UNIVERSIDADE FEDERAL DE SÃO CARLOS
Bacharelado em Ciência da Computação
Laboratório de Sistemas Operacionais
Professor Gustavo Maciel Dias Vieira
***Campus* Sorocaba**

Projeto 2
Organização de Sistemas Operacionais

Daniel Ramos Miola 438340
Giulianno Raphael Sbrugnera 408093

Sorocaba
2013

1. Introdução

O Projeto 2 tem por objetivo explorar mais a fundo a gerência de projetos no sistema *Unix*, as quais são auxiliadas por chamadas de sistemas. Dessa maneira, foi incrementado o interpretador de comandos elementar disponibilizado, adicionando as seguintes funcionalidades:

- comandos executados podem receber argumentos.
- comandos podem ser executados em segundo plano
- programas podem ler e escrever em um arquivo como a sua entrada e saída padrão.

Neste documento será descrito como essas funcionalidades do interpretador de comandos foram criadas, quais serviços do sistema operacional foram utilizados e de que maneira.

2. Discussão e Resultados

Ao todo foram 3 funcionalidades implementadas. Cada uma delas é desenvolvida inicialmente da mesma maneira, descritas logo abaixo.

O comando, assim como seus argumentos, são lidos via terminal e armazenados em uma *string*. Com o auxílio da função *strtok*, o comando e seus argumentos são separados e armazenados em uma matriz de *strings* para utilização posterior. Só não são armazenados caracteres especiais que definem redirecionamento de entrada e saída e & indicando a execução de um comando em segundo plano. É utilizada então a chamada de sistema ***fork***, onde é criado um processo filho como uma cópia de si mesmo. O valor de retorno dessa chamada de sistema é guardado, sendo possível verificar através dele quem é o processo pai e quem é o processo filho.
A partir desse ponto, cada funcionalidade possui suas peculiaridades que serão explicadas caso a caso.

2.1. Permitir que os comandos executados recebam argumentos.

Serviços do sistema operacional utilizados:

- *fork*
- *waitpid*
- *execvp*

Fork faz com que o processo crie uma cópia de si mesmo. Nesse caso, o processo filho é quem vai executar o comando especificado. Há conhecimento sobre quem é o processo pai e quem é o processo filho pois é guardado o valor de retorno de ***fork***.

Waitpid espera o término de um processo. Nesse caso, é esperado até que o processo filho se finalize por completo para retornar ao processo pai.

Execvp é o responsável por chamar o processo equivalente ao comando. Seu primeiro argumento é o nome do comando e o segundo é a lista de argumentos do comando terminada por *NULL*. Caso o processo de nome especificado não exista, é gerado um erro.

```
//aloca e retira comando
token = strtok(linhaComando," ");
arg = (char**)malloc((sizeof(char*))*20);
arg[0] = token;

//separa argumentos retirando redirecionamentos e wait
n = 0;
while (token != NULL || n >= 19)
{
    n++;
    token = strtok(NULL, " ");
    if(token == NULL)
        continue;
    if(!strcmp(token,"<")){
       token = strtok(NULL," ");
       in = token;
       n--;
    }else if(!strcmp(token,">")){
        token = strtok(NULL," ");
        out = token;
        n--;
    }else if(!strcmp(token,"&")){
        wait = 1;
        n--;
    }else{
        arg[n] = token;
    }
}

//seta ultimo argumento nulo
arg[n+1] = NULL;
```

Imagem 1. Trecho de código responsável pela separação dos argumentos.

O código acima é responsável por separar os argumentos e guarda-los em um vetor de

ponteiros que será passado como argumento de *execvp.* Primeiramente ele separa o comando em si e depois todos os argumentos restantes caso existam. As strings de caminhos para o redirecionamento de entrada e saida padrão ja são separadas aqui para nao gerar retrabalho assim como a verificação de um caracter para a execução em segundo plano. Por fim, setamos um ultimo argumento como *NULL* que será o criterio de parada para *execvp.*

2.2. Execução de comandos em segundo plano.

Serviços do sistema operacional utilizados:

- *fork*
- *execvp*

Fork e ***execvp*** são utilizados da mesma forma da funcionalidade 1.
Como é desejada a execução de comandos em segundo plano, a chamada de sistema ***waitpid*** não é utilizada. Dessa maneira, o processo pai não espera o término do processo filho, o que garante a execução em segundo plano de comandos.

```
if (pid)
{
    if(!wait)
        waitpid(pid, NULL, 0);
}
```

Imagem 2. Trecho responsável pela execução em segundo plano.

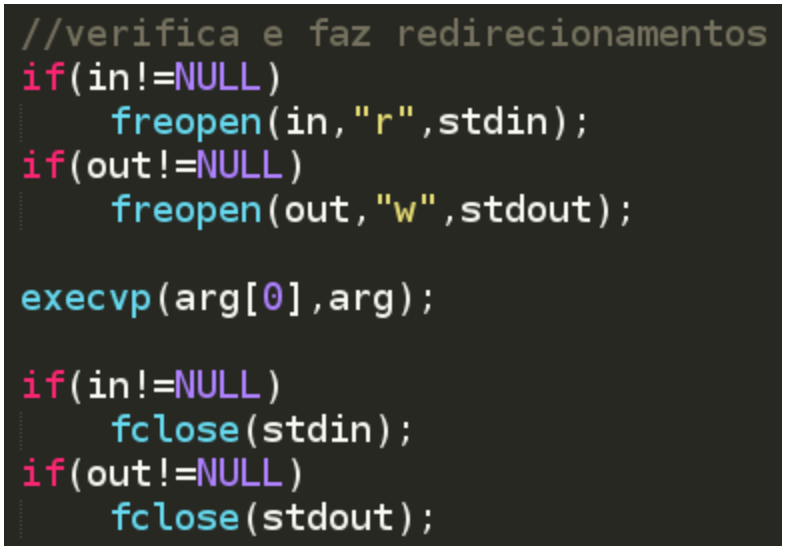
O trecho acima apenas verifica uma flag gerada na separação dos argumentos onde o comando *waitpid* só será executado caso não seja solicitada a execução em segundo plano.

2.3. Redireção da entrada e saída padrão.

Serviços do sistema operacional utilizados:

- *fork*
- *waitpid*
- *execvp*

Fork, ***waitpid*** e ***execvp*** são utilizados da mesma forma da funcionalidade 1.
De forma auxiliar, é utilizada a função *freopen* para a redireção de entrada e saída, lendo de um arquivo e escrevendo em um arquivo de acordo com o comando. Para garantir a integridade dos dados, os arquivos são fechados ao final.



```
//verifica e faz redirecionamentos
if(in!=NULL)
    freopen(in,"r",stdin);
if(out!=NULL)
    freopen(out,"w",stdout);

execvp(arg[0],arg);

if(in!=NULL)
    fclose(stdin);
if(out!=NULL)
    fclose(stdout);
```

Imagem 3. Parte responsável pelo redirecionamento de entrada e saida padrão.

Acima são executados os redirecionamentos de entrada e saida padrão para os arquivos do comando(caso existam) antes da execução de *execvp.* Logo após a execução do comando a entrada e saida padrão são fechados.

3. Dificuldades encontradas na realização

O principal ponto de dificuldade na realização do projeto foi a separação do comando a ser executado em um vetor de ponteiros utilizando a função *strtok.* O restante da implementação se mostrou com uma lógica bem simples associada aos comandos de chamada de sistema.

4. Conlusão

Após a realização do projeto ficou claro o poder das funções de chamada de sistema associadas a gerência de processos. Toda a implementação apresentou uma solução simples com as funções de chamada de forma que, é possivel criar funcionalidades de forma rapida para um interpretador de comandos Unix.

5. Referências Bibliográficas

[1] Slides do curso Laboratório de Sistemas Operacionais – UFSCar Sorocaba

[2] Slides do curso Sistemas Operacionais – UFSCar Sorocaba