TRABALHO / RESOLUÇÃO:

Na função main(), o parâmetro “argc” (contador de argumentos) é uma variável do tipo inteiro que indica quantos argumentos foram inseridos na linha de comando quando o programa foi iniciado. O segundo parâmetro “argv” (vetor de argumentos) é um vetor de strings no qual cada string deste vetor é um dos argumentos introduzidos na linha de comando.

Para o programa testar se o número de argumentos introduzidos na linha de comando é correto, testa-se se “argc” é diferente de ‘2’. Se for diferente, então quer dizer que não foram introduzidos precisamente dois argumentos. Por isso o programa vai emitir uma mensagem de erro porque o número de argumentos está incorreto e o programa vai terminar. Para o programa ser executado corretamente na linha de comandos, só podem ser introduzidos estes dois argumentos na linha de comandos: 1º argumento “/.gsn” e o 2º argumento: “n”. Em que “n” é um número do tipo inteiro positivo.

No programa, a função “atoi()” é utilizada para converter o segundo argumento introduzido na linha de comandos, que é o argumento “n”, em um valor inteiro. Esse valor inteiro é então comparado ao valor “0”. Se o valor do argumento “n” for menor ou igual a zero, vai emitir uma mensagem de erro e o programa vai terminar.

O Processo A é o programa “gsn.c”, e o processo-pai deste processo será a bash (terminal/interpretador de comandos), onde este programa foi executado pelo comando “./gsn n”.

O programa “gsn.c” executa a sua tarefa em 3 passos distintos, criando os processos B, C e D, um de cada vez, para executarem cada passo do programa. Para criar estes processos fazemos recurso à função de sistema “fork()” que cria uma duplicação do mesmo processo, criando assim um processo-filho, mas que contem a mesma imagem. Então por isso temos de substituir cada imagem de cada processo-filho, B, C, e D para os comandos “head, hexdump e sort” nesta ordem para executar os 3 primeiros passos do programa A.

Depois de ter sido verificado os argumentos introduzidos, o programa imprime no terminal os números de identificação do Processo A e do seu processo-pai.

Começando a executar o primeiro passo, que é o comando “head”, utiliza-se a função “fork()” para criar uma cópia do Processo A e altera-se a sua imagem para o comando “head”, e assim é criado o Processo B.

A variável “pidB” é inicializada, e é lhe atribuída o valor de retorno da função “fork()”. Se o valor de retorno for maior que zero ou igual a -1, quer dizer que a função “fork()” falhou na criação do processo-filho. Nesse caso, a função “perror()” é utilizada para indicar que houve um erro na criação do processo. Se então essa variável “pidB” for igual a zero, ou seja, a função “fork()” retornou o valor “0”, quer dizer que o Processo B foi criado com sucesso.

Antes de ser substituida a imagem do Processo B para o comando “head”, a saída padrão stdout é redirecionada para o ficheiro “tmp.bin”, usando a função freopen(). Esta função é usada para associar um arquivo ao fluxo de entrada/saida padrão (stdin/stdout) do programa. Quer seja para ler (read/”r”) ou para escrever (write/”w”).

A substituição de imagem nos 3 processos criados teve que ser feita obrigatoriamente utilizando 3 funções de sistema diferentes da família exec() (família de funções de substituição de imagem do processo), por isso no Processo B utilizei a função “execl()”, no Processo C utilizei a função “execvp()” e para o Processo D utilizei a função “execv()”.

Como os processos B, C , e D devem ser criados um de cada vez para assim executar um passo de cada vez do programa, antes de o Processo C ser criado, após ser criado o Processo B, o Processo A usa a função waitpid() para aguardar pela terminação do Processo B.

A estrutura de realização para a criação dos Processos B, C e D foi sempre a mesma, alterando apenas em cada uma na função de sistema exec(), usada para a substituição de imagem.

No Processo B, a saída do programa é redirecionada para o ficheiro “tmp.bin”, e é executado o comando “head” que, usando /dev/urandom, gera dados aleatórios e armazena-os neste ficheiro binário.

No Processo C, a entrada do programa é redirecionada para o ficheiro “tmp.bin” e a saída é redirecionada para o ficheiro “tmp.txt”. É então executado o comando “hexdump” que lista os bytes do ficheiro tmp.bin em decimal e os imprime no ficheiro “tmp.txt”.

Por fim no Processo D, a entrada do programa é redirecionada para o ficheiro “tmp.txt” e é executado o comando “sort” que ordena os números(linhas) do ficheiro tmp.txt.

A realização deste E-Fólio permitiu-me aprender sobre os comandos head, hexdump e sort, assim como o funcionamento do arquivo gerador de números aleatórios “*dev/*urandom”.

Não tive quaisquer erros de compilação. O programa foi compilado com o comando: “gcc -Wall -o gsn gsn.c” no sistema operativo Linux Mint.

Alguns exemplos dos resultados que obtive com a execução do programa “gsn.c”:

```
./gsn 8
Processo A: PID=12881 PPID=7610
Processo B: PID=12882 PPID=12881
Processo C: PID=12883 PPID=12881
Processo D: PID=12884 PPID=12881
-125
-98
-53
-9
40
68
99
110
```

```
./gsn 5
Processo A: PID=12676 PPID=7610
Processo B: PID=12677 PPID=12676
Processo C: PID=12678 PPID=12676
Processo D: PID=12679 PPID=12676
-48
62
65
93
123
```

```
./gsn 4
Processo A: PID=17948 PPID=7610
Processo B: PID=17949 PPID=17948
Processo C: PID=17950 PPID=17948
Processo D: PID=17951 PPID=17948
-84
25
29
118
```

```
jose@jose-MS-7817:~/Documentos/Uni
./gsn 3
Processo A: PID=12672 PPID=7610
Processo B: PID=12673 PPID=12672
Processo C: PID=12674 PPID=12672
Processo D: PID=12675 PPID=12672
-128
56
83
```

FIM