Sistemas Operacionais

Profª. Roberta Lima Gomes - email: soufes@gmail.com

1º Trabalho de Programação

Período: 2015/2

Data de Entrega: 26/10/2015 Composição dos Grupos: até 2 pessoas

Material a entregar

- Por email: enviar um email para **soufes@gmail.com** seguindo o seguinte formato:
  - Subject do email: "**Trabalho 1**"
  - Corpo do email: lista contendo os nomes completos dos componentes do grupo em ordem alfabética
  - Em anexo: um arquivo compactado com o seguinte nome "**nome-do-grupo.extensão**" (*ex: joao-maria-jose.rar*). Este arquivo deverá conter todos os arquivos (incluindo os *makefile*) criados com o código muito bem comentado.

Serão pontuados: clareza, indentação e comentários no programa.
Desconto por atraso: 1 ponto por dia

Descrição do Trabalho

Vocês deverão implementar uma nova shell na linguagen C, denominada `lsh` (*Lasy Shell*). Ao contrário de uma shell convencional, que é um interpretador de programas responsável por lançar processos de background ou forground a cada comando, a `lsh` tem um pequeno "problema":

- Sendo uma shell "preguiçosa", a cada comando que o usuário digita, ela não o executa imediatamente; ela vai enfileirando os comandos em um buffer, sem executá-los.
- O usuário, que vai perdendo a paciência, pode tentar "matar" a `lsh` por meio de um "Ctrl-c"; se isso acontecer, a `lsh` não deverá finalizar... mas para deixar o usuário mais calminho ela deverá executar a sequência de comandos digitados até então pelo usuário, esvaziando assim o buffer de comandos;
  Obs.1: Os comandos que são armazenados no buffer podem ser **comandos internos** (que não implicam na criação de novos processos, como o comando "cd") e **comandos externos** (que implicam na criação de novos processos para executar os executáveis correspondentes, como o comando "ls");
  Obs.2: Em se tratando de comandos externos, todos os processos criados pela lsh deverão rodar em background (*i.e.*: não poderão ter acesso de leitura no Terminal e não receberão nenhum Sinal gerado pelo usuário por meio de comandos "Ctrl-..." no Terminal). Além disso, como herança dessa shell, esses processos deverão **Ignorar** o sinal SIGINT.
- Com isso, enquanto a `lsh` tiver processos filhos (diretos) em execução (*i.e.* vivos!), sempre que o usuário digitar novamente o comando "Ctrl-c", a `lsh` deverá exibir a mensagem "Não posso morrer... sou mãe de família!!!". Além disso, se existirem novos comandos enfileirados no buffer, ela deverá executar a sequência de comandos, esvaziando novamente o buffer. Mas se ao digitar o comando "Ctrl-c" a `lsh` não tiver mais nenhum processo filho (direto), E CASO NÃO SEJA A PRIMEIRA VEZ QUE O USUÁRIO TENHA DIGITADO O comando "Ctrl-c", então ela deverá exibir a mensagem "Ok... você venceu! Adeus!" finalizando sua execução em seguida (mesmo que existem comandos no buffer!).

Abaixo temos um exemplo de sequência de execução da lsh:

```
lsh> ls -l          //comando externo
lsh> cd ..          //comando interno
lsh> firefox        //comando externo
lsh> ps             //comando externo
lsh>
                    //usuário digita Ctrl-c
                    //a lsh cria processos em background para executar o
                    //comando “ls -l”, executa o comando “cd”, cria processos
                    // em background para executar os comandos “firefox” e “ps”
                    //(nessa ordem)

-rwxrwxr-x  1 roberta ...               //saída do comando “ls -l”
...
(process:13980): GLib-CRITICAL **       //saída do comando “firefox”
...
  PID TTY          TIME CMD             //saída do comando “ps”
...
...

lsh> ps -l          //comando externo
lsh>
                    //usuário digita Ctrl-c
lsh> Não posso morrer... sou mãe de família!!!
F S   UID   PID  PPID ...               //saída do comando “ps -l”
...

lsh>                //o usuário fecha o firefox... agora lsh não tem mais filhos
lsh> xcalc
lsh>
                     //usuário digita Ctrl-c
lsh> “Ok... você venceu! Adeus!”
                     //FIM DA lsh!!
```

Linguagem da `lsh`

A linguagem compreendida pela `lsh` é bem simples. Cada sequência de caracteres diferentes de espaço é considerada um termo. Termos podem ser:

(i) comandos internos da shell,
(ii) nomes de programas que devem ser executados (e seus argumentos),

- *Comandos internos da shell* são as sequências de caracteres que devem sempre ser executadas pela própria shell e não resultam na criação de um novo processo. Na `lsh` as operações internas são: `cd`, `wait` e `exit`. Essas operações devem sempre terminar com um sinal de fim de linha (*return*) e devem ser entradas logo em seguida ao *prompt* (isto é, devem sempre ser entradas como linhas separadas de quaisquer outros comandos).

`cd`: Muda o diretório corrente da *shell*. Isso terá impacto sobre os arquivos visíveis sem um caminho completo (*path*).

`wait`: Faz com que a *shell* libere todos os processos filhos que estejam no estado “Zombie” antes de exibir um novo *prompt*. Cada processo que seja “encontrado” durante um `wait` deve ser informado por meio de uma mensagem na linha de comando. Caso não haja mais processos no estado “Zombie”, uma mensagem a respeito deve ser exibida e `lsh` deve continuar sua execução.

`exit`: Este comando permite terminar propriamente a operação da *shell*. Ele faz com que todos os seus herdeiros vivos (herdeiros diretos) morram também ... e a `lsh` só deve morrer após todos eles terem sido "liberados" do estado "Zombie".

- *Programas a serem executados* são identificados pelo nome do seu arquivo executável e podem ser seguidos por um número máximo de cinco argumentos (parâmetros que serão passados ao programa por meio do vetor `argv[]`). Na ocorrência de um "Ctrl-c", os processos devem ser criados e executados em backgroud e a *shell* deve continuar sua execução. Isso significa retornar imediatamente ao *prompt*.

Dicas Técnicas

Este trabalho exercita as principais funções relacionadas ao controle de processo, como `fork`, `execvp`, `waitpid`, `chdir`, e Sinais, entre outras. Certifique-se de consultar as páginas de manual a respeito para obter detalhes sobre os parâmetros usados, valores de retorno, condições de erro, etc (além dos slides da aula sobre SVCs no UNIX).

Outras funções que podem ser úteis são aquelas de manipulação de strings para tratar os comandos lidos da entrada. Há basicamente duas opções principais: pode-se usar `scanf("%s")`, que vai retornar cada sequência de caracteres delimitada por espaços, ou usar `fgets` para ler uma linha de cada vez para a memória e aí fazer o processamento de seu conteúdo, seja manipulando diretamente os caracteres do vetor resultante ou usando funções como `strtok`.

Ao consultar o manual, notem que as páginas de manual do sistema (acessíveis pelo comando `man`) são agrupadas em seções numeradas. A seção 1 corresponde a programas utilitários (comandos), a seção 2 corresponde às chamadas do sistema e a seção 3 às funções da biblioteca padrão. Em alguns casos, pode haver um comando com o mesmo nome da função que você procura e a página errada é exibida. Isso pode ser corrigido colocando-se o número da seção desejada antes da função, por exemplo, "`man 2 fork`". Na dúvida se uma função é da biblioteca ou do sistema, experimente tanto 2 quanto 3. O número da seção que está sendo usada aparece no topo da página do manual.

Verificação de erros

Muitos problemas podem ocorrer a cada chamada de uma função da biblioteca ou do sistema. Certifique-se de testar cada valor de retorno das funções e, em muitos casos, verificar também o valor do erro, caso ele ocorra. Isso é essencial, por exemplo, no uso da chamada `wait`. Além disso, certifique-se de verificar erros no formato dos comandos, no nome dos programas a serem executados, etc. Um tratamento mais detalhado desses elementos da linguagem é normalmente discutido na disciplina de compiladores ou linguagens de programação, mas a linguagem usada neste trabalho foi simplificada a fim de não exigir técnicas mais sofisticadas para seu processamento, mas você deve fazer um tratamento mínimo.

Bibliografia Extra: Kay A. Robbins, Steven Robbins, *UNIX Systems Programming: Communication, Concurrency and Threads, 2nd Edition* (Cap 1-3).