Programming Problems 1110590450 歐佳昀

Chap. 2

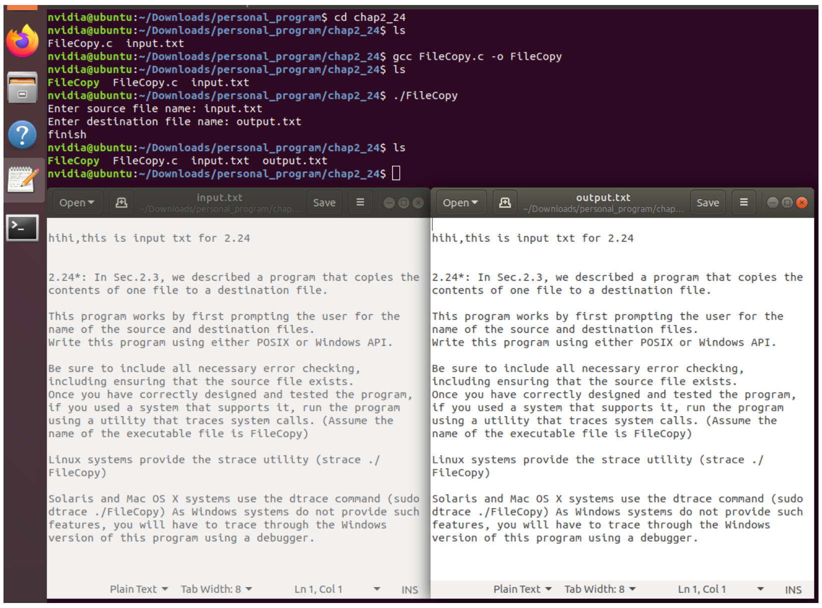
- 2.24*: In Sec.2.3, we described a program that copies the contents of one file to a destination file.
    - This program works by first prompting the user for the name of the source and destination files.
- Write this program using either POSIX or Windows API.
    - Be sure to include all necessary error checking, including ensuring that the source file exists.
- Once you have correctly designed and tested the program, if you used a system that supports it, run the program using a utility that traces system calls. (Assume the name of the executable file is FileCopy)
    - Linux systems provide the strace utility

      (strace ./FileCopy)
    - Solaris and Mac OS X systems use the dtrace command

      (sudo dtrace ./FileCopy)

      As Windows systems do not provide such features, you will have to trace through the Windows version of this program using a debugger.

./FileCopy



```
nvidia@ubuntu:~/Downloads/personal_program$ cd chap2_24
nvidia@ubuntu:~/Downloads/personal_program/chap2_24$ ls
FileCopy.c  input.txt
nvidia@ubuntu:~/Downloads/personal_program/chap2_24$ gcc FileCopy.c -o FileCopy
nvidia@ubuntu:~/Downloads/personal_program/chap2_24$ ls
FileCopy  FileCopy.c  input.txt
nvidia@ubuntu:~/Downloads/personal_program/chap2_24$ ./FileCopy
Enter source file name: input.txt
Enter destination file name: output.txt
finish
nvidia@ubuntu:~/Downloads/personal_program/chap2_24$ ls
FileCopy  FileCopy.c  input.txt  output.txt
nvidia@ubuntu:~/Downloads/personal_program/chap2_24$ 
```

input.txt ~/Downloads/personal_program/chap...

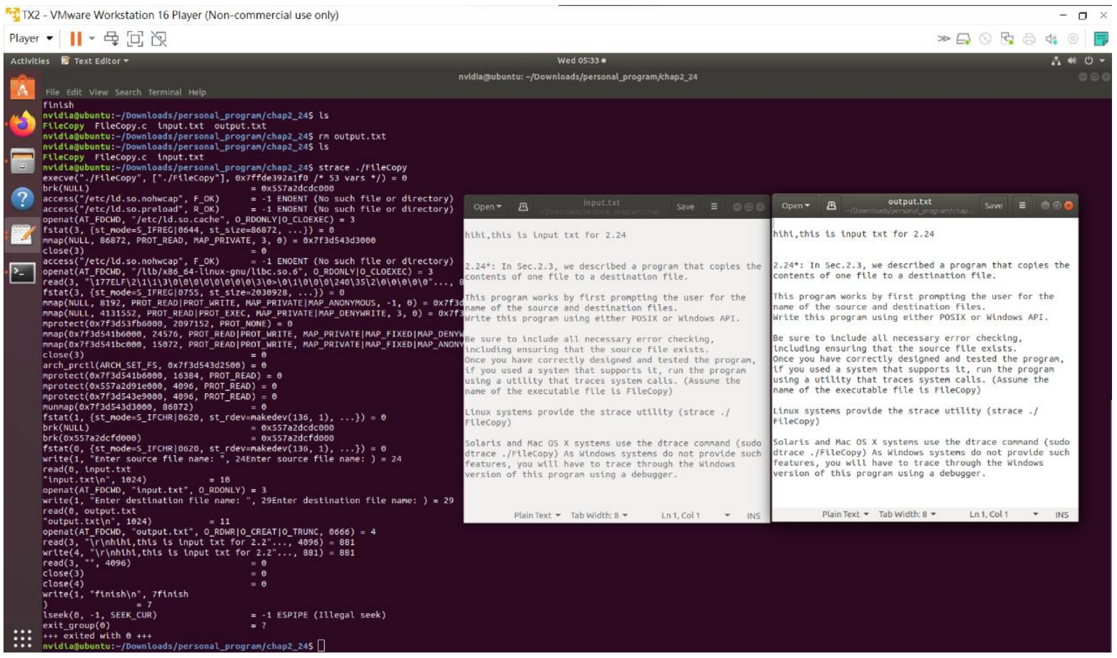
```
hihi,this is input txt for 2.24


2.24*: In Sec.2.3, we described a program that copies the
contents of one file to a destination file.

This program works by first prompting the user for the
name of the source and destination files.
Write this program using either POSIX or Windows API.

Be sure to include all necessary error checking,
including ensuring that the source file exists.
Once you have correctly designed and tested the program,
if you used a system that supports it, run the program
using a utility that traces system calls. (Assume the
name of the executable file is FileCopy)

Linux systems provide the strace utility (strace ./
FileCopy)

Solaris and Mac OS X systems use the dtrace command (sudo
dtrace ./FileCopy) As Windows systems do not provide such
features, you will have to trace through the Windows
version of this program using a debugger.
```

output.txt ~/Downloads/personal_program/chap...

```
hihi,this is input txt for 2.24


2.24*: In Sec.2.3, we described a program that copies the
contents of one file to a destination file.

This program works by first prompting the user for the
name of the source and destination files.
Write this program using either POSIX or Windows API.

Be sure to include all necessary error checking,
including ensuring that the source file exists.
Once you have correctly designed and tested the program,
if you used a system that supports it, run the program
using a utility that traces system calls. (Assume the
name of the executable file is FileCopy)

Linux systems provide the strace utility (strace ./
FileCopy)

Solaris and Mac OS X systems use the dtrace command (sudo
dtrace ./FileCopy) As Windows systems do not provide such
features, you will have to trace through the Windows
version of this program using a debugger.
```

or use this command to strace

```
strace ./FileCopycle
```


TX2 - VMware Workstation 16 Player (Non-commercial use only)
Player
Activities
Text Editor
Wed 05:33
nvidia@ubuntu: ~/Downloads/personal_program/chap2_24
File Edit View Search Terminal Help

```
finish
nvidia@ubuntu:~/Downloads/personal_program/chap2_24$ ls
FileCopy  FileCopy.c  input.txt  output.txt
nvidia@ubuntu:~/Downloads/personal_program/chap2_24$ rm output.txt
nvidia@ubuntu:~/Downloads/personal_program/chap2_24$ ls
FileCopy  FileCopy.c  input.txt
nvidia@ubuntu:~/Downloads/personal_program/chap2_24$ strace ./FileCopy
execve("./FileCopy", ["./FileCopy"], 0x7ffde392a1f0 /* 53 vars */) = 0
brk(NULL)                               = 0x557a2dcdc000
access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or directory)
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
fstat(3, {st_mode=S_IFREG|0644, st_size=86872, ...}) = 0
mmap(NULL, 86872, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f3d543d3000
close(3)                                = 0
access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\240\35\2\0\0\0\0\0"..., 8
fstat(3, {st_mode=S_IFREG|0755, st_size=2030928, ...}) = 0
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f3d
mmap(NULL, 4131552, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7f3
mprotect(0x7f3d53fb6000, 2097152, PROT_NONE) = 0
mmap(0x7f3d541b6000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYW
mmap(0x7f3d541bc000, 15072, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONY
close(3)                                = 0
arch_prctl(ARCH_SET_FS, 0x7f3d543d2500) = 0
mprotect(0x7f3d541b6000, 16384, PROT_READ) = 0
mprotect(0x557a2d91e000, 4096, PROT_READ) = 0
mprotect(0x7f3d543e9000, 4096, PROT_READ) = 0
munmap(0x7f3d543d3000, 86872)           = 0
fstat(1, {st_mode=S_IFCHR|0620, st_rdev=makedev(136, 1), ...}) = 0
brk(NULL)                               = 0x557a2dcdc000
brk(0x557a2dcfd000)                     = 0x557a2dcfd000
fstat(0, {st_mode=S_IFCHR|0620, st_rdev=makedev(136, 1), ...}) = 0
write(1, "Enter source file name: ", 24Enter source file name: ) = 24
read(0, input.txt
"input.txt\n", 1024)            = 10
openat(AT_FDCWD, "input.txt", O_RDONLY) = 3
write(1, "Enter destination file name: ", 29Enter destination file name: ) = 29
read(0, output.txt
"output.txt\n", 1024)           = 11
openat(AT_FDCWD, "output.txt", O_RDWR|O_CREAT|O_TRUNC, 0666) = 4
read(3, "\r\nhihi,this is input txt for 2.2"..., 4096) = 881
write(4, "\r\nhihi,this is input txt for 2.2"..., 881) = 881
read(3, "", 4096)                       = 0
close(3)                                = 0
close(4)                                = 0
write(1, "finish\n", 7finish
)                   = 7
lseek(0, -1, SEEK_CUR)                  = -1 ESPIPE (Illegal seek)
exit_group(0)                           = ?
+++ exited with 0 +++
nvidia@ubuntu:~/Downloads/personal_program/chap2_24$
```

```
hihi,this is input txt for 2.24


2.24*: In Sec.2.3, we described a program that copies the
contents of one file to a destination file.

This program works by first prompting the user for the
name of the source and destination files.
Write this program using either POSIX or Windows API.

Be sure to include all necessary error checking,
including ensuring that the source file exists.
Once you have correctly designed and tested the program,
if you used a system that supports it, run the program
using a utility that traces system calls. (Assume the
name of the executable file is FileCopy)

Linux systems provide the strace utility (strace ./
FileCopy)

Solaris and Mac OS X systems use the dtrace command (sudo
dtrace ./FileCopy) As Windows systems do not provide such
features, you will have to trace through the Windows
version of this program using a debugger.
```

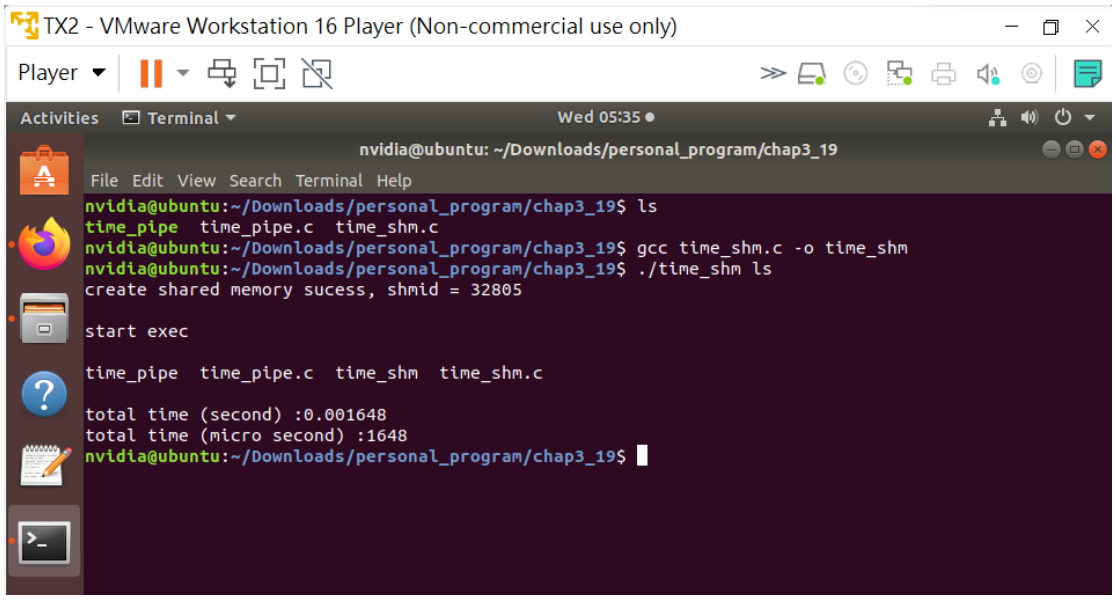
```

hihi,this is input txt for 2.24


2.24*: In Sec.2.3, we described a program that copies the
contents of one file to a destination file.

This program works by first prompting the user for the
name of the source and destination files.
Write this program using either POSIX or Windows API.

Be sure to include all necessary error checking,
including ensuring that the source file exists.
Once you have correctly designed and tested the program,
if you used a system that supports it, run the program
using a utility that traces system calls. (Assume the
name of the executable file is FileCopy)

Linux systems provide the strace utility (strace ./
FileCopy)

Solaris and Mac OS X systems use the dtrace command (sudo
dtrace ./FileCopy) As Windows systems do not provide such
features, you will have to trace through the Windows
version of this program using a debugger.
```

Chap. 3

- 3.19*: Write a C program called time.c that determines the amount of time necessary to run a command from the command line.
- This program will be run as "./time <command>" and will report the amount of elapsed time to run the specified command.
- This will involve using fork() and exec() functions, as well as the gettimeofday() function to determine the elapsed time.
- It will also require the use of two different IPC mechanisms.
- The first version will have the child process write the starting time to a region of shared memory before it calls exec().
    - After the child process terminates, the parent will read the starting time from shared memory.
    - Refer to Section 3.7.1 for details using POSIX shared memory.
- The second version will use a pipe.
    - The child will write the starting time to the pipe, and the parent will read from it following the termination of the child process.

version 1

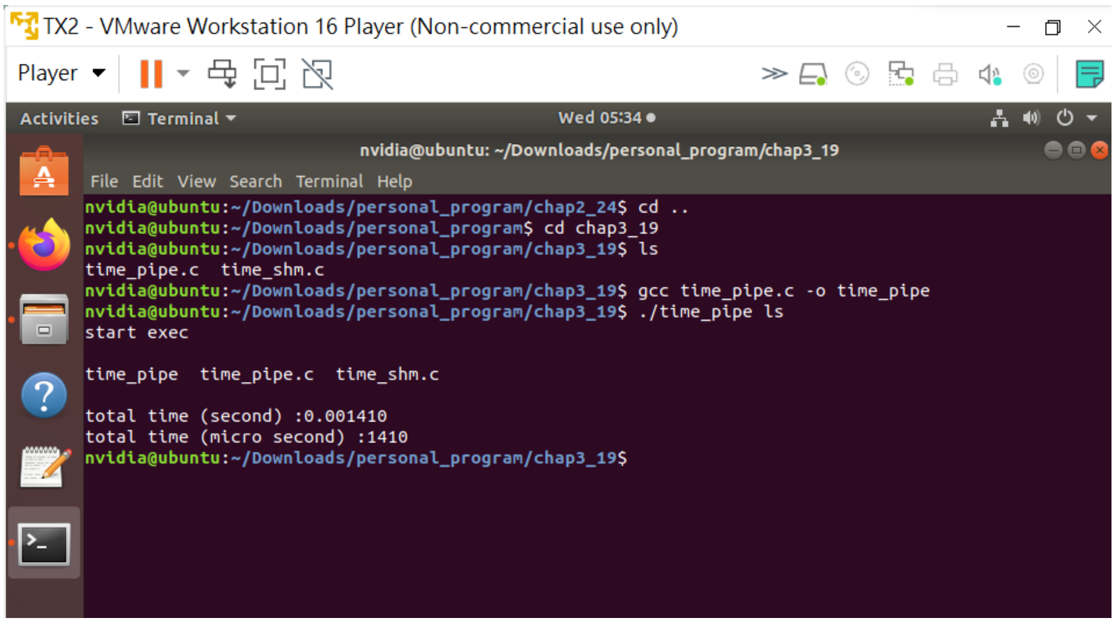
```
./time_shm <command>
```


TX2 - VMware Workstation 16 Player (Non-commercial use only)
Player
Activities
Terminal
Wed 05:35
nvidia@ubuntu: ~/Downloads/personal_program/chap3_19
File Edit View Search Terminal Help

```
nvidia@ubuntu:~/Downloads/personal_program/chap3_19$ ls
time_pipe  time_pipe.c  time_shm.c
nvidia@ubuntu:~/Downloads/personal_program/chap3_19$ gcc time_shm.c -o time_shm
nvidia@ubuntu:~/Downloads/personal_program/chap3_19$ ./time_shm ls
create shared memory sucess, shmid = 32805

start exec

time_pipe  time_pipe.c  time_shm  time_shm.c

total time (second) :0.001648
total time (micro second) :1648
nvidia@ubuntu:~/Downloads/personal_program/chap3_19$
```

version 2

```
./time_pipe <command>
```


TX2 - VMware Workstation 16 Player (Non-commercial use only)
Player
Activities
Terminal
Wed 05:34
nvidia@ubuntu: ~/Downloads/personal_program/chap3_19
File Edit View Search Terminal Help

```
nvidia@ubuntu:~/Downloads/personal_program/chap2_24$ cd ..
nvidia@ubuntu:~/Downloads/personal_program$ cd chap3_19
nvidia@ubuntu:~/Downloads/personal_program/chap3_19$ ls
time_pipe.c  time_shm.c
nvidia@ubuntu:~/Downloads/personal_program/chap3_19$ gcc time_pipe.c -o time_pipe
nvidia@ubuntu:~/Downloads/personal_program/chap3_19$ ./time_pipe ls
start exec

time_pipe  time_pipe.c  time_shm.c

total time (second) :0.001410
total time (micro second) :1410
nvidia@ubuntu:~/Downloads/personal_program/chap3_19$
```

- [optional] (3.21**): The Collatz conjecture concerns what happens when we take any positive integer n and apply the following algorithm:

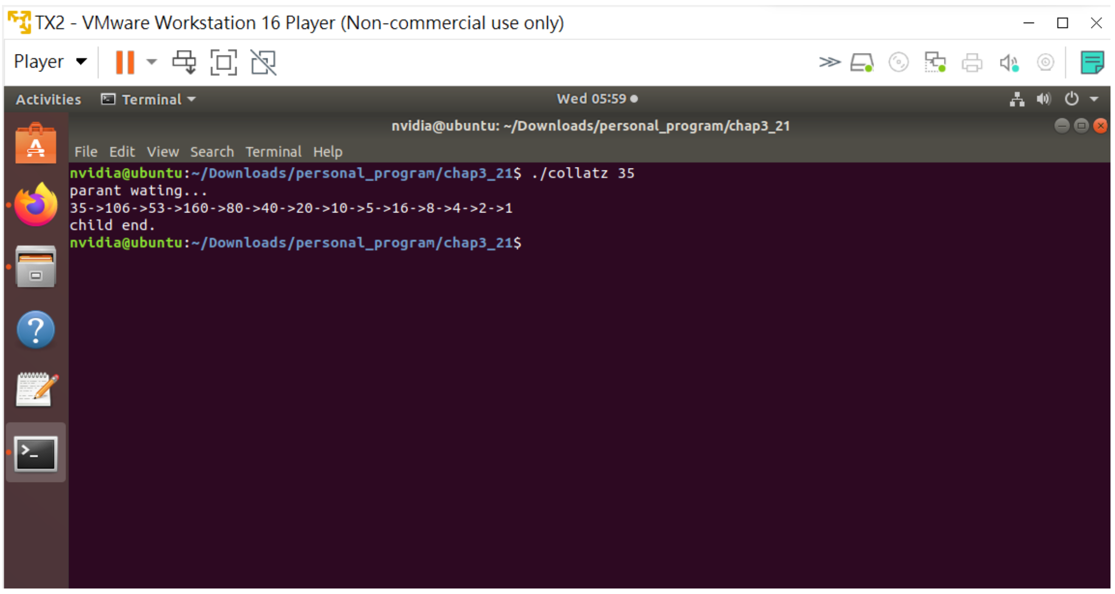
n= n/2, if n is even

n= 3*n+1, if n is odd

The conjecture states that when this algorithm is continually applied, all positive integers will eventually reach 1.For example, if n=35, the sequence is: 35, 106, 53, 160, 80, 40, 20, 10, 5, 16, 8, 4, 2, 1.

- Write a C program using the fork() system call that generates this sequence in the child process. The starting number will be provided from the command line.
  - For example, if 8 is passed as a parameter on the command line, the child process will output 8, 4, 2, 1.
- Because the parent and child processes have their own copies of the data, it will be necessary for the child to output the sequence.
- Have the parent invoke the wait() call to wait for the child process to complete before exiting the program.
- Perform necessary error checking to ensure that a positive integer is passed on the command line.

```
./collatz <parameter>
```


TX2 - VMware Workstation 16 Player (Non-commercial use only)
Player
Activities
Terminal
Wed 05:59
nvidia@ubuntu: ~/Downloads/personal_program/chap3_21
File Edit View Search Terminal Help

```
nvidia@ubuntu:~/Downloads/personal_program/chap3_21$ ./collatz 35
parant wating...
35->106->53->160->80->40->20->10->5->16->8->4->2->1
child end.
nvidia@ubuntu:~/Downloads/personal_program/chap3_21$
```

- [optional] (3.27**): Design a file-copying program named filecopy using ordinary pipes.

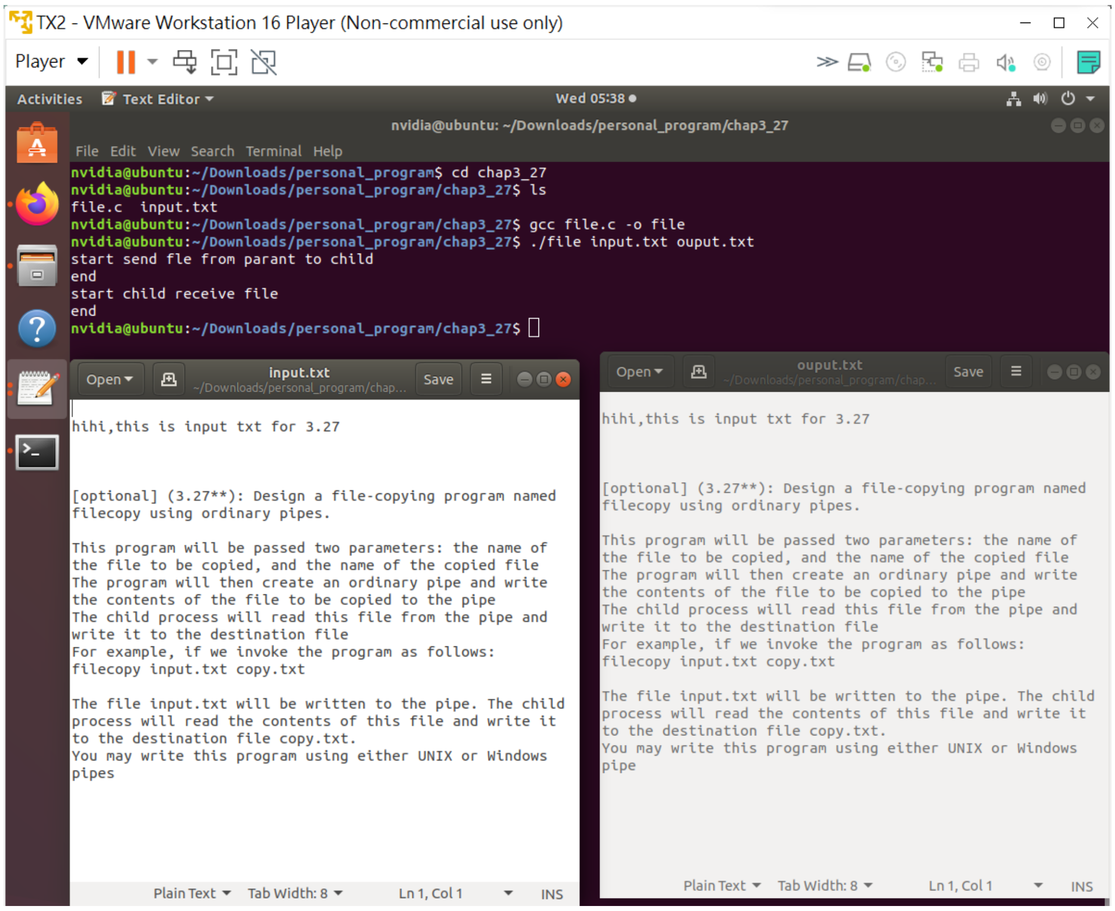
  - This program will be passed two parameters: the name of the file to be copied, and the name of the copied file
  - The program will then create an ordinary pipe and write the contents of the file to be copied to the pipe
  - The child process will read this file from the pipe and write it to the destination file

- For example, if we invoke the program as follows:

  filecopy input.txt copy.txt

  - The file input.txt will be written to the pipe. The child process will read the contents of this file and write it to the destination file copy.txt.
- You may write this program using either UNIX or Windows pipes.

```
./file input.txt <copy file name>
```


TX2 - VMware Workstation 16 Player (Non-commercial use only)
Player
Activities
Text Editor
Wed 05:38
nvidia@ubuntu: ~/Downloads/personal_program/chap3_27
File Edit View Search Terminal Help

```
nvidia@ubuntu:~/Downloads/personal_program$ cd chap3_27
nvidia@ubuntu:~/Downloads/personal_program/chap3_27$ ls
file.c  input.txt
nvidia@ubuntu:~/Downloads/personal_program/chap3_27$ gcc file.c -o file
nvidia@ubuntu:~/Downloads/personal_program/chap3_27$ ./file input.txt ouput.txt
start send fle from parant to child
end
start child receive file
end
nvidia@ubuntu:~/Downloads/personal_program/chap3_27$ 
```

```
hihi,this is input txt for 3.27


[optional] (3.27**): Design a file-copying program named
filecopy using ordinary pipes.

This program will be passed two parameters: the name of
the file to be copied, and the name of the copied file
The program will then create an ordinary pipe and write
the contents of the file to be copied to the pipe
The child process will read this file from the pipe and
write it to the destination file
For example, if we invoke the program as follows:
filecopy input.txt copy.txt

The file input.txt will be written to the pipe. The child
process will read the contents of this file and write it
to the destination file copy.txt.
You may write this program using either UNIX or Windows
pipes
```

```
hihi,this is input txt for 3.27


[optional] (3.27**): Design a file-copying program named
filecopy using ordinary pipes.

This program will be passed two parameters: the name of
the file to be copied, and the name of the copied file
The program will then create an ordinary pipe and write
the contents of the file to be copied to the pipe
The child process will read this file from the pipe and
write it to the destination file
For example, if we invoke the program as follows:
filecopy input.txt copy.txt

The file input.txt will be written to the pipe. The child
process will read the contents of this file and write it
to the destination file copy.txt.
You may write this program using either UNIX or Windows
pipe
```