בס"ד

מטלה 2 מערכות הפעלה

מגישות: רננה תורג'מן ותמר בר אילן.

PART A:

In this part we implement two small programs, that fills like a regular CMD tools.

Tool 1: “cmp"

the tool compare two files, and return “0” if they are equal, and “1” if not (return an INT)

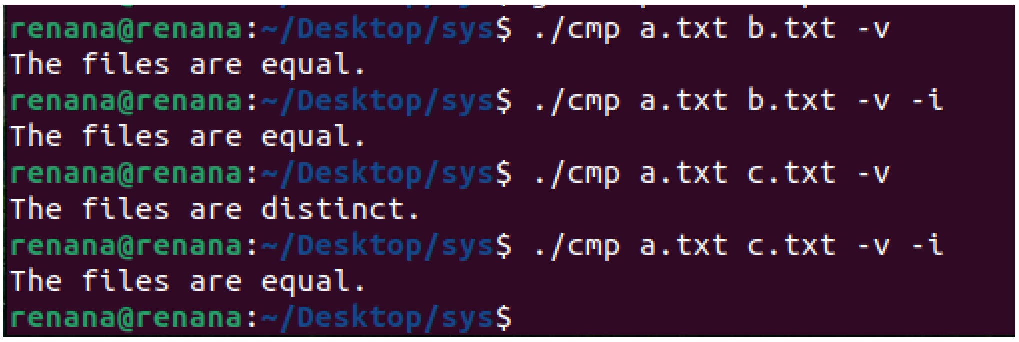
The tool will support -v flag for verbose output. By this we mean that the tool will print .“equal” or "distinct ", in addition to returning the int value

The toll support -i flag, that mean “ignore case” so “AAA” and “aaa” meaning equals

For run this part: ./cmp <file1> <file2> -v -i

For example we add to our submission three text files. a.txt and b.txt are the same, c.txt is different in that it has some capitalized words.

So we can see that the results are match:



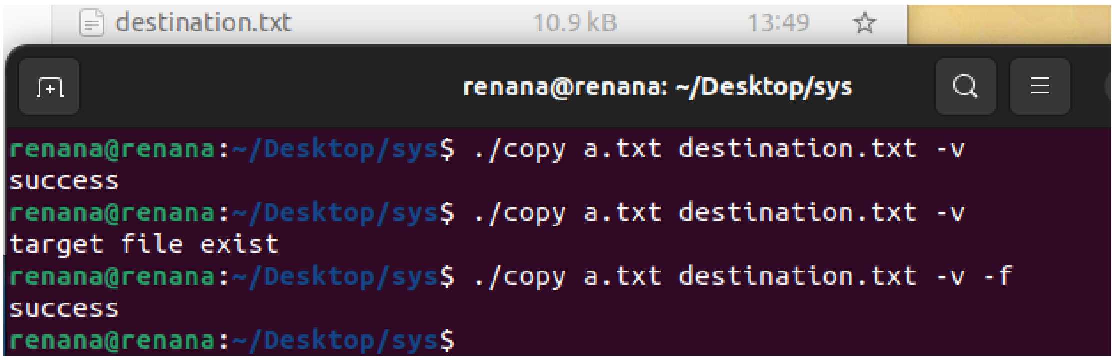
```
renana@renana:~/Desktop/sys$ ./cmp a.txt b.txt -v
The files are equal.
renana@renana:~/Desktop/sys$ ./cmp a.txt b.txt -v -i
The files are equal.
renana@renana:~/Desktop/sys$ ./cmp a.txt c.txt -v
The files are distinct.
renana@renana:~/Desktop/sys$ ./cmp a.txt c.txt -v -i
The files are equal.
renana@renana:~/Desktop/sys$
```

Tool 2: “copy”

 the tool copy a file to another place and/or name. The tool return “0” on success, or “1” on failure (return an INT) The tool create a new file, if it does not exist, but it will not overwrite a file if it do exist.
the tool support -v flag, that will output “success” if the file is copied, or “target file exist” if this is the case, or “general failure” on some other problem (in addition to the returned INT value). The tool support -f flag (that means force), that allows to overwrite the target file.

for run:
./copy <file1> <file2> [-f] [-v]

for example: ./copy source.txt destination.txt -v
This will copy the contents of "source.txt" to "destination.txt" and print "success" if the operation was successful.



```
renana@renana:~/Desktop/sys$ ./copy a.txt destination.txt -v
success
renana@renana:~/Desktop/sys$ ./copy a.txt destination.txt -v
target file exist
renana@renana:~/Desktop/sys$ ./copy a.txt destination.txt -v -f
success
renana@renana:~/Desktop/sys$
```

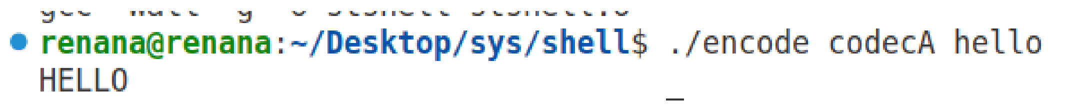
In the first run the destination.txt file was not exist, and we got success. (and the file was open). In the second run the destination.txt file was exist and we don't get the -f flag as an argument so we got " target file exist" message. In the third run the destination.txt file was still exist but we pass the -f flag as an argument so we got success and the file update.

PART B:

we implement a coding library. We have two codding methods.
Method a, named **codecA:** covert all lower case chars to upper case, and all upper case to lower case. All other chars remain unchanged.
Method b, named **codecB:** convert all chars to the 3-rd next char (adding a number of 3 to the ascii val).
The libraries support "encode" and "decode" methods.
the libraries is "reversable", meaning that if one does "encode" and then "decode, he will get the original string
we write 2 shared libraries , each implementing it's algorithm and two tools, named encode and decode, to utilize the libraries.
The tools will gets some text and convert it according to selected library.

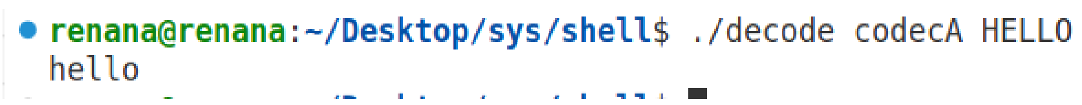
This program is implementation of dynamic linking and loading libraries in C programming language. It includes two codec libraries, codecA and codecB, each with an encoding and decoding function. The program also includes an encoding tool and a decoding tool that use these libraries to encode and decode messages respectively.

When the program is run with the command "./encode codecA hello", it loads the dynamic library libcodecA.so, which contains the codecA_encode and codecA_decode functions. It then calls the codecA_encode function with the message "hello" and prints the encoded message "HELLO" to the console.



```
renana@renana:~/Desktop/sys/shell$ ./encode codecA hello
HELLO
```

Similarly, when the program is run with the command "./decode codecA HELLO", it loads the dynamic library libcodecA.so, obtains a function pointer to the codecA_decode function, and calls it with the message "HELLO". It then prints the decoded message "hello" to the console.



```
renana@renana:~/Desktop/sys/shell$ ./decode codecA HELLO
hello
```

The same functionality is implemented for codecB by replacing the "codecA" string with "codecB" in the command line arguments.

The program uses the dlfcn.h header file to load the dynamic libraries at runtime and obtain function pointers to the exported functions in the libraries. It also frees the memory allocated by the libraries using the dlclose function.

Usage : encode/decode
output: encoded/decoded string

PART C:

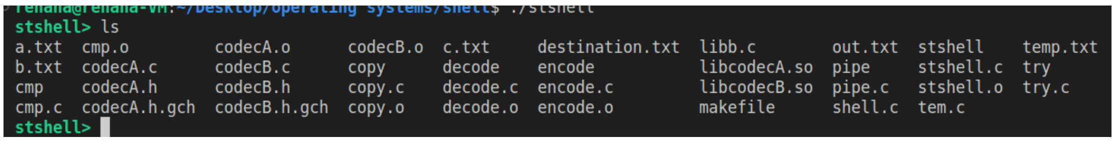
In this part we write a shell program named **stshell**.
The program provides a prompt where the user can enter commands. It then parses the user input and executes the command as a child process.

Features:

1) Be able to run CMD tools that exist on system (by fork + exec + wait)
2) Be able to stop running tool by pressing Ctrl+c, but not killing the shell itself (by signal handler)
3) Be able to redirect output with “>” and “>>”, and allow piping with “|”, at least for 2 following pipes.
4) be able to stop itself by “exit” command

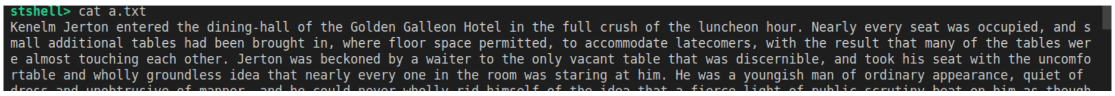
Examples of commands:

ls: lists the files in the current directory



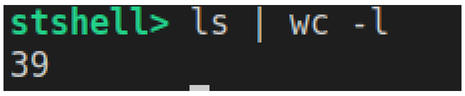
```
renana@renana-VM:~/Desktop/operating systems/shell$ ./stshell
stshell> ls
a.txt  cmp.o         codecA.o      codecB.o  c.txt     destination.txt  libb.c        out.txt  stshell    temp.txt
b.txt  codecA.c      codecB.c      copy      decode    encode           libcodecA.so  pipe     stshell.c  try
cmp    codecA.h      codecB.h      copy.c    decode.c  encode.c         libcodecB.so  pipe.c   stshell.o  try.c
cmp.c  codecA.h.gch  codecB.h.gch  copy.o    decode.o  encode.o         makefile      shell.c  tem.c
stshell> 
```

cat file.txt: displays the contents of the file "file.txt"



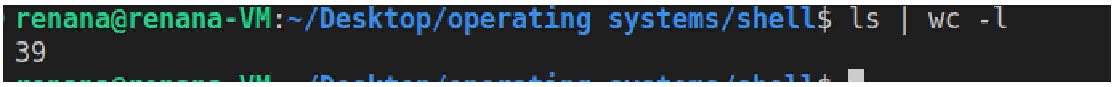
```
stshell> cat a.txt
Kenelm Jerton entered the dining-hall of the Golden Galleon Hotel in the full crush of the luncheon hour. Nearly every seat was occupied, and s
mall additional tables had been brought in, where floor space permitted, to accommodate latecomers, with the result that many of the tables wer
e almost touching each other. Jerton was beckoned by a waiter to the only vacant table that was discernible, and took his seat with the uncomfo
rtable and wholly groundless idea that nearly every one in the room was staring at him. He was a youngish man of ordinary appearance, quiet of
dress and unobtrusive of manner, and he could never wholly rid himself of the idea that a fierce light of public scrutiny beat on him as though
```

ls | wc -l: lists the files in the current directory and then counts the number of lines in the output



```
stshell> ls | wc -l
39
```

We can see the answer is the same if we run this command in the tirminal:

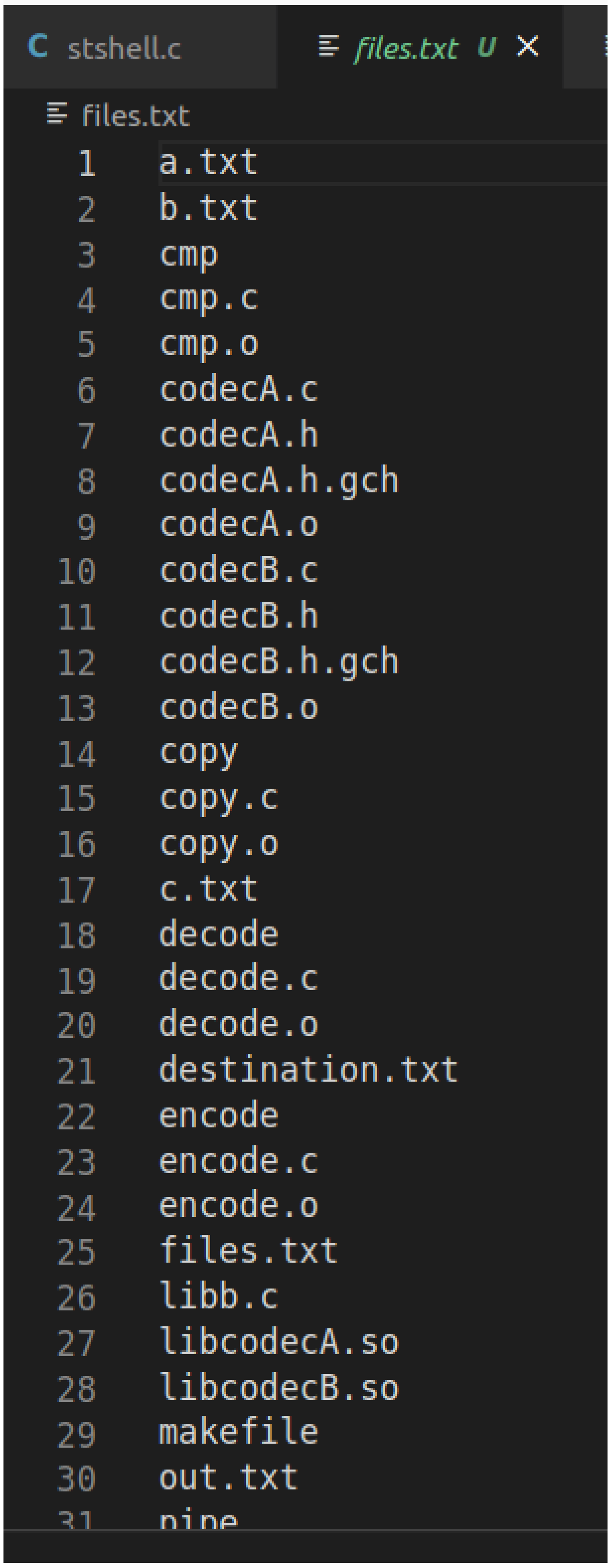


```
renana@renana-VM:~/Desktop/operating systems/shell$ ls | wc -l
39
renana@renana-VM:~/Desktop/operating systems/shell$ 
```

To redirect output to a file, use the > or >> operator followed by the filename. The > operator overwrites the file if it already exists, while the >> operator appends to the file if it already exists.

Example of output redirection:

ls > files.txt: writes the output of the "ls" command to a file called "files.txt"
in this case its create this file for the first time

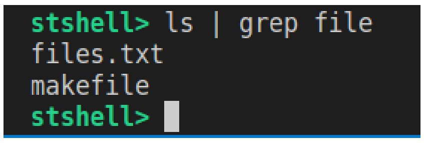


```
a.txt
b.txt
cmp
cmp.c
cmp.o
codecA.c
codecA.h
codecA.h.gch
codecA.o
codecB.c
codecB.h
codecB.h.gch
codecB.o
copy
copy.c
copy.o
c.txt
decode
decode.c
decode.o
destination.txt
encode
encode.c
encode.o
files.txt
libb.c
libcodecA.so
libcodecB.so
makefile
out.txt
pipe
```

To pipe commands, use the | operator to send the output of one command as input to the next command.

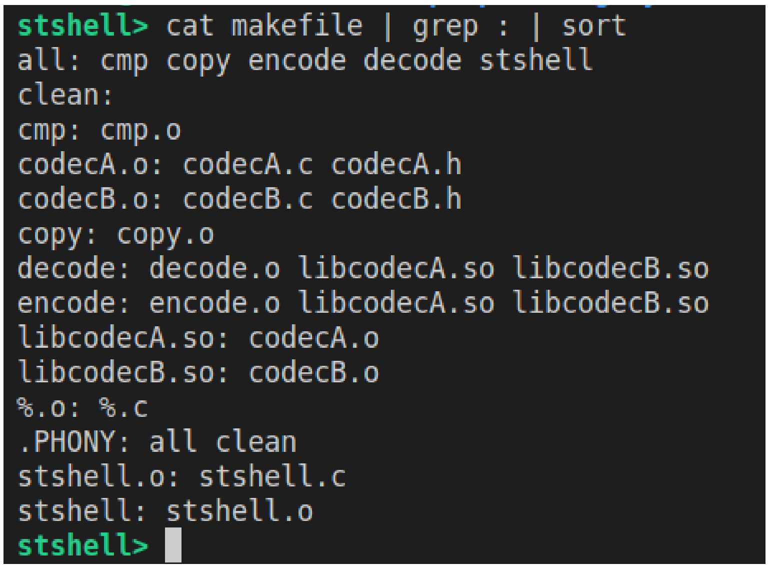
Example of command piping:

ls | grep "file": lists the files in the current directory and then searches for the word "file" in the output



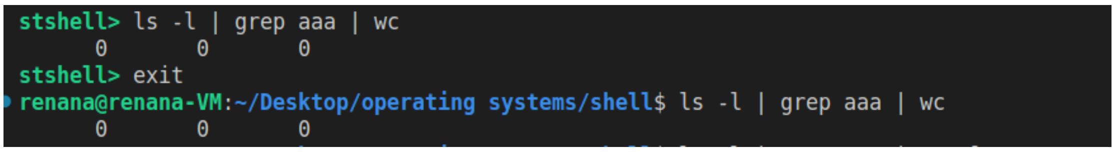
```
stshell> ls | grep file
files.txt
makefile
stshell> 
```

cat makefile | grep : | sort : This command reads the contents of the file "makefile" using the "cat" command, pipes the output to the "grep" command to search for lines containing the character ":", and then pipes the output to the "sort" command to sort the lines in alphabetical order.



```
stshell> cat makefile | grep : | sort
all: cmp copy encode decode stshell
clean:
cmp: cmp.o
codecA.o: codecA.c codecA.h
codecB.o: codecB.c codecB.h
copy: copy.o
decode: decode.o libcodecA.so libcodecB.so
encode: encode.o libcodecA.so libcodecB.so
libcodecA.so: codecA.o
libcodecB.so: codecB.o
%.o: %.c
.PHONY: all clean
stshell.o: stshell.c
stshell: stshell.o
stshell> 
```

ls -l | grep aaa | wc



```
stshell> ls -l | grep aaa | wc
      0       0       0
stshell> exit
renana@renana-VM:~/Desktop/operating systems/shell$ ls -l | grep aaa | wc
      0       0       0
```

To exit the shell, simply type "exit" and press enter. (can see it in the picture below)