Sharif-OS-Lab / summer1403-4-99101087_99100422

Code | Issues 1 | Pull requests | Actions | Projects | Security | Insights

Edit | New issue | Jump to bottom

Session 4 Report #1

Open | 31 tasks done | **BozorgmehrZia** opened this issue last week · 0 comments

Labels: documentation

BozorgmehrZia commented last week

Team Name: `99101087-99100422`

Student Name of member 1: `AmirReza Azari`
Student No. of member 1: `99101087`

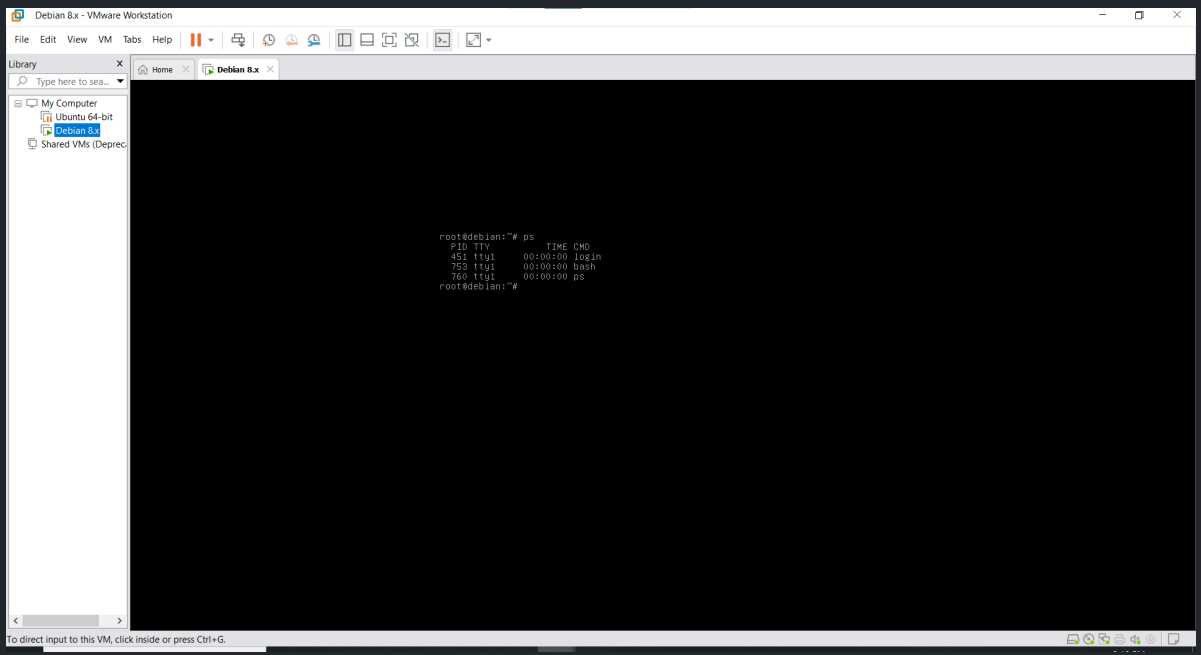
Student Name of member 2: `Bozorgmehr Zia`
Student No. of member 2: `99100422`

- [x] Read Session Contents.

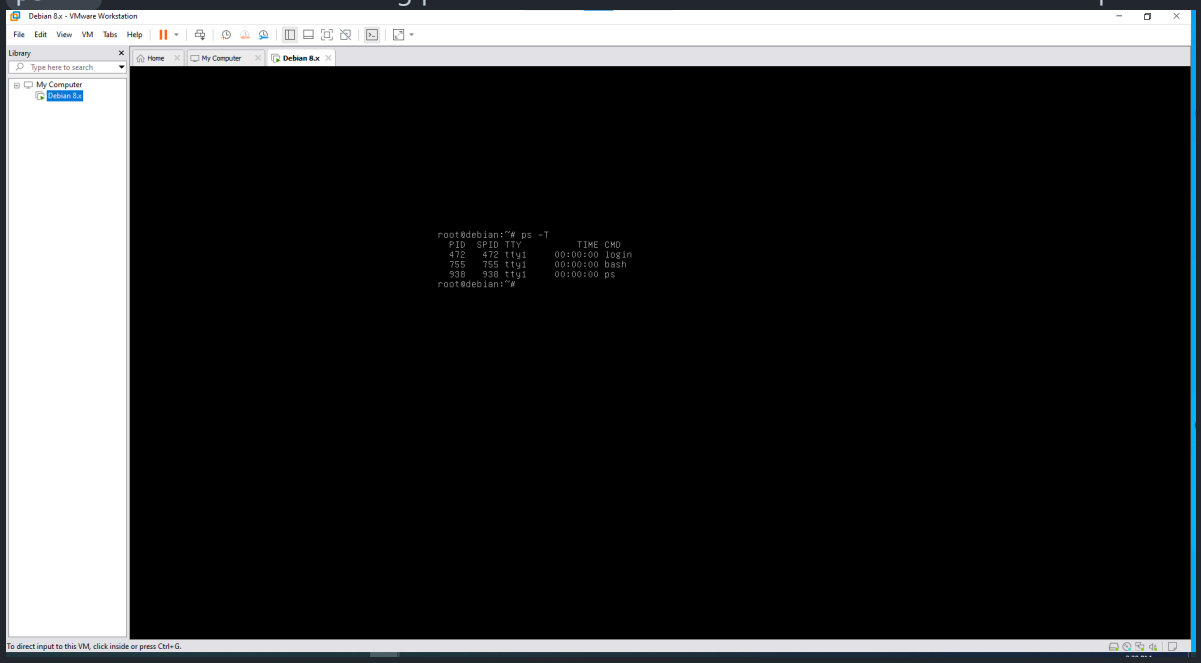
Section 4.4.1

- [x] Investigate the ps command
  `ps` command is for showing processes running in the current shell session. The output is:

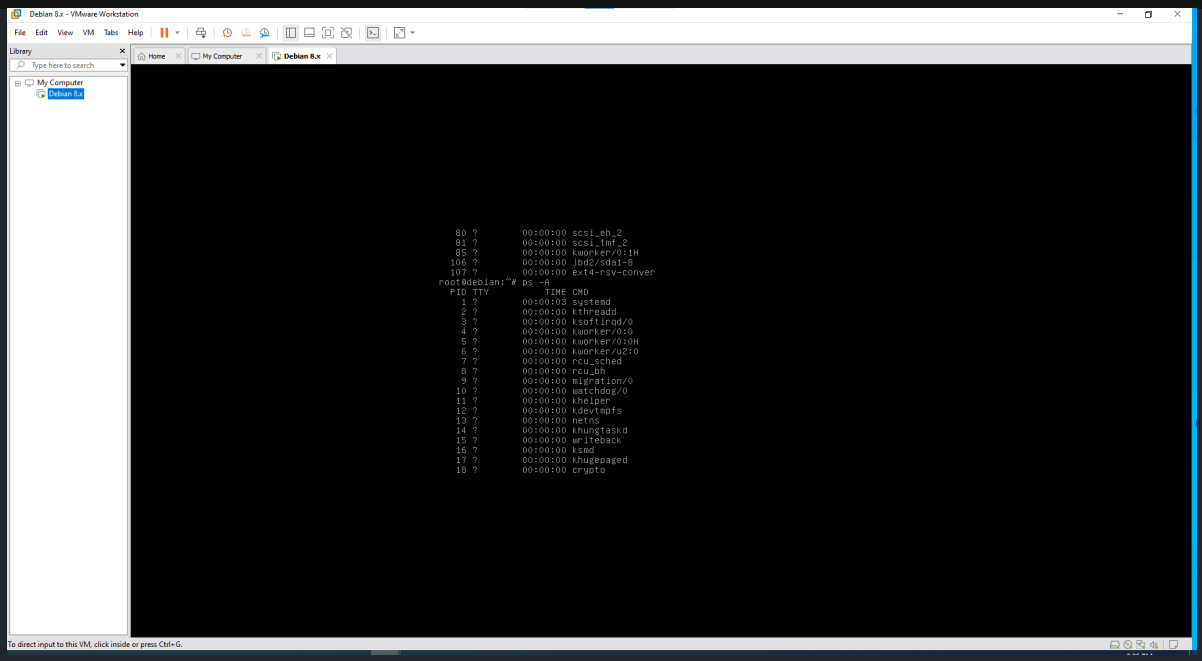
- [x]



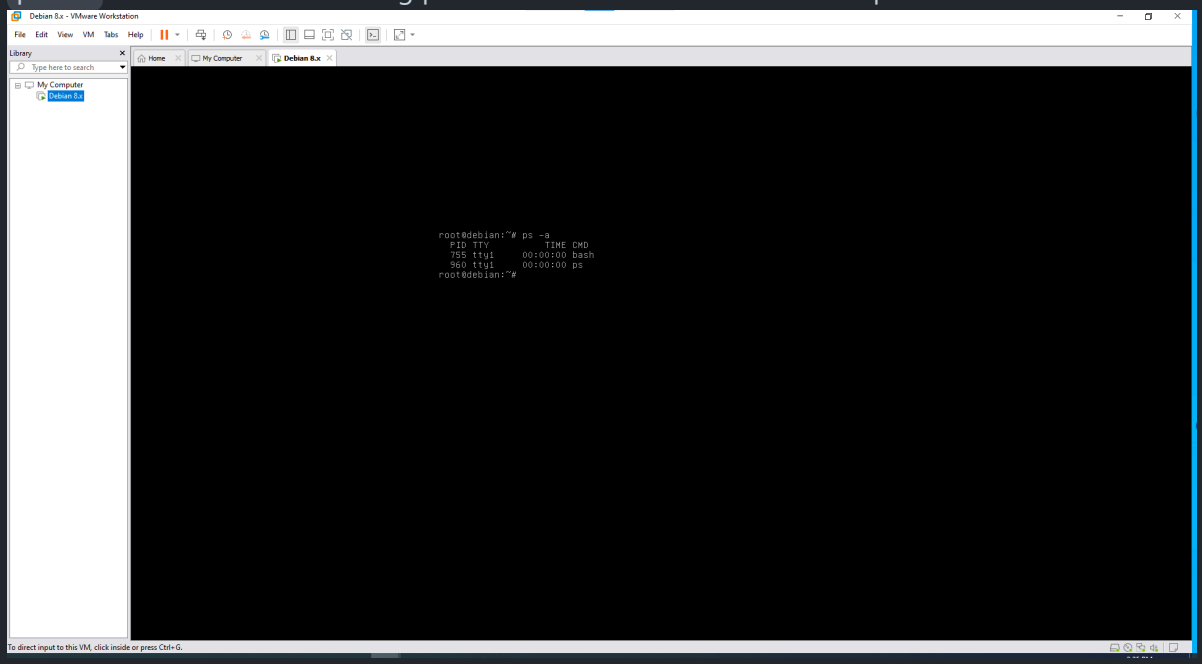
`ps -T` command is for showing processes associated with the current terminal. The output is:



`ps -A` command is for showing all running processes on the system. The output is:

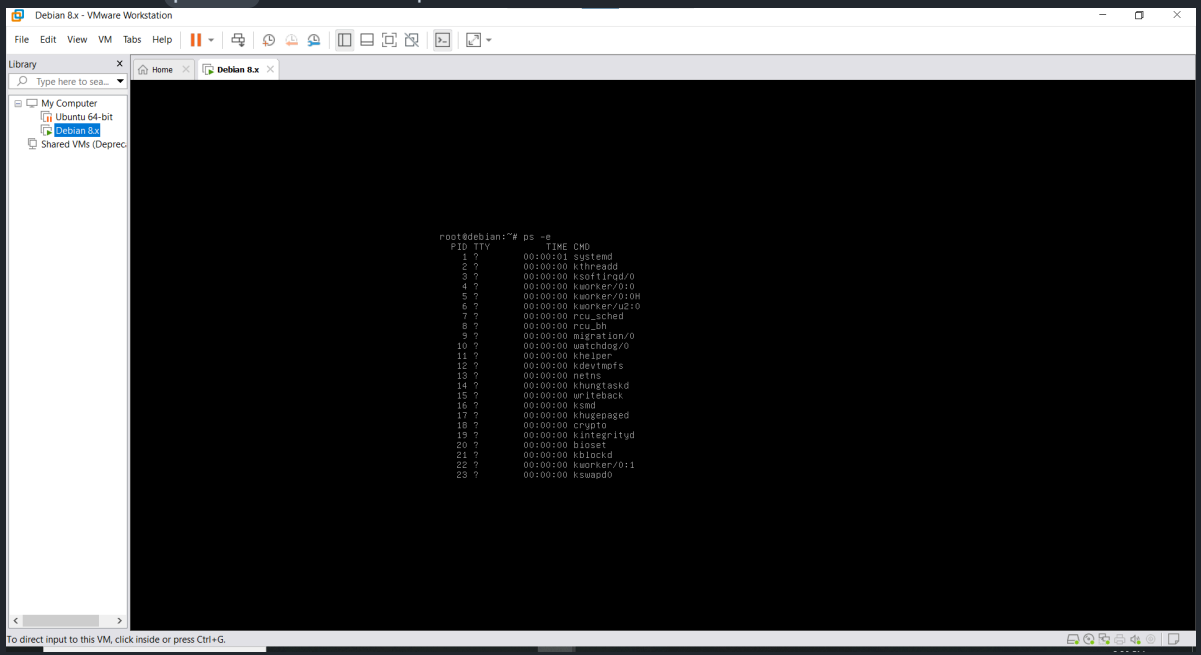


`ps -a` command is for showing processes not for a terminal. The output is:

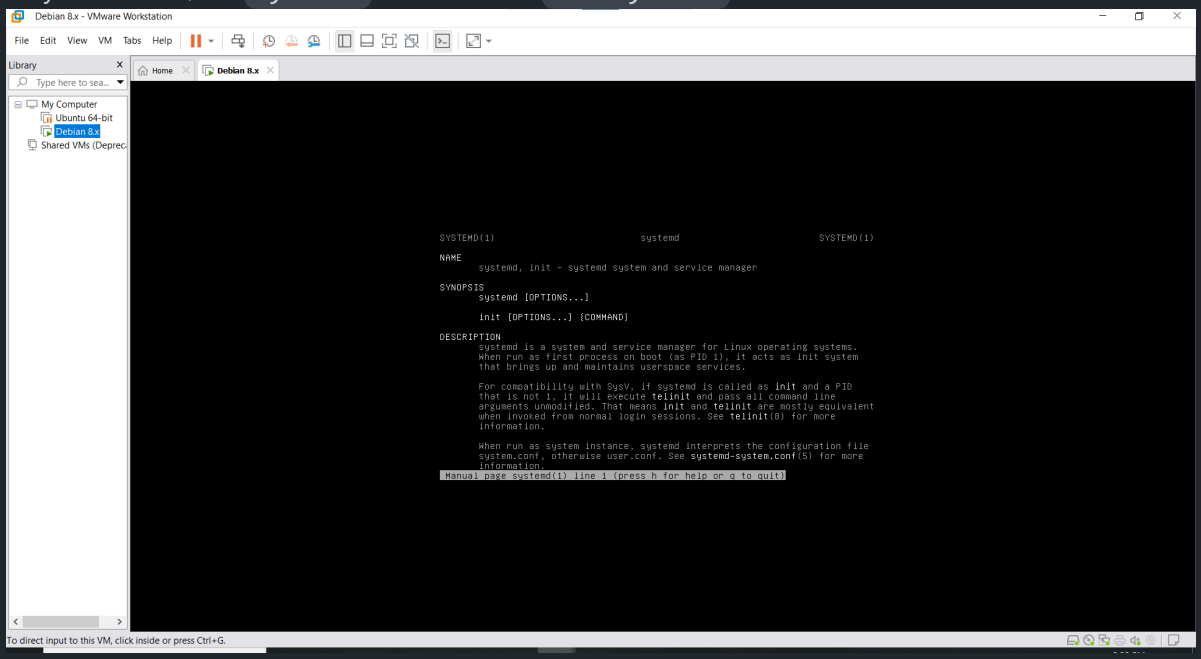


- [x] Infromation about processes with PID = 1

- [x] First, we run `ps -e` to see what process has PID 1:



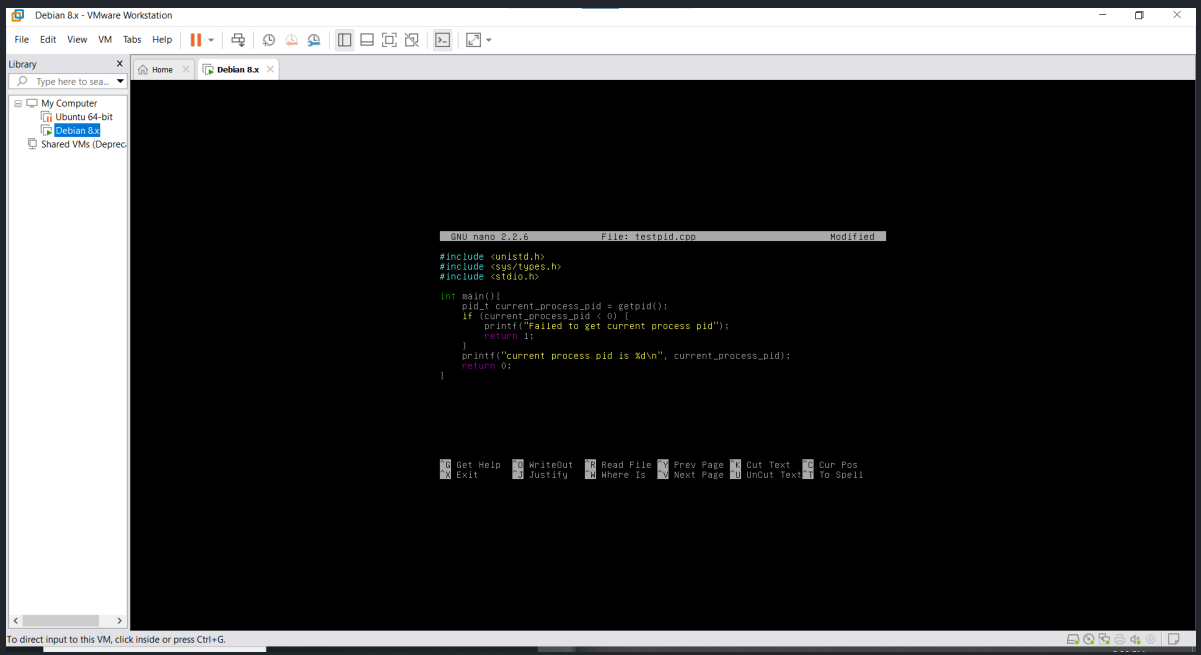
As you can see, it's `systemd`. Now we run `man systemd`:



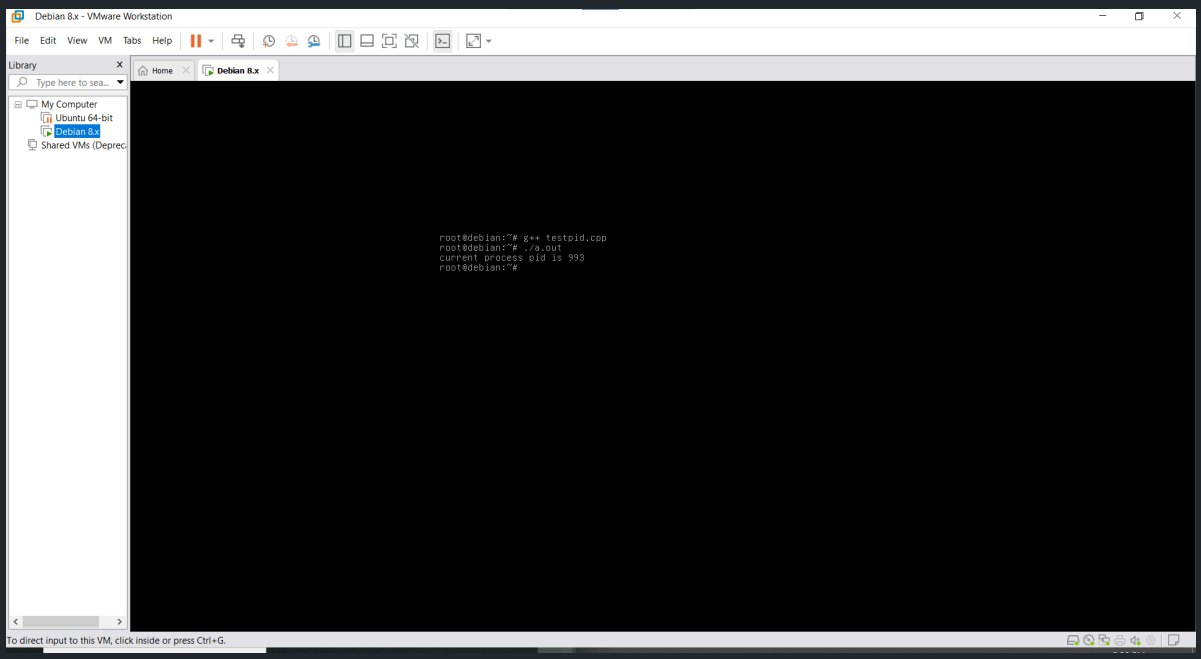
According to the description, this process is system and service manager in linux, which runs as first process on boot with PID 1, and it acts as init system that brings up and maintains userspace services. Separate instances are started for logged-in users to start their services.

- [x] Program using getpid

- [x]



Result:

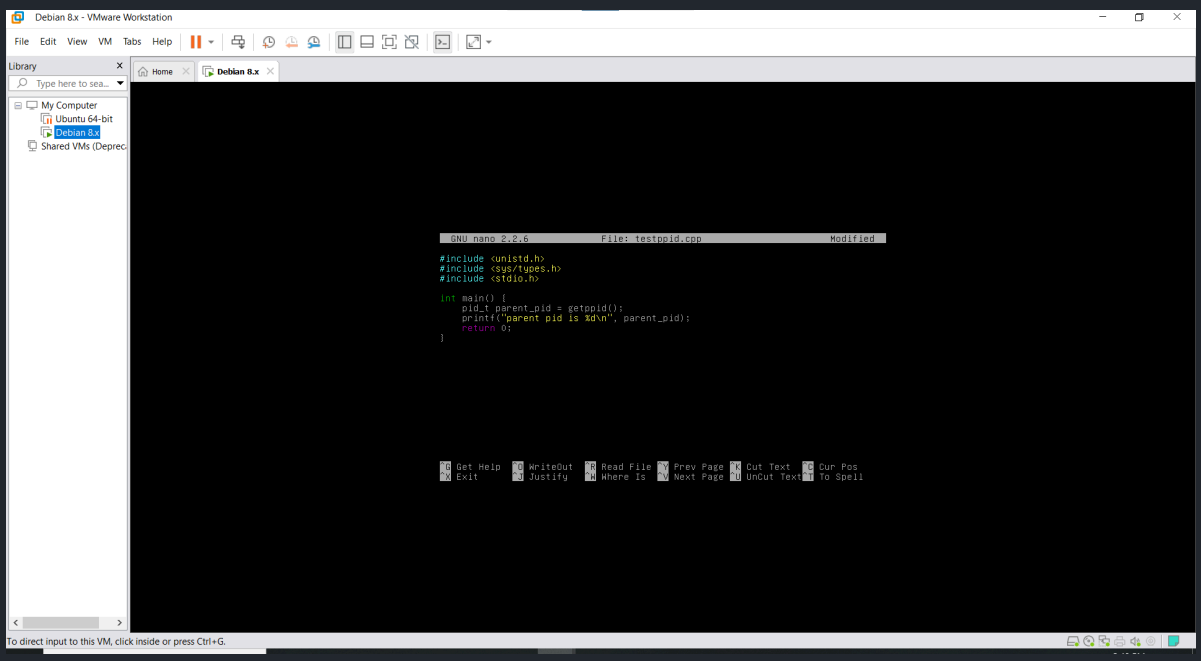


The code is in file `4-4-1.cpp`.

Section 4.4.2

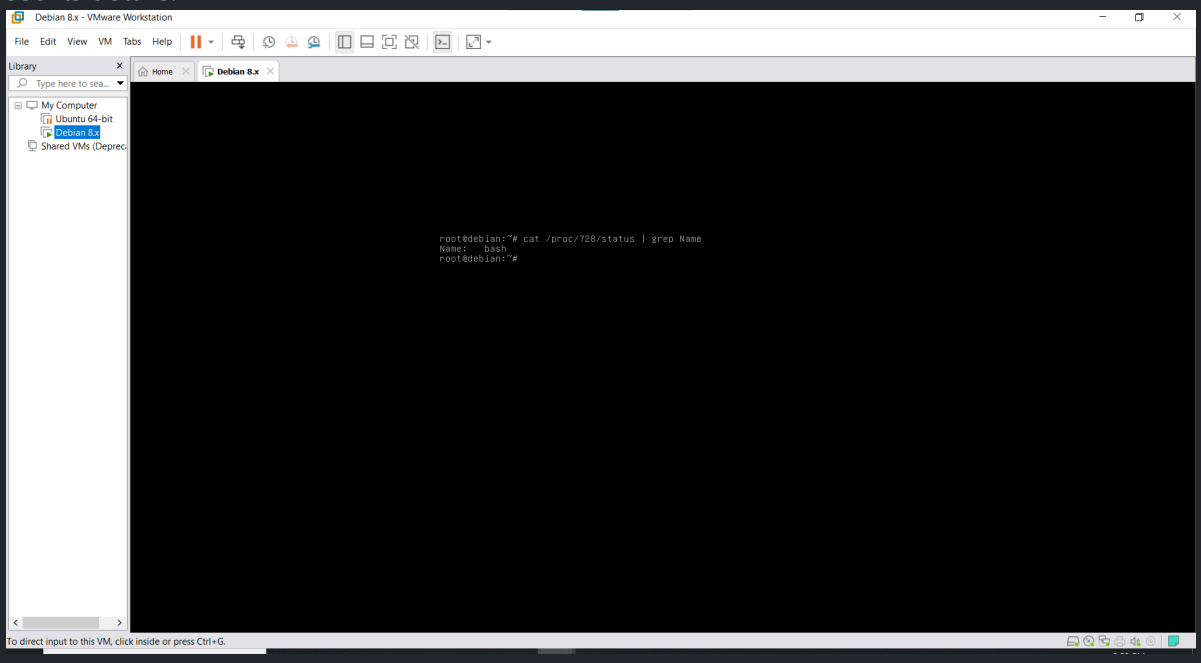
- [x] Program using getppid

- [x]



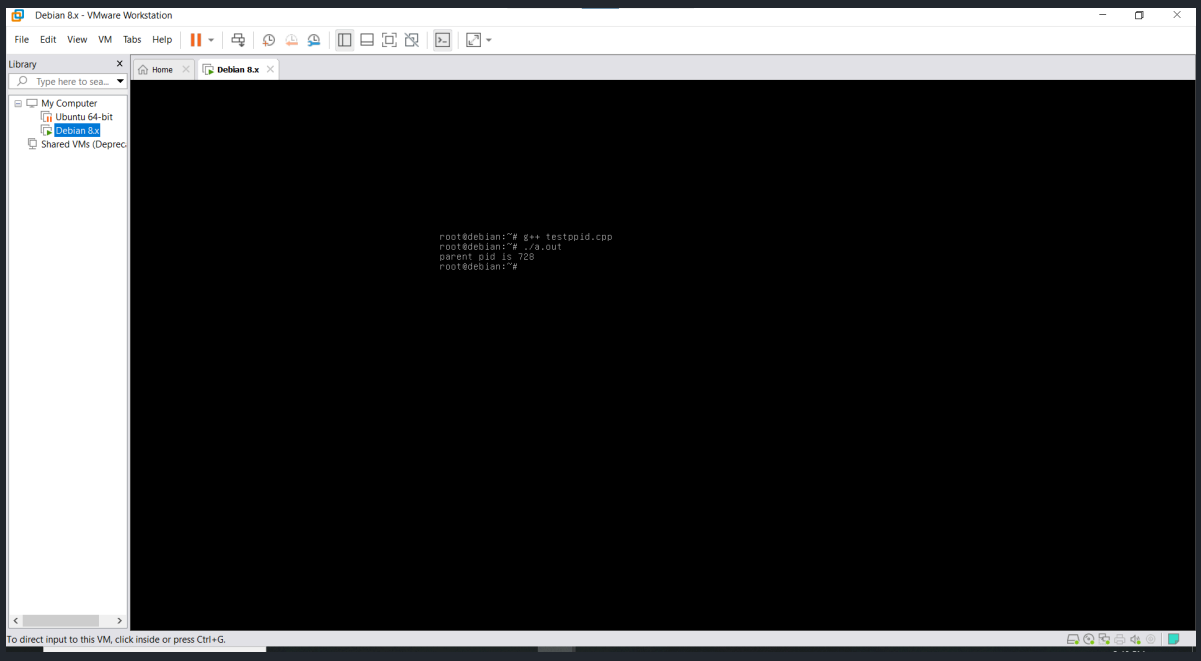
This code is in file `4-4-2-1.cpp`.

- [x] If we run the code, the result is 728. If we run `cat /proc/728/status | grep Name`, we will see its details:



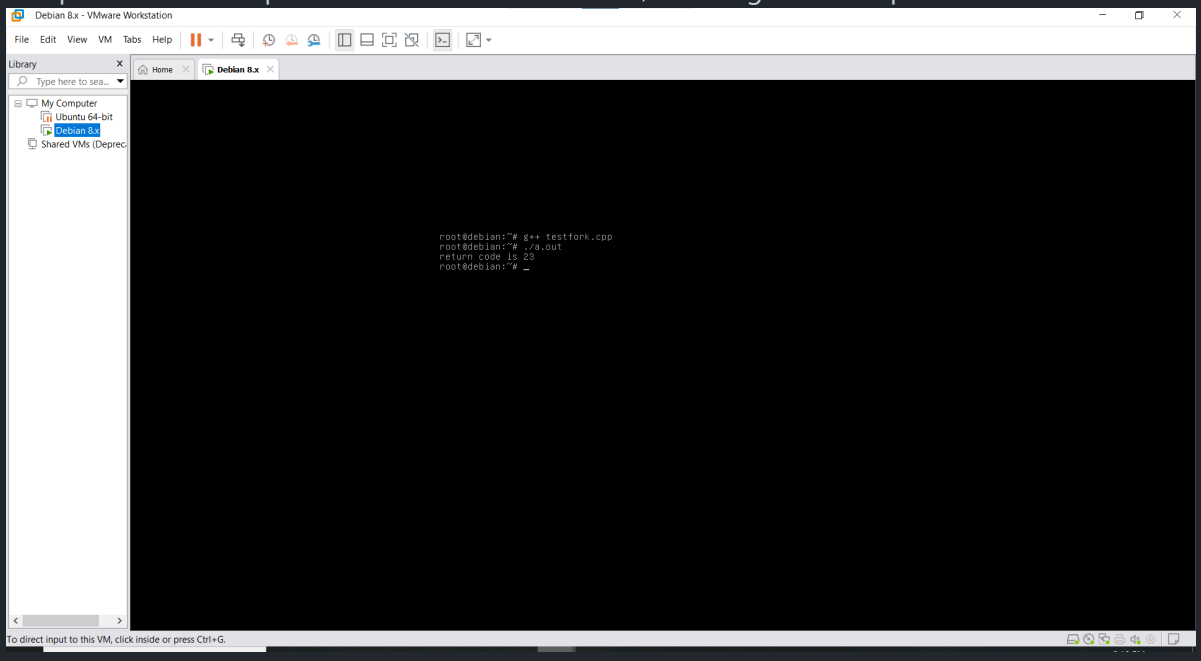
As you see, it's `bash` that runs our code and refers to an instance of running the Bash shell, which is a widely-used
command interpreter on Linux. When we open a terminal on a Linux system, a Bash process is typically started. This process runs interactively, reads commands from the user and executes them. The result of above code is:

- [x]

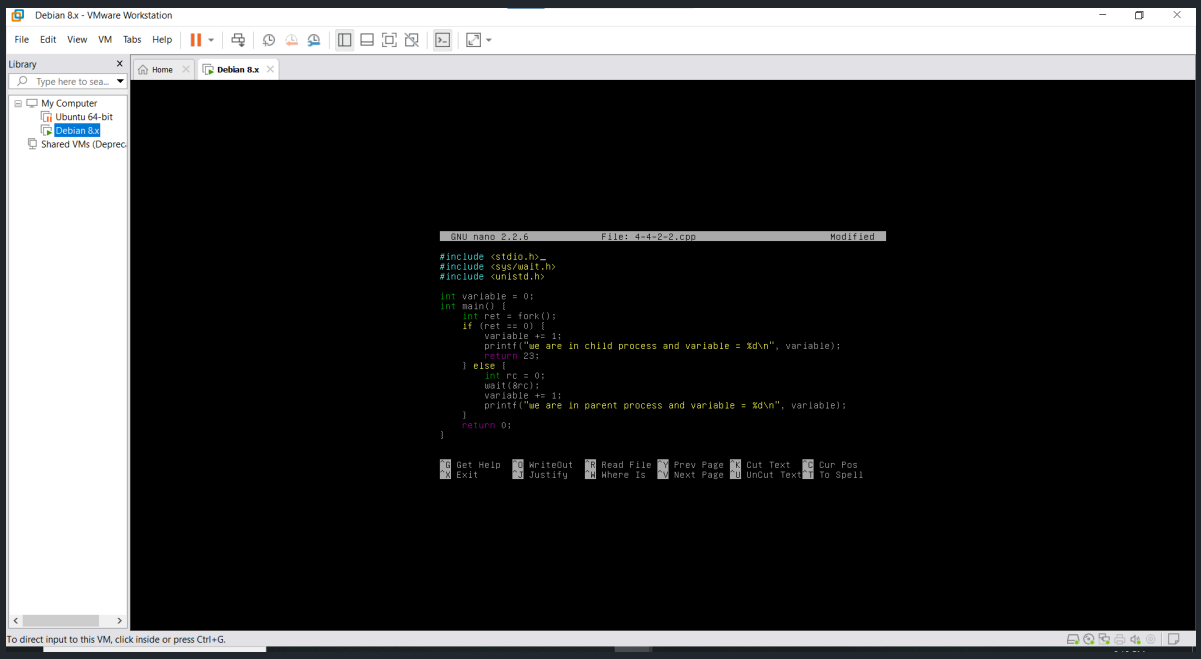


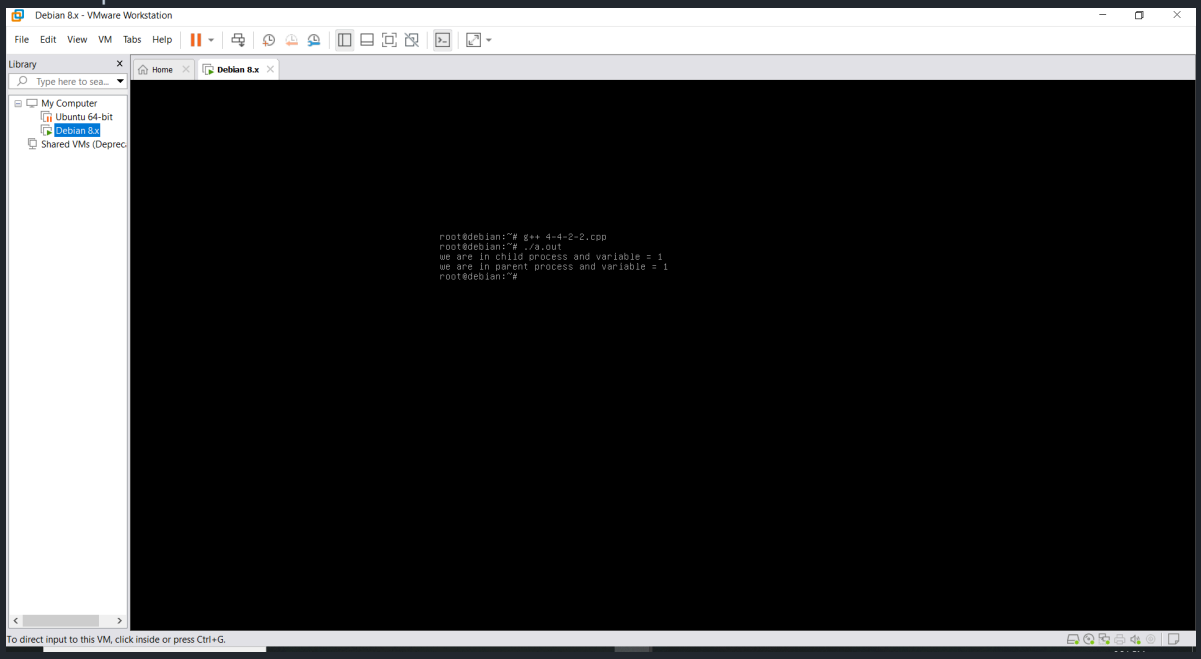
- [x] Describe the C program (fork program)

  - [x] This code first creates a child process with calling `fork()`, which is a system call that is used to create a new process by duplicating the calling process. This system call returns 0 to the child process, the child's PID (process ID) to the parent process, and -1 if the creation of the child process failed. So the child process will execute the lines `// ...` and `return 23;`, and passes a status back to its parent. The parent process will execute the `else` part and waits for the child process to finish using the wait function. The `wait()` function suspends execution of the calling process until one of its children terminates. The exit status of the child process is stored in rc. `WEXITSTATUS(rc)` is a macro that extracts the exit status from rc. Then the parent process prints the child's exit status. The exit status is the return value 23 from the child process. The printf statement prints this value to the console, resulting in the output:



- [x] Program showing that memory of the parent and the child is seperate

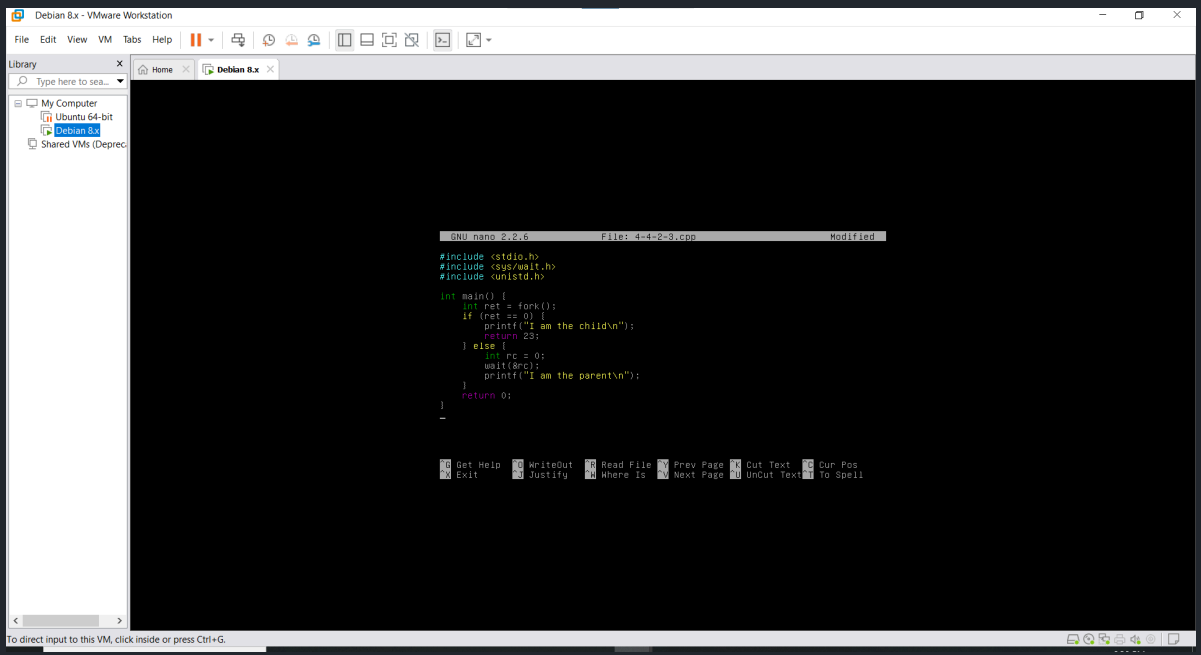
  - [x] 

    We use a variable named `variable`, which is a global variable. We know that if the variable in parent is increased by 1, the next increase should show 2 but our output shows that it would remain 1 which means that the parent and child process have different memories.
    The output of code is:
    

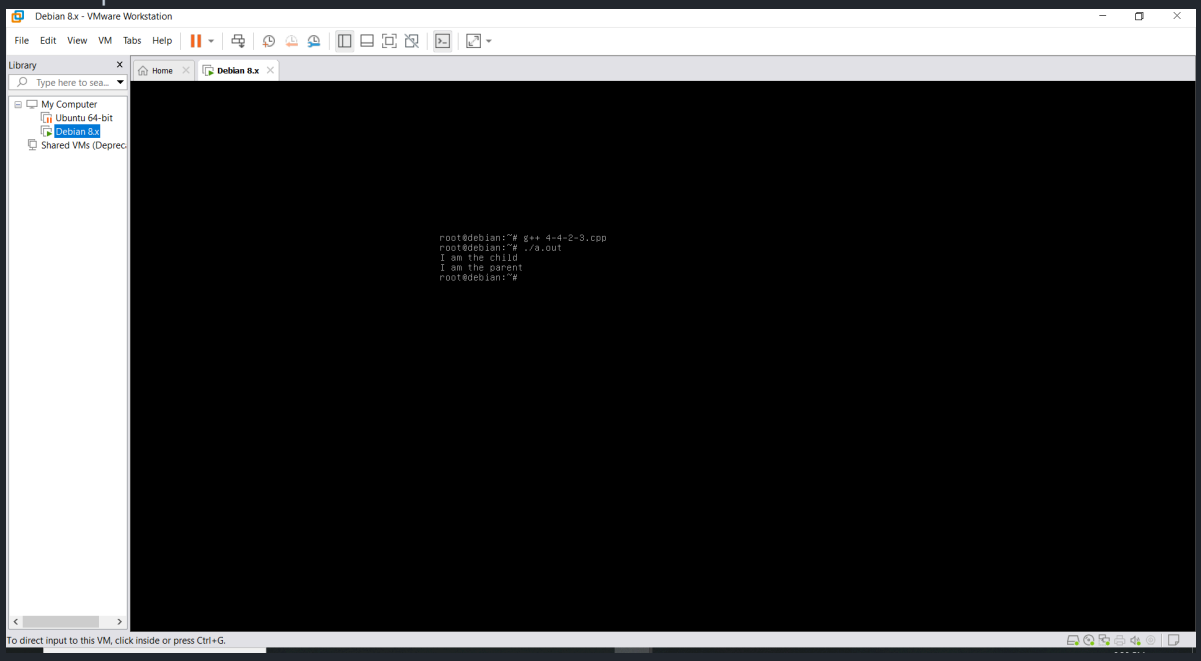
    The code is in file `4-4-2-2.cpp`.

- [x] Program printing different messages for parent and child process

- [x]



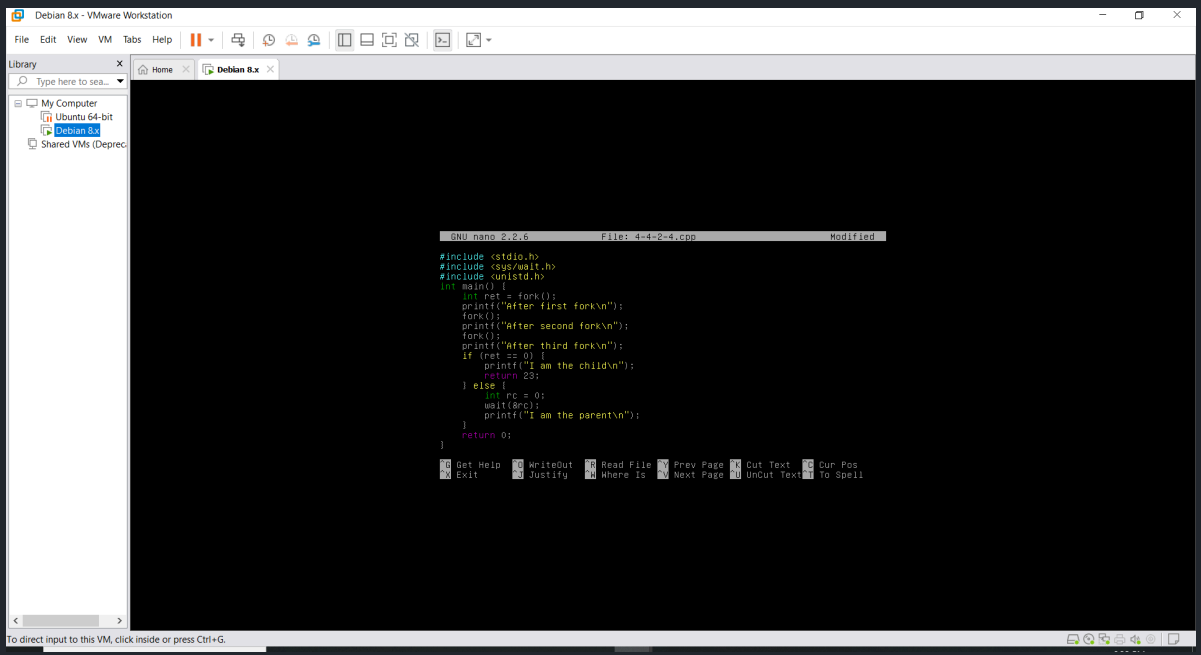
The output is:



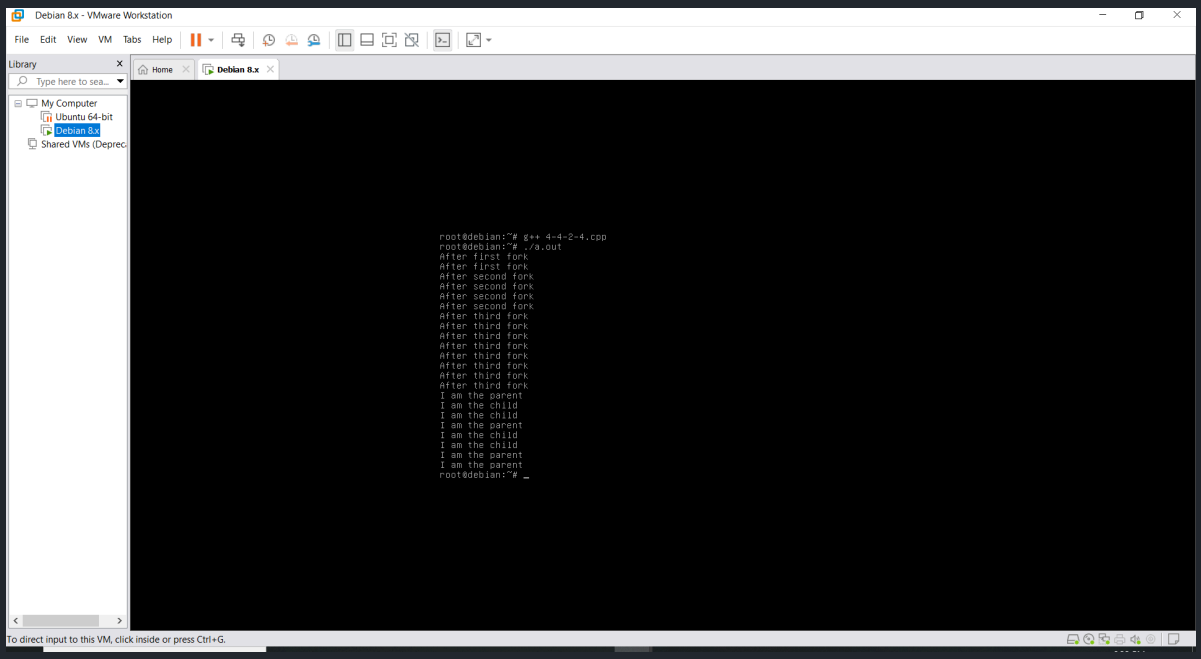
The code is in file `4-4-2-3.cpp`.

- [x] Program for the last task of this section

- [x]



- [x]



After the first fork, we have 2 processes, So the phrase `After first fork` is printed 2 times. After the second fork, we have 4 processes (each of the first 2 processes creates another process), So the phrase `After second fork` is printed 4 times. After the third fork we have 8 processes, So the phrase `After third fork` is printed 8 times.

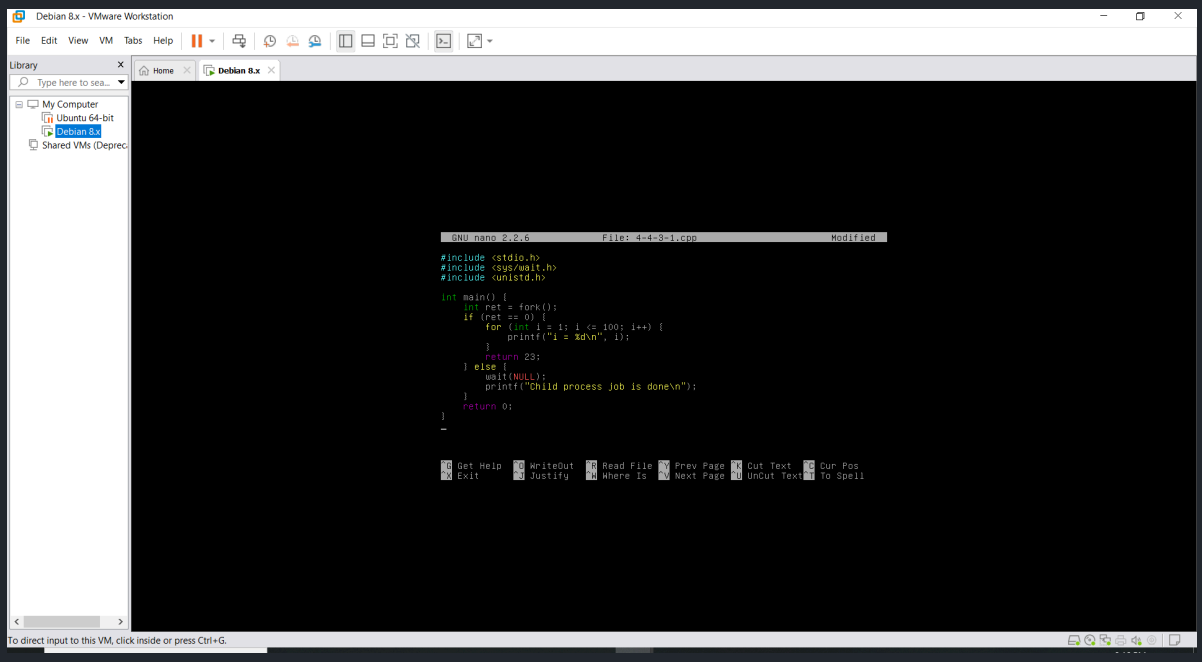
We know that `ret` has a common value among all processes, so in half of the processes it is 0 and in other half is not 0. So the phrases `I am the child` and `I am the parent` each one is printed 4 times.

The code is in file `4-4-2-4.cpp`.

Section 4.4.3

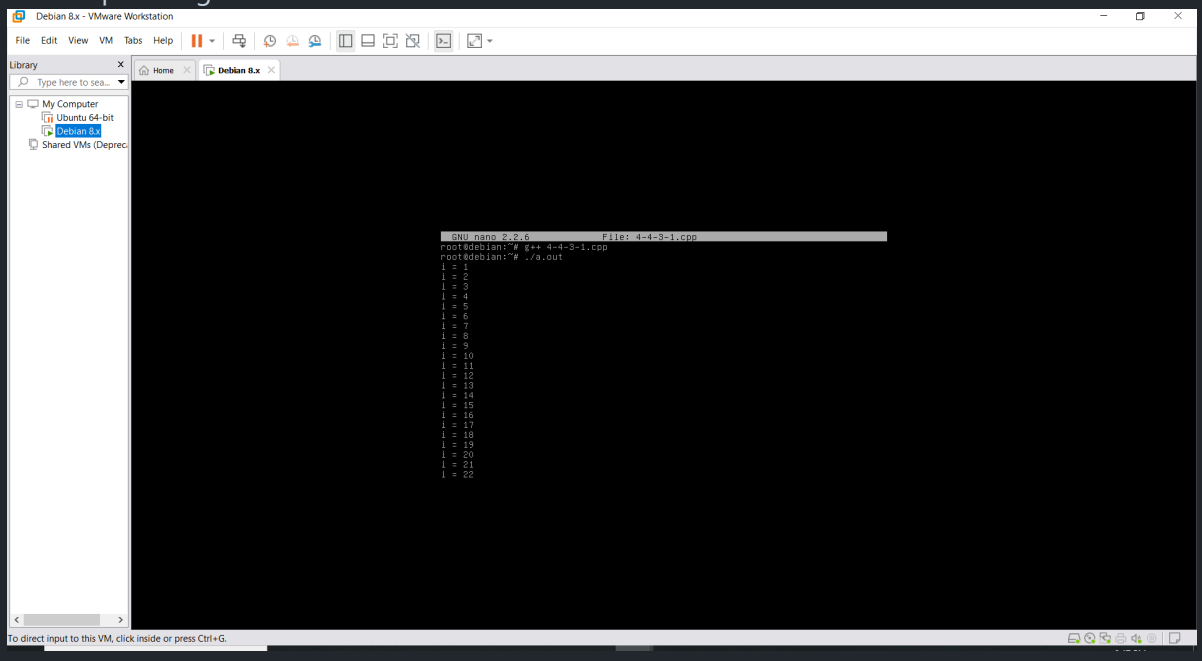
- [x] Program using `wait` and counting from 1 to 100

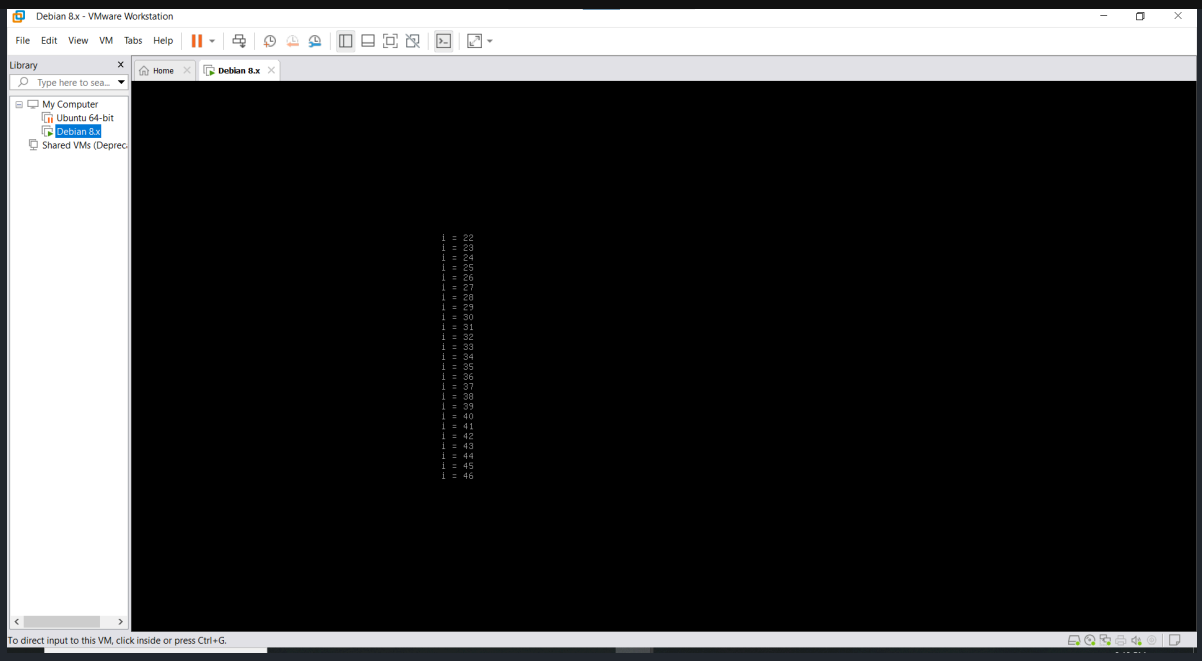
  - [x]

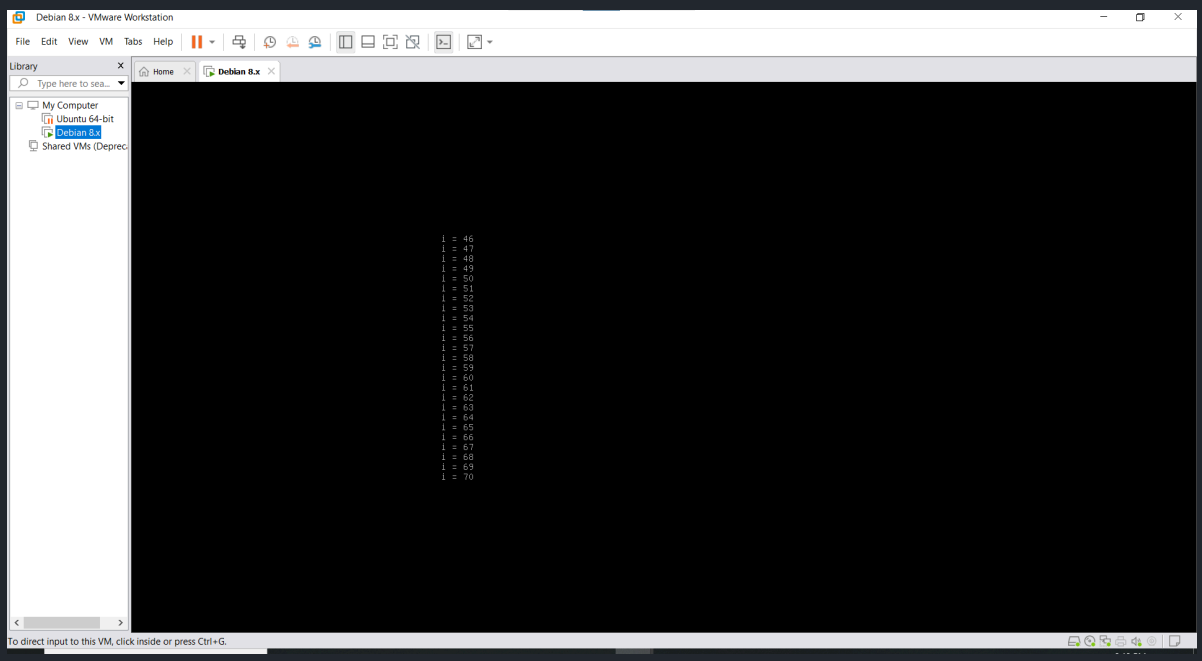


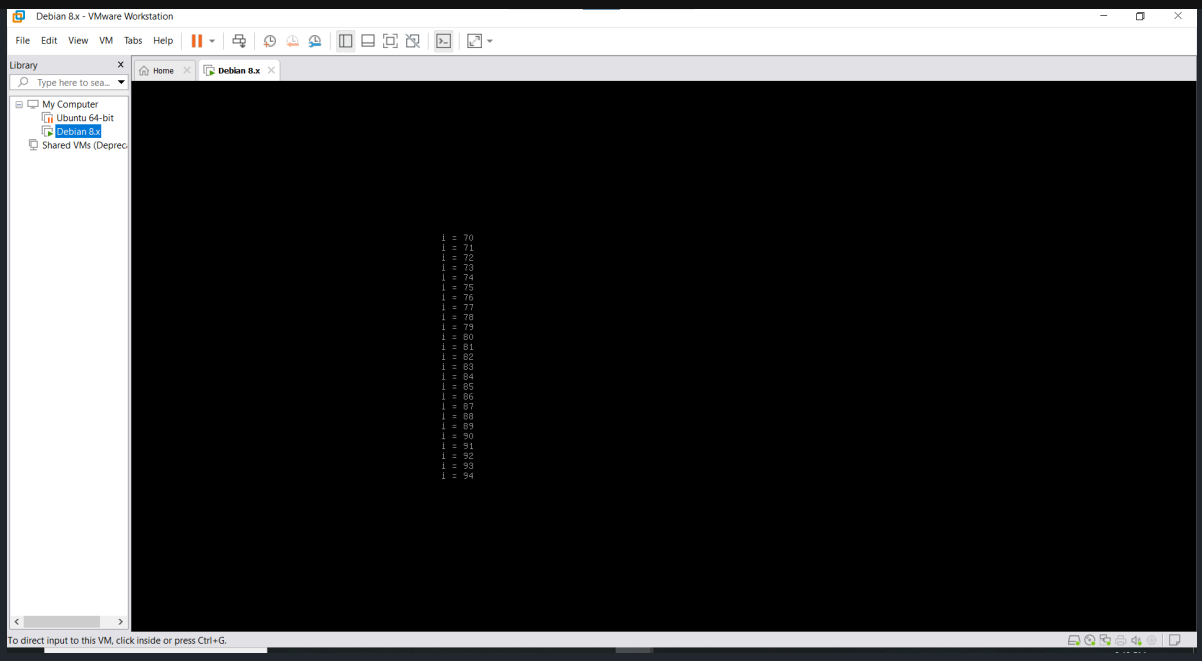
The code is in file `4-4-3-1.cpp`.

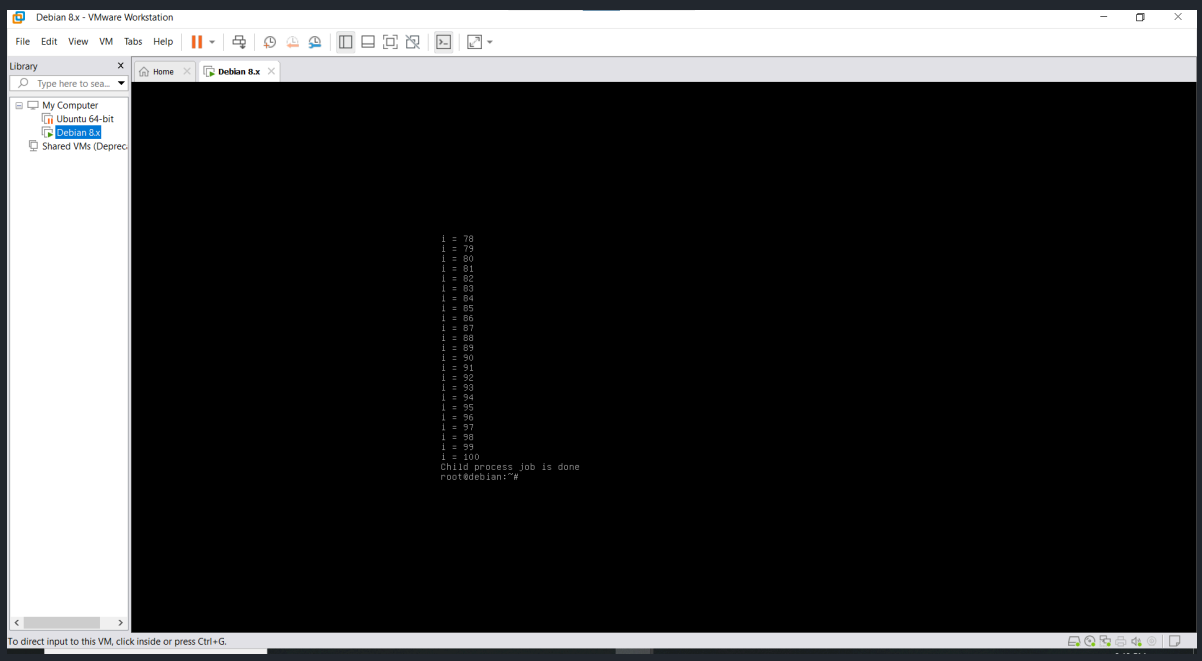
The output images are:




Debian 8.x - VMware Workstation
File Edit View VM Tabs Help
Library
Type here to sea...
My Computer
Ubuntu 64-bit
Debian 8.x
Shared VMs (Deprec
Home
Debian 8.x
i = 22
i = 23
i = 24
i = 25
i = 26
i = 27
i = 28
i = 29
i = 30
i = 31
i = 32
i = 33
i = 34
i = 35
i = 36
i = 37
i = 38
i = 39
i = 40
i = 41
i = 42
i = 43
i = 44
i = 45
i = 46
To direct input to this VM, click inside or press Ctrl+G.


Debian 8.x - VMware Workstation
File Edit View VM Tabs Help
Library
Type here to sea...
My Computer
Ubuntu 64-bit
Debian 8.x
Shared VMs (Deprec
Home
Debian 8.x
i = 46
i = 47
i = 48
i = 49
i = 50
i = 51
i = 52
i = 53
i = 54
i = 55
i = 56
i = 57
i = 58
i = 59
i = 60
i = 61
i = 62
i = 63
i = 64
i = 65
i = 66
i = 67
i = 68
i = 69
i = 70
To direct input to this VM, click inside or press Ctrl+G.

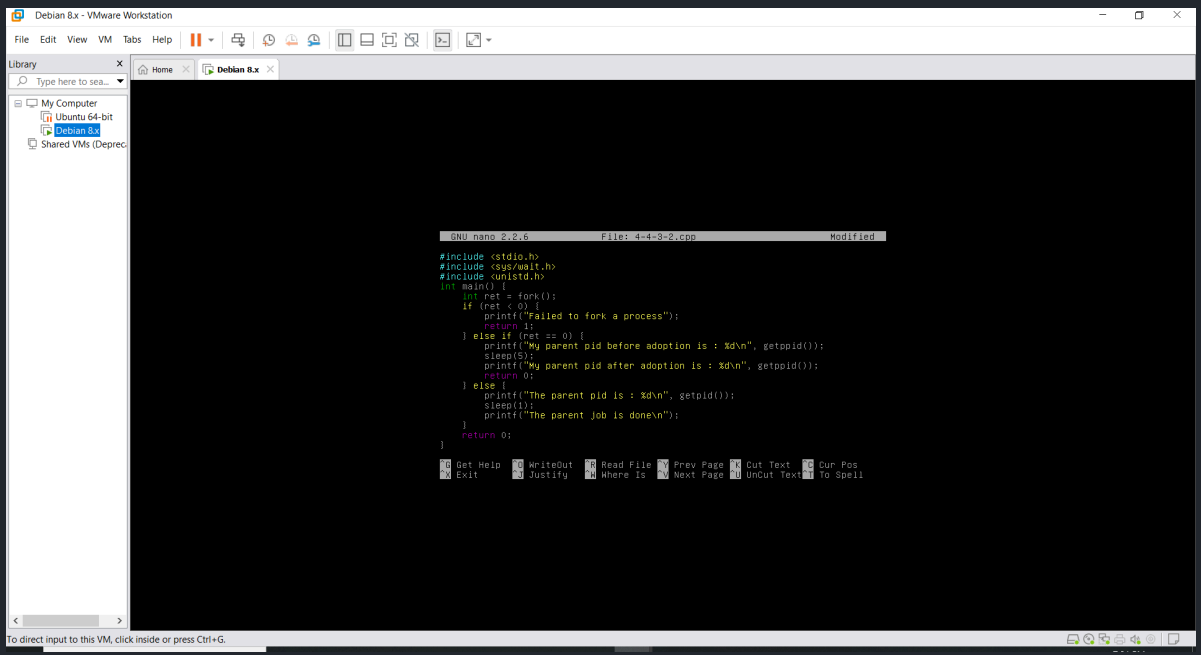




- [x] The first parameter of the `wait()` function is a pointer to an integer where the exit status of the child process will be stored. When `wait(NULL)` is used, it means that the parent process does not care about the exit status of the child process. The parent process is only interested in waiting for any child process to terminate.

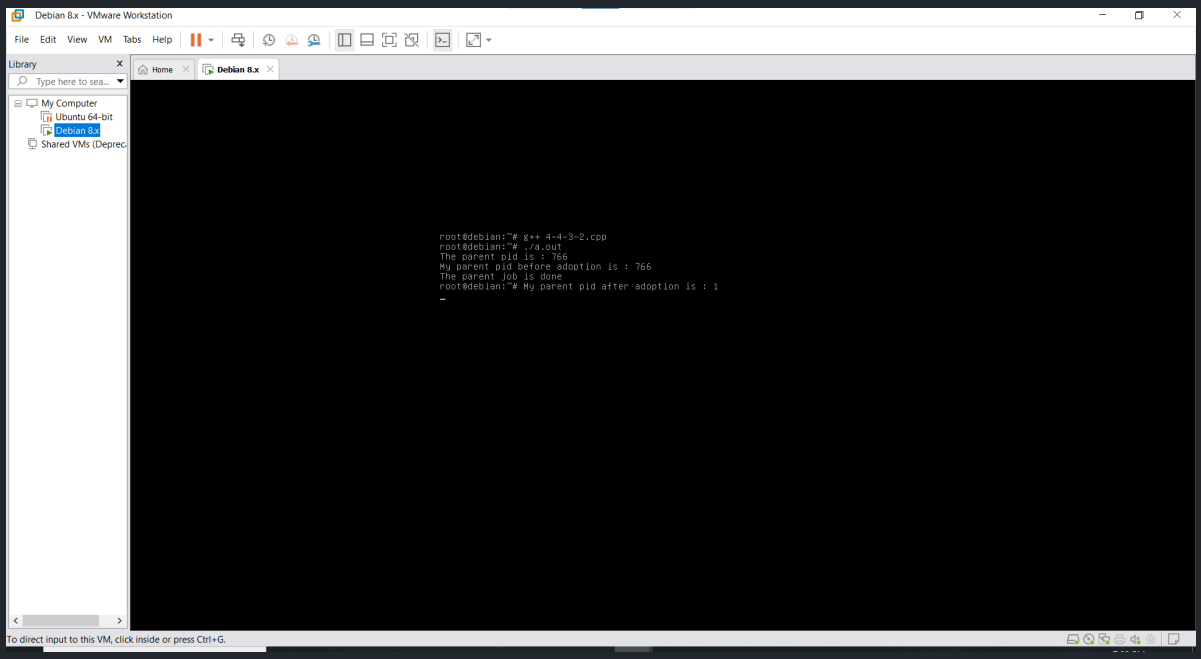
- [x] Program showing process adoption

- [x]



The code is in file `4-4-3-2.cpp`.

- [x]



As you can see, at first the parent pid is 766, then after adoption is 1, which is for the `init` process.

Section 4.4.4

- [x] Describe following commands/APIs:
  The exec family of functions in Linux is used to replace the current process image with a new process image.

  i. `execv`: (execute vector) It replaces the current process with a new process specified by the path and the argument list vector. The prototype is `int execv(const char *path, char *const`

`argv[]);`. This function takes two arguments: the path to the executable file(which must be full path), and a NULL-terminated array of strings representing the argument list available to the new program. The first argument should be the filename associated with the file being executed. The vector in its name refers to the array of arguments. An example is: `char *args[] = {"/bin/ls", "-l", NULL}; execv("/bin/ls", args);`.

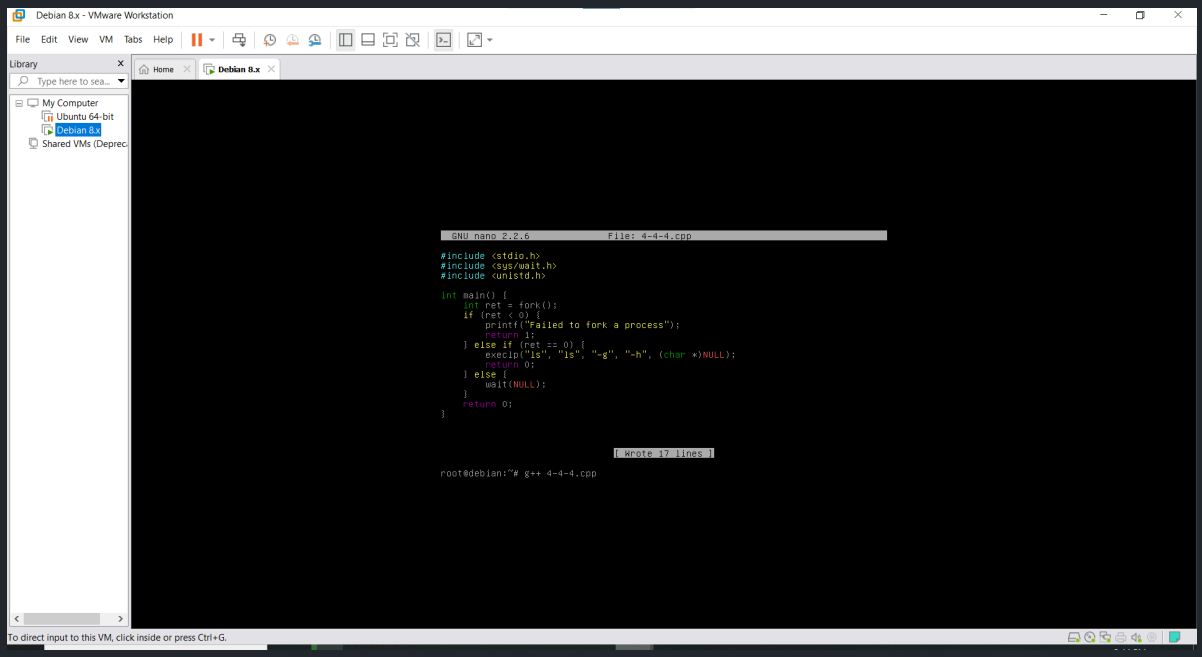
ii. `execl`: (execute list) It is similar to `execv`, but it takes a variable number of arguments. Like `execv`, the first argument is the path to the executable file and must be full path, followed by the individual arguments to the program. The list in its name refers to the variable list of arguments. The prototype is `int execl(const char *path, const char *arg, ...);`. The ... part consists of subsequent arguments representing the argument list available to the new program, ending with a NULL pointer. For example, `execl("/bin/ls", "ls", "-l", (char *)NULL);`.

iii. `execvp`: (execute vector with PATH search) It is like `execv`, but it searches for the file in the directories listed in the PATH environment variable. So the file name doesn't have to be full path, but must be a valid executable file that exists in one of the directories in PATH. Like `execv`, `execvp` takes an array of pointers to null-terminated strings that represent the arguments to the program. The prototype is `int execvp(const char *file, char *const argv[]);`. An example is: `char *args[] = {"ls", "-l", NULL}; execvp("ls", args);`.

iv. `execlp`: (execute list with PATH search) It is like `execl`, but it searches for the file in the directories listed in the PATH environment variable. So the file name doesn't have to be full path, but must be a valid executable file that exists in one of the directories in PATH. Like `execl`, `execlp` takes a variable number of arguments. The prototype is `int execlp(const char *file, const char *arg, ...);`. The ... part consists of subsequent arguments representing the argument list available to the new program, ending with a NULL pointer. A usage of it exists in the `4-4-4.cpp` code.

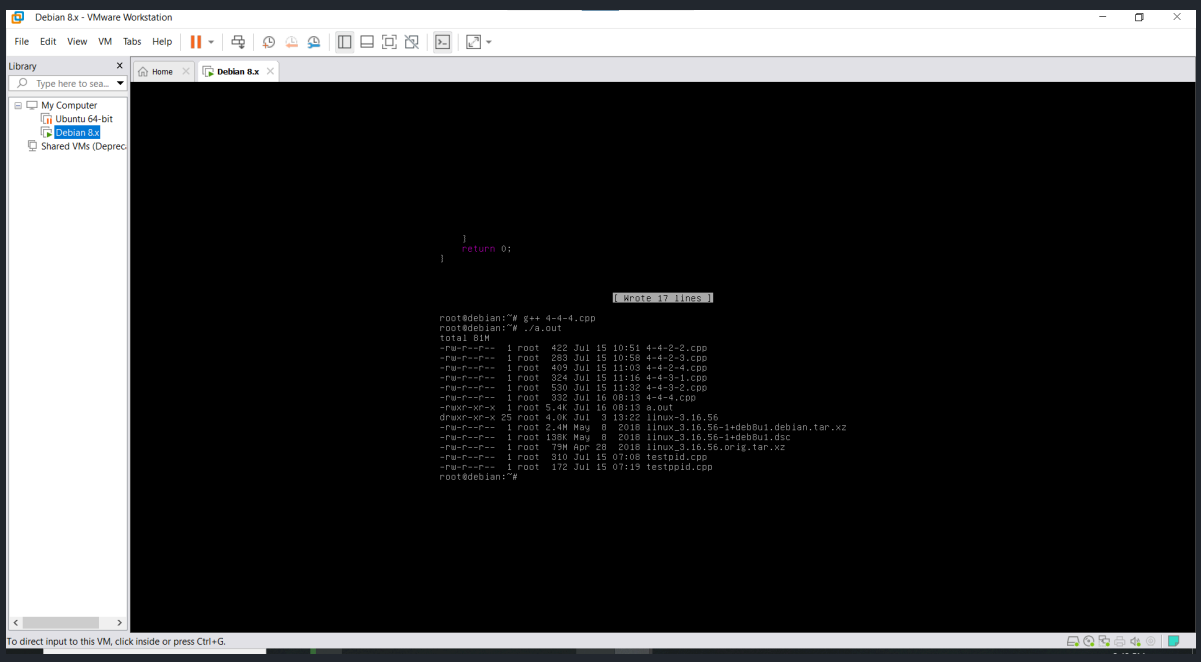
- [x] Program which forks and executues `ls` command

  - [x]



The code is in file `4-4-4.cpp` and I used `execlp` function which is explained above. The output is:

- [x]



Source Code Submission

please submit all your codes in a zip file

- [x] codes.zip

BozorgmehrZia added the documentation label last week

Assignees

No one—assign yourself

Labels

documentation

Projects

None yet

Milestone

No milestone

Development

Create a branch for this issue or link a pull request.

1 participant

Pin issue