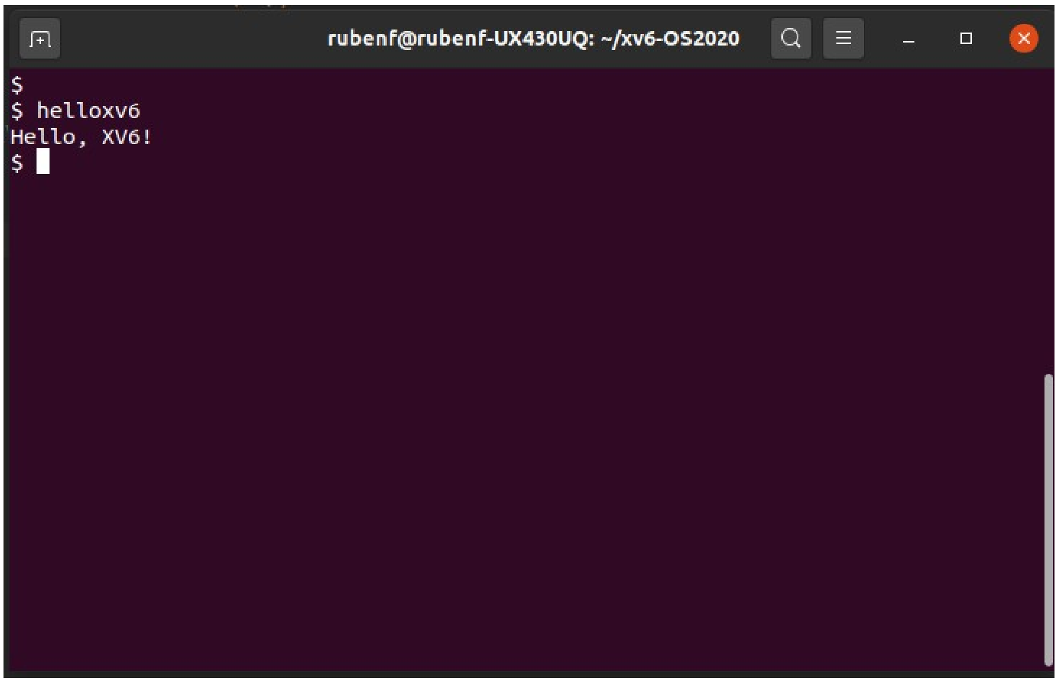
Report – XV6
Operating System 2020

ראובן פרתי – 340895499
שחר בר – 3302957824

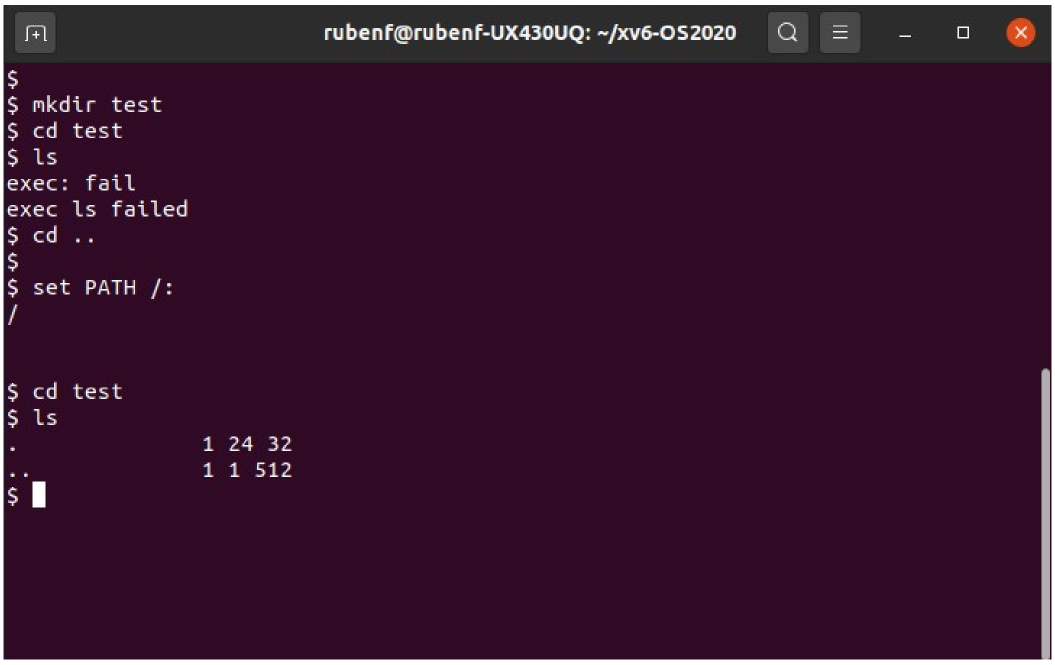
Task 1: helloxv6.c



```
rubenf@rubenf-UX430UQ: ~/xv6-OS2020
$
$ helloxv6
Hello, XV6!
$ 
```

Task 2: Support the PATH environment variable ('**set PATH**')

If, upon typing a command, the required file is not found in the current working directory, the shell attempts to execute the file from one of the directories specified by the PATH. An error message is printed only if the required file was not found in the working directory or any of the directories listed in the PATH.

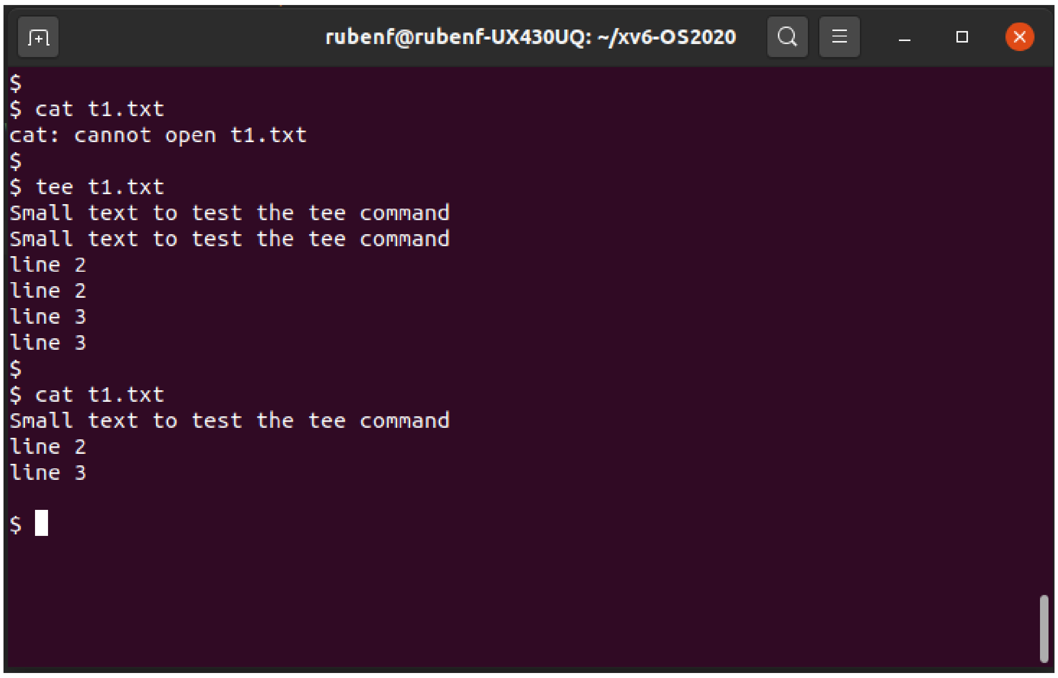


```
rubenf@rubenf-UX430UQ: ~/xv6-OS2020
$
$ mkdir test
$ cd test
$ ls
exec: fail
exec ls failed
$ cd ..
$
$ set PATH /:
/


$ cd test
$ ls
.              1 24 32
..             1 1 512
$ 
```

Task 3.1: tee.c

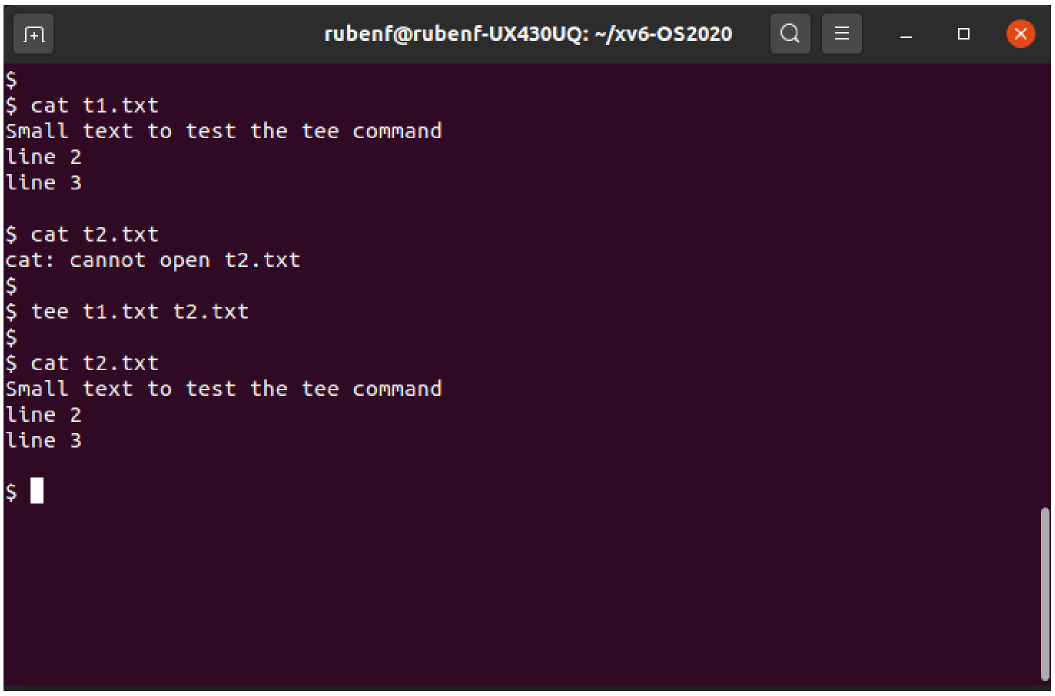
1. One argument: **tee t1.txt** → read from standard input (until ^D)and write to both standard output and the file provided as argument



```
$
$ cat t1.txt
cat: cannot open t1.txt
$
$ tee t1.txt
Small text to test the tee command
Small text to test the tee command
line 2
line 2
line 3
line 3
$
$ cat t1.txt
Small text to test the tee command
line 2
line 3

$ 
```

2. Two arguments: **tee t1.txt t2.txt** → read from the first file (argument 1) and write to the second file (argument 2)



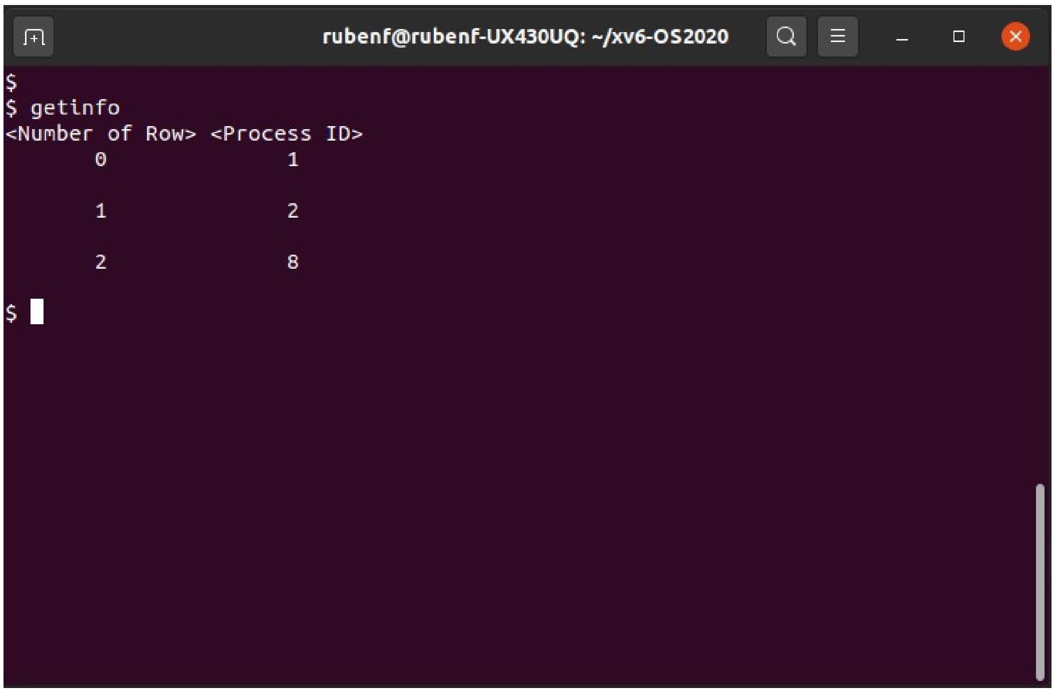
```
$
$ cat t1.txt
Small text to test the tee command
line 2
line 3

$ cat t2.txt
cat: cannot open t2.txt
$
$ tee t1.txt t2.txt
$
$ cat t2.txt
Small text to test the tee command
line 2
line 3

$ 
```

Task 3.2 : getpinfo system call (cmd getinfo – System Call 22)

getpinfo is system call the print a list of currently process, it's displays information about the process



```
$
$ getinfo
<Number of Row> <Process ID>
       0              1

       1              2

       2              8

$ 
```

Explanation: Three processes are currently running when the command is called:

0	init	ID: 1
1	sh	ID: 2
2	getinfo	ID: 8

Task 4: Wait and exit system calls

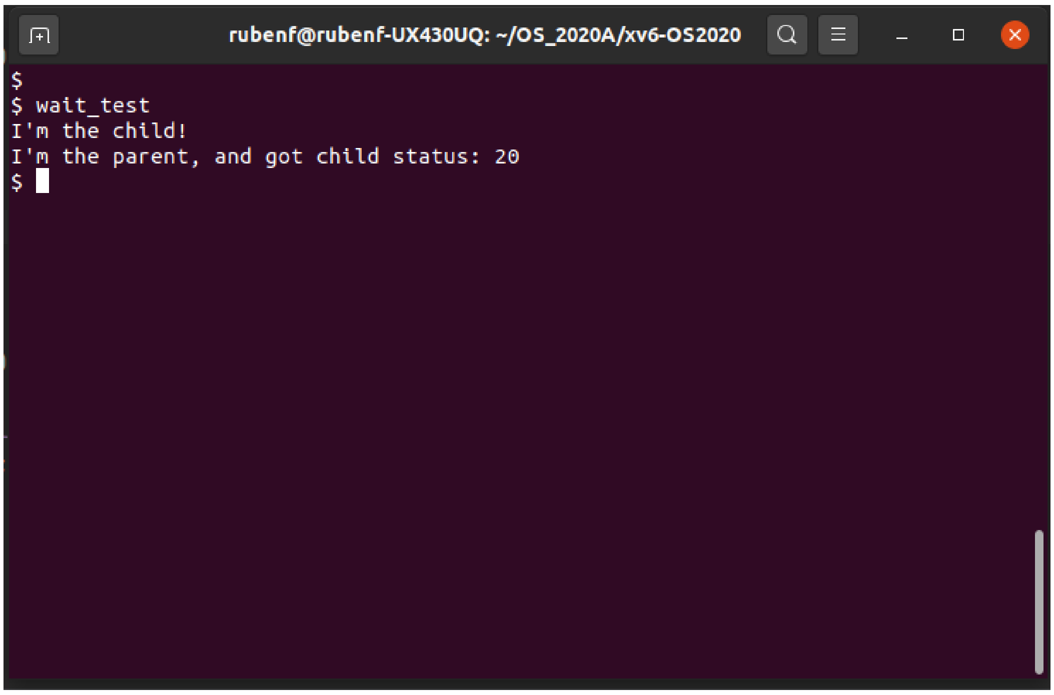
In **Tasks 4.1** and **4.2** we updated the exit and wait system calls.

The exit call signature has been changed to receive a status: void exit(int status). In addition it stores the exit status of the terminated process in the proc structure

The wait call signature has been changed to update the status: void exit(int *status). The wait system call now blocks the current process execution until any of its child processes is terminated (if any exists) and return the terminated child exit status through the status argument. In addition it updates the status received with the child status stored in the proc structure by the exit system call.

To test the exit and wait functionalities, we added a tester (wait_test.c) that creates a child process and wait for its termination. The child exit with a specific status (20).

The exit function saves the status number in the proc structure and when the parent resumes (after its child termination), it can read the terminated child status (in our case 20):

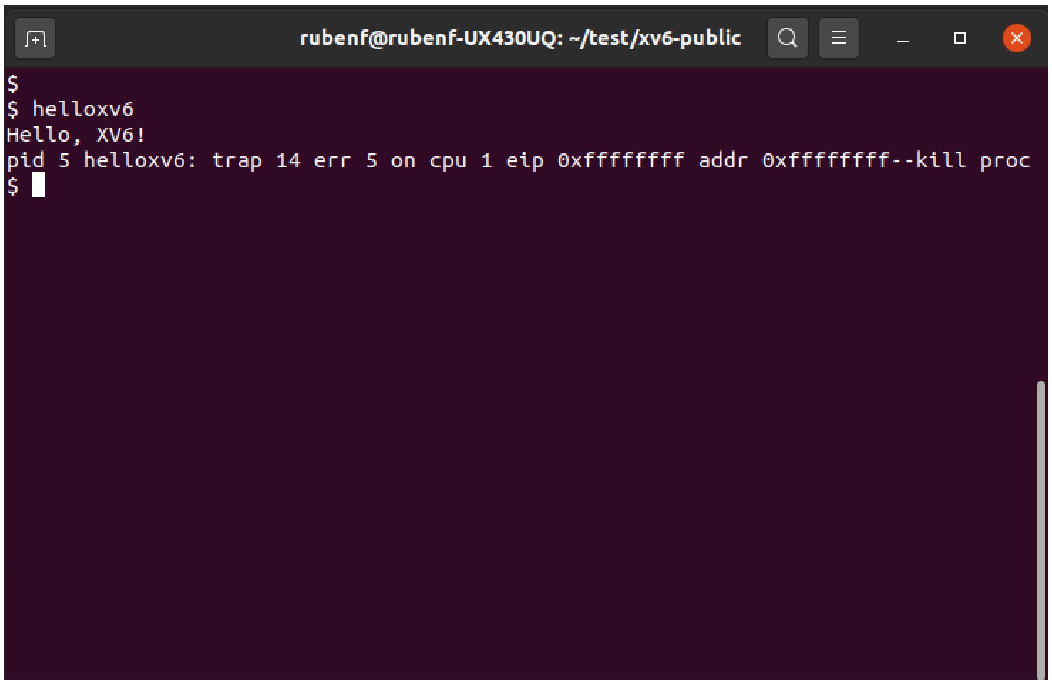


```
$
$ wait_test
I'm the child!
I'm the parent, and got child status: 20
$ 
```

Task 4.3: Implicit exit call

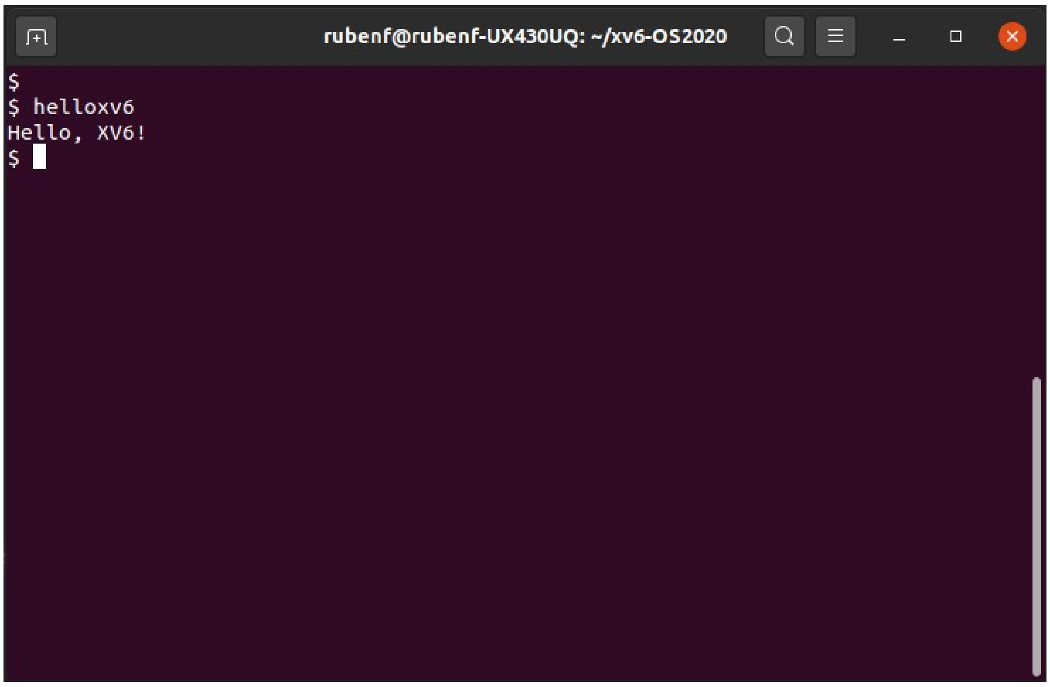
If a user program exit the main function, the exit system call will be implicitly performed, and the return value of the main function must be the status argument for the implicit exit system call.

Before:



```
$
$ helloxv6
Hello, XV6!
pid 5 helloxv6: trap 14 err 5 on cpu 1 eip 0xffffffff addr 0xffffffff--kill proc
$ 
```

After (Same code – helloxv6.c):



```
$
$ helloxv6
Hello, XV6!
$
```

Explanation: Although the code is the same, after the implementation of the implicit call to the exit function at the end of every process, the kernel isn't obliged to kill the process to finish it when the explicit exit call is missing.