Informe Tarea 1 de Sistemas Operativos

de Carlos Sebastián Salinas Pereira (2023441619)

GitHub: ***https://github.com/CSalinasP/SO_Tarea1.git***

Parte 1:

Al principio he querido hacer mi Shell en ***C****, pero he tenido muchos problemas respecto a la implementación de los comandos con pipes, los comandos sin pipes en la terminal funcionaban bien, pero al agregarle los pipes, no mostraban nada en pantalla.*

He estado buscando el error por semanas y al final me he rendido, asumiendo que es un problema con mi PC ya que uso Windows con un entorno Linux WSL instalado, así que decidí hacer la Shell en ***C++.***

Al aprender los nuevos comandos de ***C++*** *que tenía que usar, he terminado la tarea mucho más rápido y fácil ya que apenas he tenido errores.*

Considerando que ***C++*** *no usa las mismas funciones que vimos en* ***C*** *y en la ayudantía, voy a mencionar las variables y funciones más importantes que usé en* ***C++*** *que no están en* ***C:***

1) ***getline:*** *Se usa para leer el comando escrito en la terminal, y también para dividir en tokens los comandos (muchísimo mejor que* ***strtok*** *en* ***C****).*
2) ***args_c:*** *Vector que guarda los tokens originales de los comandos, pero en formato* ***C,*** *ya que, aunque estemos en* ***C++****, para usar* ***execvp*** *necesitamos que los strings estén en el formato* ***C*** *y no en el de* ***C++****.*
3) ***stringstream:*** *Crea un stream del string que contiene el comando para realizar el "parseo" en tokens.*

Ahora, mencionaré las 4 funciones principales del programa respecto a la parte 1:

1) ***main:*** *Encargado de simular el Shell en sí, al leer una entrada de comando, usa parseLine para parsearlo y llama a "executeNormalCommand" o "executePipelineCommand" dependiendo si se ingresaron varios comandos con pipes o no.*

2) ***parseLine:*** *Encargado de "parsear" la línea de comando en tokens usando "getline" y "stringstream" anteriormente mencionados.*

3) ***executeNormalCommand:*** *Encargado de ejecutar un comando que no tiene pipes usando "fork" para que el proceso hijo ejecute el comando que ya ha sido "parseado".*

4) ***executePipelineCommand:*** *Encargado de ejecutar varios comandos unidos con pipes, que aparte de usar "fork", usa las funciones y variables "pipe", "pipe_fd", "dup2", etc., para realizar la comunicación de entrada y salida de información entre los comandos.*

Las 3 últimas funciones se pueden encontrar en "Funciones.cpp", que es un archivo aparte de "main.cpp".

Parte 2:

Ahora mencionaré las funciones principales respecto a la parte 2:

1) ***main:*** *Encargado de simular el Shell, llamando a las funciones "executeMiprof", "executeMiprofMaxTime" o "executeMiprofEjecsave" dependiendo de lo que quiere el usuario.*

2) ***executeMiprof:*** *Es parecido a "executeNormalCommand" pero muestra en pantalla el tiempo de ejecución del usuario, sistema, real y el "máximo resident set size".*

3) ***executeMiprofMaxTime:*** *Parecido a "executeMiprof" pero se termina la ejecución del comando si se excede el tiempo máximo insertado.*

4) ***executeMiprofEjecsave:*** *También parecido a "executeMiprof" pero envez de mostrar los resultados en la terminal, los guarda en un archivo txt.*

Cabe mencionar que la parte 1 y 2 están implementadas en el mismo ejecutable "MiShell.exe", en el README.md estarán las instrucciones sobre cómo se usa el programa.

*Por último, mostraré el resultado de usar "***miprof ejec sort –r test1.txt***" y "***miprof ejec sort –r test2.txt***", considerando que test1 y test2 tienen distinta cantidad de caracteres:*

```
mishell$ miprof ejec sort -r test1.txt
Comando usado:
sort -r test1.txt
Respuesta:
ffffffffffffff
eeeeeeeeeeeeee
ddddddddddddd
ccccccccccccc
bbbbbbbbbbbbbb
aaaaaaaaaaaaaa
Tiempo real: 0.020046 segundos
Tiempo de usuario: 0.00782 segundos
Tiempo de sistema: 0.01173 segundos
Máximo resident set size: 3200 KB
```

```
mishell$ miprof ejec sort -r test2.txt
Comando usado:
sort -r test2.txt
Respuesta:
kkkkkkkkkkkkkk
jjjjjjjjjjjjjj
iiiiiiiiiiiiii
hhhhhhhhhhhhhh
gggggggggggggg
ffffffffffffff
eeeeeeeeeeeeee
ddddddddddddd
ccccccccccccc
bbbbbbbbbbbbbb
aaaaaaaaaaaaaa
Tiempo real: 0.013581 segundos
Tiempo de usuario: 0.00782 segundos
Tiempo de sistema: 0.023942 segundos
Máximo resident set size: 3200 KB
```

Podemos notar que el tiempo real del archivo pequeño es mayor al del archivo grande, pero el tiempo del sistema fue menor, mientras que el tiempo del usuario fue el mismo.

El tiempo que demuestra que test2.txt es más grande, es el tiempo del sistema, mientras que el tiempo real puede variar en cada ejecución.