CSE312 – Operating Systems
REPORT - Makeup

Furkan ÖZEV
161044036

General Structure:

I wrote and customized syscalls according to the needs of the processes.
In syscall.cpp, I did not use any syscall other than the syscalls I wrote.
When Shell.asm runs, It receives a command string from the user with "READ_COMMAND_SYSCALL".
Then, this command is parsed to determine the processes(like sort, search etc.) and information(like stdin, stdout, pipe etc.) of the processes.
If there is no error in the command or an error occurred while creating the processes, Then the determined processes are added to the process table.
These processes are run. When the processes are finished, Shell returns to the process and receives a new command.
If an error has occurred, it prints the error message and waits for the command to enter again.
If the "exit" command is entered into the shell, the shell ends.

Piping, Redirection, Standard Input, Standard Output, Screen Input and Screen Output:

Some information about the process is kept in the process table. One of these are stdin and stdout information.
When the read and write syscalls are called, it understands where to read or write by looking at this information and takes the necessary action accordingly.

So, the main purpose here is to determine stdin and stdout.
The command received by the shell process is parsed according to certain tokens.
These are:
Redirectiom input: ' < '
Redirection output: ' > '
Pipe: ' | '
Background: “&”

So, stdin and stdout will determine with these tokens.

SCREEN:
If the process is not piping and has not received a filename as an argument(has no redirection), stdin and stdout will be **“screen”**.
Like “ls”, “random”, “sort”
So, if the process needs an input, it will take it off the screen, or if it wants to print something, it will print on the screen.

REDIRECTION:
If the process has received a filename as an argument (has redirection), stdin and stdout are wiil be these filenames.
It understands whether there is redirection with the tokens '<' or '>'.

If there is '<' token, it satisfy redirection input.
First, It parse filename. Then, stdin will be **input filename**.
Like, sort < 1.txt (stdin will be 1.txt)

If there is '>' token, it satisfy redirection output.
First, It parse filename. Then, stdout will be **output filename**.
Like, sort > 1.txt (stdout will be 1.txt)

Both can be together
Parses files according to tokens and stds change.
Like, sort < 1.txt > 2.txt or sort > 2.txt < 1.txt (stdin: 1.txt and stdout: 2.txt)

PIPING:
It understands whether there is piping with the tokens ' | ' .
If the process is piping, It creates 1 special pipe file.
This file is special for 2 processes that are pipes.
While the first process' stdout is set as this pipe file, the second process' stdin is set as this pipe file.

Like “random | sort”
stdout of the random process will be **"pipecountx.txt".**
stdin of the sort process will be **"pipecountx.txt".**
They are same file, therefore, 2 processes will be able to communicate over this file.

If tokens are used properly, It can run multiple states via a single command.
For example:
1. “random | sort | search”
2. “ls & random | sort > 1.txt”
3. “type a.txt” or “type a.txt > b.txt” or “type a.txt > b.txt &” or “type a.txt > b.txt & ls”

In this way, many commands can be produced.
As long as these commands are given properly, they will work without any problems.
If there is an error in the command, the error message is printed on the screen.
Since homework pdf is not an example of more than one case working together, I based the terminal.

Some of the wrong commands are as follows:
1. "ls& | ls” or "type a.txt & | ls" etc. ('|' should not come immediately after the '&' token. Because background task and piping are not possible for the same process. But “ls | ls&” or “ls | type a.txt&” etc. possible commands)
2. “ random > ” or “ sort < ” or “ls | “ (If missing commands are entered, an error message is issued.)
3. “type “ (If the parameter is required, an error message is given if the parameter is not entered.)
4. If no input file or parameter file is available, an error message is printed.

BACKGROUND:
There are a few things to consider for background task:
1. Background task can take stdin and stdout as follows:
"ls > 1.txt &" (stdin: 1.txt) or "sort < 1.txt > 2.txt &" (stdin: 1.txt, stdout: 2.txt)
2. If you misuse it, it will not fail, but stdin and stdout will not change. I made this part based on the terminal. It works like this in the ubuntu terminal.
"ls& > 1.txt" (stdin: screen) or "sort& < 1.txt > 2.txt" (stdin: screen, stdout: screen)
3. Multiple processes can run with the background task.
"type a.txt& ls& random | sort > 2.txt" or "type a.txt > b.txt & random | sort | search& ls"

SYSTEMCALLS:

READ_COMMAND_SYSCALL:
This syscall is used by the shell.asm.
A command is received from the user and parsed in this function.
While the command is being parsed, the processes(like sort, search etc.) and information of the processes(like input, output, pipe, background task, redirection, parameter etc.) are determined.
Required files are opened here. stdin and stdout are determined here.
Processes are loaded into the spim here.
Processes are added to the process table with their information.
Then, swapProcess () function is called to run the processes.
It does not take parameters, but if any errors occur, it reports shell.asm with R [3] = -1.

GENERATE_RANDOM_NUMBER:
This syscall is used by the random.asm .
It does not take parameters, It generates a random number and writes it to R [V0].
Then, swapProcess () function is called to run next the processes for multi programming or parallel running.

READ_INT_INPUT:
This syscall is used by the sort.asm and search.asm .
It does not take parameters, It reads 1 integer from Stdin and writes it to R [V0].
If Stdin is the "screen", it takes the numbers from the user.
Otherwise, reads 1 integer from the stdin file.
If there are no more integer, it will check process whether is pipe.
If it is pipe process, It will check their pipe process partner is finished. Because input may come from pipe partner.
Otherwise, it returns -1 with R[REG_V0] = -1
Thus, the processes will understand that there is no input to read.
Then, swapProcess () function is called to run next the processes for multi programming or parallel running.

PRINT_INT_OUTPUT:
This syscall is used by the random.asm, sort.asm and search.asm .
It takes parameters from R[REG_A0], there is no return value.
Prints 1 integer to Stdout.
If Stdout is the screen, it prints the numbers to the screen.
Otherwise, prints 1 integer to stdout file.
swapProcess () function is called to run next the processes for multi programming or parallel running.

PROCESS_EXIT:
All assembly files use this syscall to exit.
Deletes the process that finished from the process table.
It does not take parameters.
If all processes are finished (If shell.asm call this syscall), it will free spim memory.
If the file is a pipe process, pipe files are closed and pipe counter is reduced by 1 . This counter is used when creating a pipe file.
If all processes are finished, it will deletes pipe files.
Then, swapProcess () function is called to run next the processes for multi programming or parallel running.

READ_CHAR_INPUT:
This syscall is used by the type.asm .
This is special for syscall type process. Because type process takes a file as a parameter.
Process parameter file is kept in Process table. Thus, it can be reached directly.
It does not take parameters, It reads 1 character from file parameter and writes it to R [V0].
If reading is successful, R [3] = 1, if not successful, R [3] = -1.
So type.asm will check the loop condition.
Then, swapProcess () function is called to run next the processes for multi programming or parallel running.

PRINT_CHAR_OUTPUT:
This syscall is used by the type.asm .
It takes parameters from R[REG_A0], there is no return value.
Prints 1 character to Stdout.
If Stdout is the screen, it prints the character to the screen.
Otherwise, prints 1 character to stdout file.
Then, swapProcess () function is called to run next the processes for multi programming or parallel running.

PRINT_STRING_OUTPUT:
This syscall is used by the type.asm and ls.asm .
If process is “ls”, it does not take parameter, it will print their namelist. This name list keep scanned directory files.
Otherwise it takes parameters from R[REG_A0], there is no return value.
If Stdout is the screen, It will print string screen.
Otherwise, It will print this string to file.
Then, swapProcess () function is called to run next the processes for multi programming or parallel running.

SCANDIR:
This syscall is used by the ls.asm .
This syscall is special for ls process to scan directory.
It will determine file index in scanned namelist.
It ignores files created for pipe processes because those files will be deleted.
It also ignore '.' and '..' directory paths.
One file name will be loaded at a time.
swapProcess () function is called to run next the processes for multi programming or parallel running.

PROGRAMS RUNNING PARALLEL & MULTI PROGRAMMING:

I created one process struct.
This struct holds the following information:

1. process name
2. stdin
3. stdout
4. file parameter
5. stdin file descriptor (If stdin is file)
6. stdout file descriptor (If stdout is file)
7. Process ID
8. pipe, background, run etc. some spesific variables.
9. Program Counter
10. Process State
11. Registers
12. namelist (for ls process)

I created a vector in this struct type. This vector denotes Process Table.

When Shell.asm runs, It receives a command string from the user with "READ_COMMAND_SYSCALL".
Then, this command is parsed to determine the processes(like sort, search etc.) and information(like stdin, stdout, pipe etc.) of the processes.
If there is no error in the command or an error occurred while creating the processes, Then the determined processes are added to the process table.

I wrote and customized syscalls according to the needs of the processes.
In syscall.cpp, I did not use any syscall other than the syscalls I wrote.
The swapProcess() function is called after every syscall I use or write.
Thus, after the necessary operations are performed on each syscall, the next process is run thanks to the swapProcess() function.
Thus, a different process will be run after each syscall, thus a parallel programming will be provided.
When all processes in the process table are completed, the shell process continues from where it left off.

swapProcess () is mainly as follows:

1. Update current process
2. Restore new process to spim

FILE OPERATIONS:

If process's stdout or stdin is a file, or process have a file parameter, file operations are used.
I used the open (), close (), read (), write () functions from the unix systemcall functions for file operations.
While filling in the process information, if there is a file, the file is opened.

OPEN & CLOSE FILES:
After opening the files, the process is written to the relevant descriptor.
I opened the stdin files or file parameter like this:
int fd = open(path, O_RDWR, 0666);
File mode is read and write.
If the file does not exist, it will print an error message.

I opened the stdout files like this:
int fdout = open(path, O_RDWR | O_CREAT | O_TRUNC, 0666);
File mode is read and write.
If the file does not exist, create it.
If the file exists, reset its contents. (In order not to overwrite the existing file)

When the process is completed, open files are closed with the close () function.
close(fd);

READ & WRITE FILES:
It read input on file with read() function.
read(fd, bufferpointer, bytes);
It write output on file with write() function.
write(fout, bufferpointer, bytesread)) == -1);

PIPE FILES:
There is a counter for specifying files for pipe processes.
This counter starts with 0.
This counter is increased by 1 whenever a pipe process occurs.
Pipe file names like that “processcount%d.txt”, counter
Like processcount1.txt, processcount2.txt … etc.
Thus, there will be a unique pipe file for each pipe process pair.

ASSEMBLY FILES:

SORT.ASM:
It reads from standard input positive integers (negative integer indicates the end of list), then sorts the integers in increasing order and prints the sorted integers to standard output.
It is a selection sort algorithm.
It takes 1 integer from stdin using READ_INT_INPUT syscall. And add these number to array.
These operations are repeated until negative integer comes.
Then sort array.
It prints sorted array by one by to stdout using PRINT_INT_OUTPUT syscall.
Then, exit program using PROCESS_EXIT syscall.

RANDOM.ASM:
It produces 100 random positive integers and prints them to the standard output. The last integer is a negative number.
It takes 1 random integer using GENERATE_RANDOM_NUMBER syscall.
It prints this number to stdout using PRINT_INT_OUTPUT syscall.
These operations are repeated 100 times.
End of the program It prints -1 to stdout using PRINT_INT_OUTPUT syscall.
Then, exit program using PROCESS_EXIT syscall.

SEARCH.ASM:
It reads from standard input an integer, then reads a number of positive integers. Makes a search on the list. prints the result to the standard output.
It is a linear search algorithm.
It takes 1 integer from stdin using READ_INT_INPUT syscall. And add these number to array.
These operations are repeated until negative integer comes.
Then search first number in array.
If found, prints the index to stdout using PRINT_INT_OUTPUT syscall.
Otherwise, prints -1 to stdout using PRINT_INT_OUTPUT syscall.
Then, exit program using PROCESS_EXIT syscall.

TYPE.ASM:
It works like UNIX type command. Prints the contents of a file to the standard output. At the end prints -1.
It takes 1 character from file parameter using READ_CHAR_INPUT syscall.
It prints this character to stdout using PRINT_CHAR_OUTPUT syscall.
These operations are repeated until end of file parameter comes.
End of the program It prints “-1” to stdout using PRINT_STRING_OUTPUT syscall.
Then, exit program using PROCESS_EXIT syscall.

LS.ASM:
It very simplified ls command of UNIX. prints the contents of the directory to the standard output.
There is only one directory and it is current directory.
It scans current directory using SCANDIR syscall.
It prints this scanned file name to stdout using PRINT_STRING_OUTPUT syscall.
These operations are repeated until last file.
Then, exit program using PROCESS_EXIT syscall.

SHELL.ASM:
Shell program that will be very similar to tcsh. It can support file processing (read, write, create) and multiprocessing.
It can run sort.asm, random.asm, search.asm, type.asm, ls.asm.
It also includes features such as piping, redirection, background task, stdin, stdout, screen and keyboard read / write.
It takes an command using READ_COMMAND_SYSCALL.
If command is exit, it will exit program using PROCESS_EXIT syscall.
Otherwise, takes command operations will continue.

RUN EXAMPLES:

```
cse312@ubuntu:~/Desktop/makeup$ spim read Shell.asm
Loaded: /usr/share/spim/exceptions.s

> random

23 57 81 99 13 30 36 57 71 13 29 79 33 0 63 26 30 55 68 18 90 4 40 60 50 43 94 9
2 33 5 11 56 14 92 7 27 22 43 36 45 8 18 76 41 18 39 19 0 46 88 19 37 92 59 97 9
4 54 44 39 87 49 50 95 63 42 2 42 64 97 79 61 57 97 89 50 67 28 70 68 75 10 39 6
4 2 50 13 49 5 57 88 44 58 38 92 73 80 46 16 96 96 -1

> random | sort

1 1 1 2 3 4 7 8 8 8 9 11 11 11 11 12 12 14 14 15 15 21 21 22 23 23 26 27 30 30 3
1 33 33 33 35 36 37 37 38 39 43 43 47 48 49 49 50 50 52 52 53 53 55 57 58 60 61
63 63 64 64 65 65 68 69 70 70 71 72 73 73 73 74 76 76 78 78 78 78 79 80 80 81 84
 86 87 87 89 89 90 91 92 92 94 95 95 98 98 99 99

> random | sort | search

-1

> random | sort > 1.txt
```

```
> type 1.txt

1 2 2 6 8 8 8 13 13 15 16 17 17 21 23 23 25 30 31 32 32 33 33 33 35 35 35 36 39 41 44 46
 48 49 49 50 50 52 52 53 55 57 58 58 58 62 62 63 64 65 65 66 67 67 68 68 68 69 69 69 71
71 73 73 74 76 77 78 79 80 81 81 81 82 82 84 84 85 85 87 88 88 89 89 92 93 94 95 96 96 9
6 97 97 97 97 98 98 98 99 99  -1

> type 1.txt > 2.txt



> ls&

1.txt   2.txt   Report.odt      Shell.asm
Shell.asm~      ls.asm  random.asm      search.asm
sort.asm        spimsimulator-code-r730 type.asm
```

```
> sort

Enter number: 4
Enter number: 5
Enter number: 3
Enter number: 0
Enter number: 9
Enter number: -1
0 3 4 5 9

> search

Enter number: 4
Enter number: 6
Enter number: 1
Enter number: 8
Enter number: 4
Enter number: 9
Enter number: -1
4
```

```
> search

Enter number: 4
Enter number: 6
Enter number: 1
Enter number: 8
Enter number: 9
Enter number: -1
-1
```

search take input 1.txt and print result screen
random output pipefile, sort take input this pipe file and print output 2.txt
type take an input from 2.txt parameter file, and print output pipefile, sort take input this pipe file and print output 3.txt
sort take input 1.txt and print result screen
ls output pipefile, other ls take input this pipe file but ls process does not read from input. So, there is nothing changing, print result screen.

```
> search < 1.txt & random | sort > 2.txt& type 2.txt | sort > 3.txt& sort < 1.txt& ls|ls
&

-1




1 2 2 6 8 8 8 13 13 15 16 17 17 21 23 23 25 30 31 32 32 33 33 33 35 35 35 36 39 41 44 46 48 49
 49 50 50 52 52 53 55 57 58 58 58 62 62 63 64 65 65 66 67 67 68 68 68 69 69 69 71 71 73 73 74
76 77 78 79 80 81 81 81 82 82 84 84 85 85 87 88 88 89 89 92 93 94 95 96 96 96 97 97 97 97 98 9
8 98 99 99

1.txt   2.txt   3.txt   Report.odt
Shell.asm       Shell.asm~      ls.asm  random.asm
search.asm      sort.asm        spimsimulator-code-r730 type.asm
```

```
> ls | ls& ls > 2.txt & ls|ls

1.txt   2.txt   3.txt   Report.odt
Shell.asm       Shell.asm~      ls.asm  random.asm
search.asm      sort.asm        spimsimulator-code-r730 type.asm



1.txt   2.txt   3.txt   Report.odt
Shell.asm       Shell.asm~      ls.asm  random.asm
search.asm      sort.asm        spimsimulator-code-r730 type.asm

>
```

```
> ls > 1.txt & type 1.txt > 2.txt




> type 2.txt

1.txt   2.txt   3.txt   Report.odt
Shell.asm       Shell.asm~      ls.asm  random.asm
search.asm      sort.asm        spimsimulator-code-r730 type.asm
 -1  -1
```

```
cse312@ubuntu:~/Desktop/makeup$ spim read Shell.asm
Loaded: /usr/share/spim/exceptions.s

> random > 1.txt& random > 2.txt




> type 1.txt& type 2.txt | search& random > 3.txt& ls

42 14 69 30 84 88 81 15 68 24 84 34 27 90 18 5 5 7 93 5 49 88 53 44 87 75 78 41 50 60 86 89 7
3 53 20 57 41 1 69 7 22 53 41 48 44 58 50 46 62 43 48 10 31 98 53 19 73 29 59 22 88 43 11 59
95 30 16 35 28 82 41 49 35 81 95 76 37 45 20 99 88 67 7 17 65 59 35 37 85 92 58 71 35 66 30 2
9 94 43 63 22 -1  -1
-1


1.txt   2.txt   3.txt   Report.odt
Shell.asm       Shell.asm~      ls.asm  random.asm
search.asm      sort.asm        spimsimulator-code-r730 type.asm


> exit

cse312@ubuntu:~/Desktop/makeup$
```