Projeto 2: Gerência de Processos

48.255-2 – Laboratório de Sistemas Operacionais

Igor Racca	511382
Lucas da Rocha Pereira	379948

Objetivos

Modificar o interpretador de comandos elementar descrito em sala de aula, implementando as seguintes funcionalidades:

- Permitir que os comandos executados recebam argumentos. Deve ser possível, por exemplo, executar os comandos:

  > kill -9 <pid>

  > ls -l

- Adicionar suporte para executar comandos em segundo plano. Deve ser usada a sintaxe padrão do interpretador Unix: &. Deve ser possível, por exemplo, executar os comandos:

  > gnome-calculator &

  > gedit file1.txt &

- Fazer redireção da entrada e saída padrão, para permitir que um programa leia e escreva em um arquivo como a sua entrada e saída padrão. Deve ser usada a sintaxe padrão do interpretador Unix: < para redirecionar a

entrada padrão e > para redirecionar a saída padrão. Deve ser possível, por exemplo, executar os comandos:

> ps > arquivos.txt

> sort -r < arquivos.txt

Modificações

O interpretador de comandos elementar descrito em sala de aula, aceitava comandos simples e sem parâmetros. Foram implementadas 5 sub-rotinas na função *main* (*Figura 1*). As funções *init* e *clear* são apenas para inicialização de variáveis, alocação e desalocação de memória. As três sub-rotinas mais importantes serão descritas detalhadamente a seguir e no código fonte estão devidamente comentadas. Todas as funções de biblioteca recomendadas em sala de aula foram utilizadas, as mesmas serão também descritas a seguir.

```
int main() {
  char cmd[MAX];
  char **argv;
  int len, bgProcesses=0;
  int fileIn, fileOut, background;

  while (1) {
    // read the command
    printf("> ");
    scanf(" %[^\n]",cmd);

    // init all variables
    argv = init(&len, &background, &fileIn, &fileOut);

    // split command into an array of strings
    parse(argv, cmd, &len);

    // get the index of '&', '>' and/or '<' in the array
    checkArgs(argv, len, &background, &fileIn, &fileOut);

    // execute the command correctly
    exec(argv, &bgProcesses, background, fileIn, fileOut);

    // free the array of arguments
    clear(argv);
  }
}
```

Figura 1

I. Execução de comandos com argumentos

O interpretador de comandos inicial aceitava apenas um comando sem argumentos. Para o interpretador passar a aceitar argumentos, passamos a ler o comando com os argumentos até o final da linha e os armazenamos em uma cadeia de caracteres. Na função *parse*, esta cadeia de caracteres é dividida em cada espaço e armazenada em um vetor de cadeia de caracteres. Utilizamos a função *strtok*, como aconselhado pelo professor.

Na função *exec*, o comando inicial *execlp* foi substituído, também aconselhado pelo professor, pelo comando *execvp*, que também permite o lançamento da execução de um programa externo ao processo, porém aceita um vetor que representa a lista de argumentos para o programa executado.

II. Execução de comandos em segundo plano

Após obter o vetor com argumentos, através da função *checkArgs*, percorremos o vetor para encontrar o caracter '&', que de acordo com a sintaxe padrão do interpretador Unix, significa que o programa deve ser roda em segundo plano, a posição deste caractere no vetor, é substituída por *NULL* para sinalizar o fim do vetor para o comando *execvp*.

Ao encontrar este caractere, setamos uma *flag* que indica que o programa deve ser rodado em segundo plano. Esta *flag* é passada como parâmetro para a função *exec*, que é verificada antes do lançamento de execução. Caso verdadeira, o fragmento de código executado pelo processo pai não aguarda até o fim da execução do processo filho, ao contrário do interpretador de comandos inicial, bem como o codigo atual quando a flag é negativa, que através do comando *waitpid*, faz com que o processo pai aguarde até o fim da execução do processo filho.

Além disso, para tentar se aproximar mais do interpretador de comandos do Unix, quando o comando deve ser executado em segundo plano, nosso interpretador também imprime a quantidade de processos rodando em segundo plano e o pid do novo processo, como mostra a *Figura 2*:

Figura 2

III. Redirecionamento da entrada e saída padrão

Assim como na última funcionalidade, após obtermos o vetor com argumentos, através da função *checkArgs*, percorremos o vetor a fim de encontrar os caracteres que de acordo com a sintaxe do interpretador Unix, simbolizam o redirecionamento da entrada e saída padrão, '<' e '>', respectivamente.

Quando encontrado(s), a função *checkArgs*, salva sua posição no vetor de argumentos e seta sua posição no vetor para *NULL*, sinalizando novamente, o fim do vetor para o comando *execvp*. Antes do lançamento de execução, o fluxo de entrada e/ou saída é associado ao arquivo (que se encontra na posição do caractere, somado de um). Para implementar essa redireção, utilizamos o comando *freopen*, também sugerido pelo professor.

Dificuldades

As dificuldades encontradas neste projeto foram as seguintes:

- Relembrar manipulação de *strings* em C
- Entender o comando *strtok* e transformar a entrada no vetor de cadeia de caracteres
- Entender o funcionamento dos comandos *fork, execvp* e *freopen*