repassar este sinal para os dois processos.

4.4 receiver

Assim que iniciado, este processo deve escrever seu *PID* no arquivo ./alice.txt ou ./bob.txt, dependendo do usuário que vai usar o processo. O processo deve ler o *PID* do *logger* do arquivo ./logger.txt. Se o arquivo de *PID* não existir o processo deve esperar até o mesmo ser criado. Cada processo *receiver* deve guardar um arquivo de log local. Este arquivo é o lido pelo processo *logger*.

Após estes passos, o funcionamento do receiver deve se dar todo de um laço principal que executa as seguintes passos:

1. Esperar sinal SIGUSR1.
2. Lê mensagens e arquivos do diretório compartilhado com o *sender* do outro usuário.

3. No caso de um arquivo recebido, move arquivo para o diretório de transferências.
4. Imprime mensagens na tela e em um log local. Mensagens de transferência devem ser impressas também. É essencial que as mensagens logadas tenham a data e hora no qual o evento ocorreu.
5. Notifica o *logger* utilizando o sinal SIGUSR1.
6. Notifica o *sender* utilizando o sinal SIGUSR1.
7. Volta para o passo inicial.

Se em algum momento for recebido o sinal SIGUSR2, o processo deve limpar seu diretório temporário, limpar o diretório compartilhado, apagar o arquivo de *PID* e finalizar. Esta operação também deve ser logada.

4.5 sender

Ao iniciar, este processo deve inicialmente ler o arquivo (./alice.txt ou ./bob.txt) com o *PID* do outro usuário para descobrir com que processo vai se comunicar. Se o arquivo não existir o processo deve esperar até o mesmo ser criado. Após este passo, o funcionamento também é dado através de um laço principal:

1. Lê mensagens de entrada do usuário. Transferências de arquivos deve ser feitas através da mensagem de controle "TRANSFER:arquivo", exemplo: "TRANSFER:/home/grad/biliu/prova.c".
   Mensagens devem ser salvas no diretório temporário e depois movidas para o diretório compartilhado com o *receiver* do outro usuário.
   Ao receber uma mensagem TRANSFER o processo deve copiar o arquivo indicado para o diretório temporário e depois mover o mesmo para o compartilhado.
2. Notifica o *receiver* utilizando o sinal SIGUSR1.
3. Espera sinal SIGUSR1.
4. Volta para o passo inicial.

Se em algum momento for recebido o sinal SIGUSR2, o processo deve limpar seu diretório temporário e finalizar.

5 Critérios de Avaliação

Cada aluno/dupla deve apresentar todos os arquivos com código utilizados no trabalho, bem como um relatório curto que deve conter uma descrição da arquitetura adotada e as principais decisões de projeto envolvidas, como o controle do tempo, as estruturas de dados utilizadas e decisões de implementação não documentadas nesta especificação. Serão atribuídos pontos para a execução correta do programa, para a organização e clareza do relatório e para a organização do código.

Como discutido anteriormente, os alunos não podem, em hipótese alguma, compartilhar ou tornar públicos quaisquer trechos do código de suas implementações. Afinal, vocês são alunos honestos diferente de Bob, Alice, Vader e Maul.

6 Submissão Eletrônica

Os trabalhos deverão ser entregues (juntamente com o relatório gerado) em um arquivo do tipo zip ou tar.gz através do LearnLoop. **O código fonte deve ser bem comentado.** Junto do código fonte, um arquivo `Makefile` deve ser incluído, de forma a facilitar a compilação do programa. **Não inclua arquivos objeto (.o) nem executáveis no seu arquivo de entrega.**

7 Observações Gerais

1. **Dúvidas:** esclarecimentos podem ser solicitados através do Learnloop sempre que possível. Use e-mail para os instrutores apenas se a questão exigir o envio de mais que uma linha de código. Tentaremos responder todas as perguntas em menos de 48 horas. Perguntas relacionadas à codificação do problema podem não ser retornadas.

2. Comece a fazer este trabalho logo, enquanto o problema está fresco na memória e o prazo para terminá-lo este tão longe quanto jamais poderá estar.

3. **Planeje para que seu código seja robusto. Tente prever falhas no processo.**

4. **Vão valer pontos clareza, indentação e comentários no programa, bem como a qualidade da documentação que o acompanhe.**

5. O trabalho deve ser desenvolvido de forma independente por cada aluno/dupla. Não é permitido discutir os aspectos do programa e soluções adotadas com outros alunos, e é terminantemente proibido compartilhar programas ou trecho de programas. **Tal comportamento poderá ser punido severamente.**

Última alteração: 9 de setembro de 2008