Relatório do projeto 2 – Gerência de processos

Lucas da Rocha Pereira – 379948
Thiago Martins de Jesus – 380385

Objetivo:

Desenvolver um shell em linguagem C para interpretar os mais diversos comandos e chamadas de sistema UNIX.

Análise da implementação e dos resultados:

De início nos foi fornecido um código fonte em linguagem C que processava comandos simples. A ideia era incrementar esse algoritmo de forma que ele conseguisse executar mais comandos. As funcionalidades e como elas foram implementadas estão a seguir:

Suporte a argumentos: para que o programa suporte comandos com argumentos, foi necessário primeiramente mudar a maneira como era lido o comando, passando a ler a linha inteira digitada pelo usuário. Foi utilizado o comando read_line, desenvolvido pelo Professor Tiago na disciplina de ALG2 em 2011, que garante que a linha inteira será lida, sem precisar se preocupar com a escolha da entrada padrão, como ocorre na função fgets. Com a linha devidamente lida, foi necessário reproduzir o comportamento do parâmetro *argv[] de uma função main, ou seja, separar os argumentos e guardar o endereço de cada entrada em uma posição do vetor de ponteiros. Ao invés de utilizar a função strtok, decidimos utilizar aritmética de ponteiros em conjunto com um while, verificando cada caractere do comando e, caso esse caractere seja um espaço, é salvo mais uma entrada no *argv[]. Salvo o comando e os argumentos devidamente, é chamado o comando execvp, que tem como argumentos o comando principal e os argumentos, tornando ele dentre todos os comandos exec, o mais adequado para essa ocasião.

Execução em segundo plano: esse foi o mais fácil de ser implementado. Foi observado por nós que a diferença de um processo normal para um em segundo plano é que o processo pai não espera o filho terminar, ou seja, ele simplesmente ignora o comando wait e continua a execução. Então, para que o caractere ‘&’ fosse reconhecido, foi implementado na função que separa os argumentos um if que identificava o caractere e mudava o valor de retorno da função para 1, fazendo com que a função que trata da execução ignorasse o comando wait e preparasse para a leitura da próxima instrução.

Redireção da entrada e saída padrão: todo processo, assim que é criado em ambiente UNIX, cria uma pasta no diretório proc, com todas as informações sobre ele, inclusive os descritores de arquivos de entrada e saída. Para que a entrada e saída fossem alteradas, era necessário alterar esses descritores, porém assim que o processo fosse concluído eles teriam que voltar ao estado original para que o shell continuasse a executar os demais comandos e exibi-los na tela. Logo no começo do programa, é guardado os descritores padrão para serem restaurados ao fim de cada

processo, para isso foi utilizado a chamada de sistema dup, que atribui um valor inteiro ao descritor associado ao processo. Na função de separação de argumentos, foi inserido dois if's que identificam os caracteres simbolizadores de entrada (<) e saída (>). Identificado algum deles, ou os dois, inicia-se um laço de repetição que pega todos os caracteres seguintes até o espaço ou o fim da linha, porém não insere a entrada dele no vetor de argumento. Com o nome do arquivo de destino, é chamado a função dup2, que é muito parecida com a dup com o adicional de mudar o descritor para um arquivo selecionado. Após a mudança, o execvp é executado e a saída ou entrada é desviada corretamente.

No final da implementação foram encontrados alguns problemas como por exemplo quando a saída ou entrada era desviada mais de uma vez, o vetor guardava lixo e o descritor não era alterado corretamente. Problema que foi resolvido inicializando os vetores através da função memset cada vez que a função identificava '<'ou '>'.

Lembrando que comentários específicos das funções podem ser encontrados no código.