



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

☒ Read Session Contents.

## Section 4.4.1

☒ Investigate the ps command

ps command is for showing processes running in the current shell session. The output is:

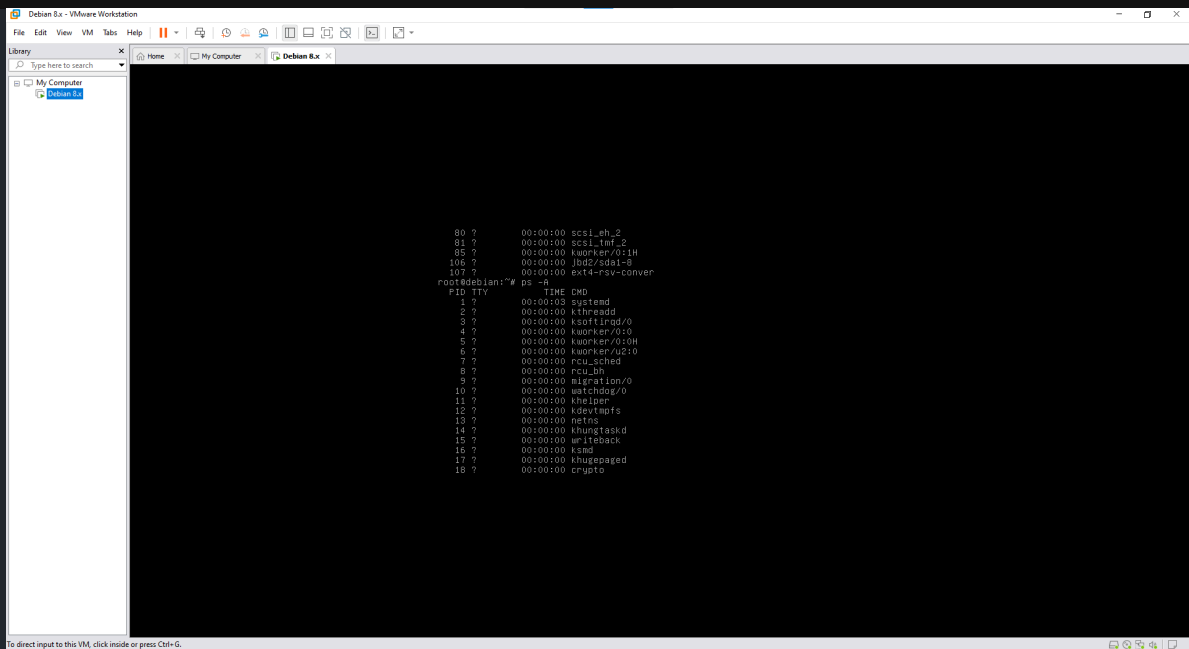


```
root@debian:~# ps
  PID TTY          TIME CMD
  451 ttty1    00:00:00 login
  753 ttty1    00:00:00 bash
  769 ttty1    00:00:00 ps
root@debian:~#
```

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

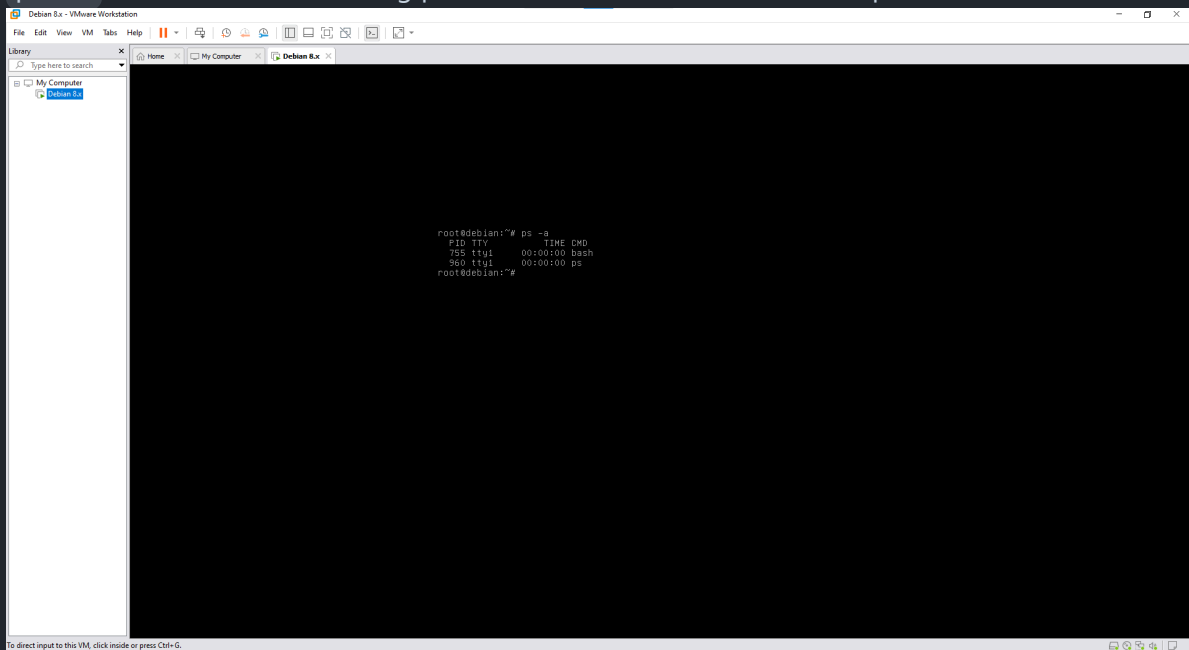
```
root@debian:~# ps -T
  PID SPID TTY          TIME CMD
  472 472 ttty1    00:00:00 login
  755 755 ttty1    00:00:00 bash
  938 938 ttty1    00:00:00 ps
root@debian:~#
```

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



```
80 ?      00:00:00 scsi_ah_2
81 ?      00:00:00 scsi_tm_2
85 ?      00:00:00 kworker/0:1H
106 ?     00:00:00 JBD2/sda1-8
107 ?     00:00:00 ext4-fsv-conver
root@debian:~# ps -a
PID TTY          TIME CMD
1 ?           00:00:00 systemd
2 ?           00:00:00 kthreadd
3 ?           00:00:00 ksoftirqd/0
4 ?           00:00:00 kworker/0:0
5 ?           00:00:00 kworker/0:0H
6 ?           00:00:00 kworker/u210
7 ?           00:00:00 rcu_sched
8 ?           00:00:00 rcu_bh
9 ?           00:00:00 migration/0
10 ?          00:00:00 watchdog/0
11 ?          00:00:00 khelper
12 ?          00:00:00 kdevtmpfs
13 ?          00:00:00 netns
14 ?          00:00:00 kunglaskd
15 ?          00:00:00 w1teback
16 ?          00:00:00 ksm
17 ?          00:00:00 khugepaged
18 ?          00:00:00 crypto
```

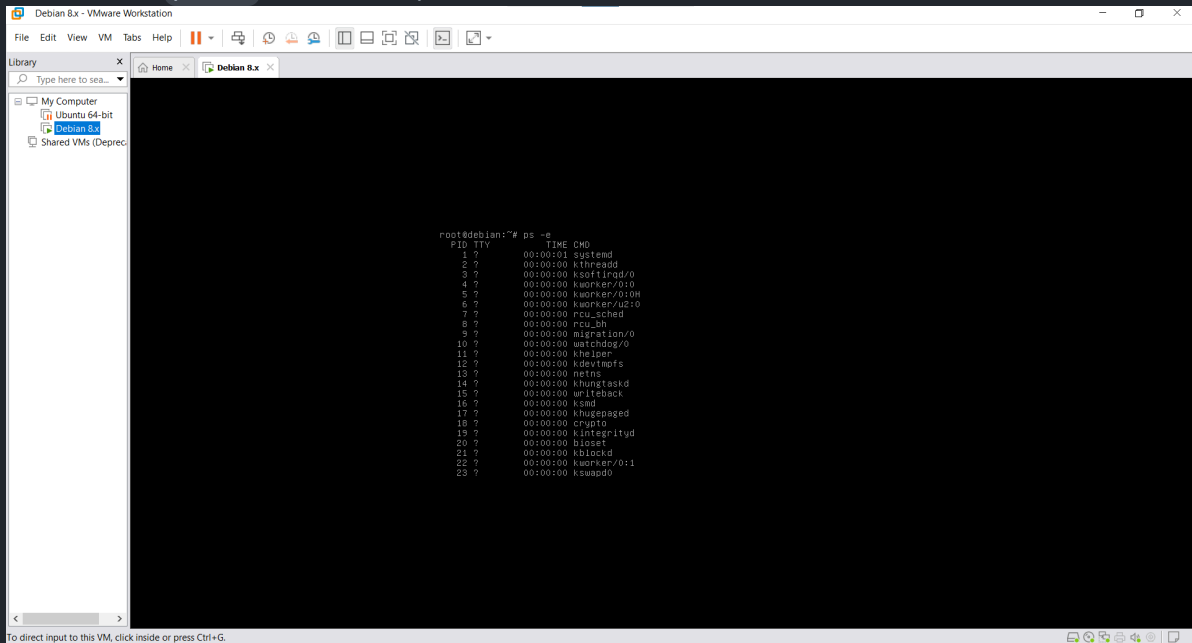
`ps -a` command is for showing processes not for a terminal. The output is:



```
root@debian:~# ps -a
PID TTY          TIME CMD
755 tty1         00:00:00 bash
960 tty1         00:00:00 ps
root@debian:~#
```

- ☑ Information about processes with PID = 1

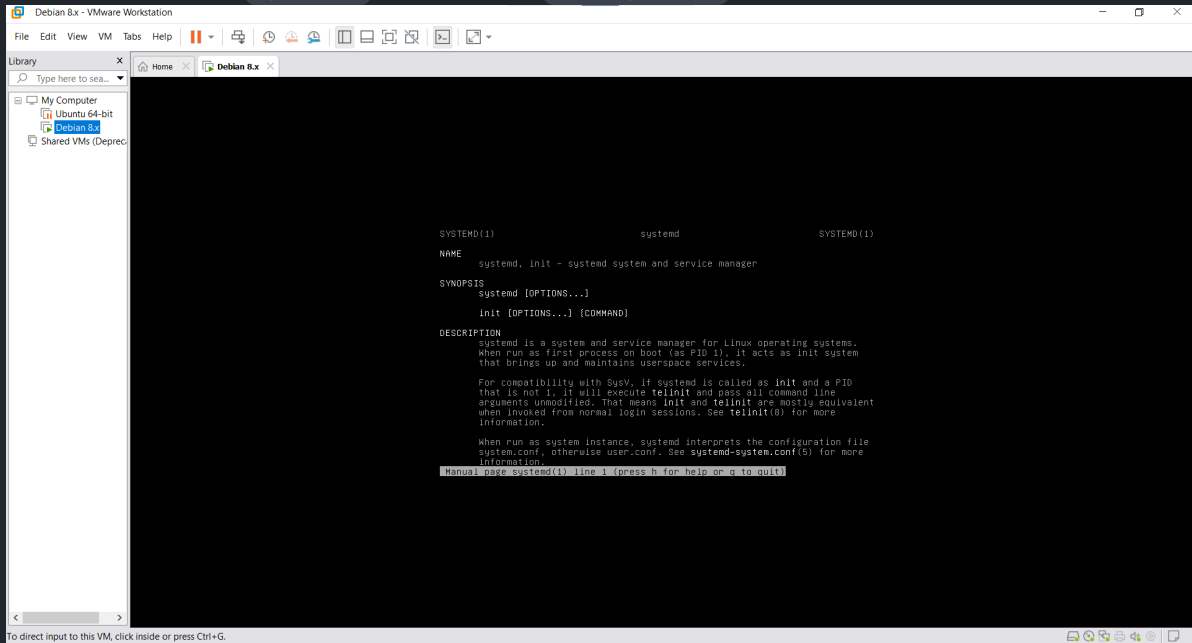
✓ First, we run `ps -e` to see what process has PID 1:



```

root@debian:~# ps -e
  PID TTY          TIME CMD
    1 ?        00:00:01 systemd
    2 ?        00:00:00 kthreadd
    3 ?        00:00:00 ksoftirqd/0
    4 ?        00:00:00 kworker/0:0
    5 ?        00:00:00 kworker/0:0H
    6 ?        00:00:00 kworker/u2:0
    7 ?        00:00:00 rcu_sched
    8 ?        00:00:00 rcu_bh
    9 ?        00:00:00 migration/0
   10 ?        00:00:00 watchdog/0
   11 ?        00:00:00 khelper
   12 ?        00:00:00 kdevmofs
   13 ?        00:00:00 netns
   14 ?        00:00:00 khungtaskd
   15 ?        00:00:00 writeback
   16 ?        00:00:00 ksm
   17 ?        00:00:00 khugepaged
   18 ?        00:00:00 crypto
   19 ?        00:00:00 kintegrityd
   20 ?        00:00:00 bioset
   21 ?        00:00:00 kblockd
   22 ?        00:00:00 kworker/0:1
   23 ?        00:00:00 kswapd0
  
```

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



```

SYSTEMD(1)                  systemd                  SYSTEMD(1)

NAME
  systemd, init - system and service manager

SYNOPSIS
  systemd [OPTIONS...]
  init [OPTIONS...] [COMMAND]

DESCRIPTION
  systemd is a system and service manager for Linux operating systems.
  When run as first process on boot (as PID 1), it acts as init system
  that brings up and maintains userspace services.

  For compatibility with SysV, if systemd is called as init and a PID
  that is not 1, it will execute telinit and pass all command line
  arguments unmodified. That means init and telinit are mostly equivalent
  when invoked from normal login sessions. See telinit(8) for more
  information.

  When run as system instance, systemd interprets the configuration file
  system.conf, otherwise user.conf. See systemd-system.conf(5) for more
  information.
  Manual page systemd(1) line 1 (press h for help or q to quit)
  
```

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.

✓ Program using getpid



The screenshot shows a VMware Workstation window titled "Debian 8.x - VMware Workstation". The interface includes a menu bar (File, Edit, View, VM, Tabs, Help), a toolbar, and a left sidebar with a "Library" pane showing "My Computer" containing "Ubuntu 64-bit" and "Debian 8.x". The main window displays a terminal window titled "GNU nano 2.2.6 File: testpid.cpp". The code in the terminal is as follows:

```
#include <unistd.h>
#include <sys/types.h>
#include <stdio.h>

int main()
{
    pid_t current_process_pid = getpid();
    if (current_process_pid < 0) {
        printf("Failed to get current process pid");
        return 1;
    }
    printf("Current process pid is %d\n", current_process_pid);
    return 0;
}
```

At the bottom of the terminal window, there is a status bar with various icons and text: "Get Help", "Exit", "Writeout", "Justify", "Read File", "Where Is", "Prev Page", "Next Page", "Cut Text", "Uncut Text", "Cur Pos", "To Spell".

## Result:

The screenshot shows the same VMware Workstation window as above, but the terminal window now displays the output of the program execution:

```
root@debian:~# g++ testpid.cpp
root@debian:~# ./a.out
Current process pid is 999
root@debian:~#
```

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

## Section 4.4.2

☒ Program using getppid



The screenshot shows a VMware Workstation window titled "Debian 8.x - VMware Workstation". The interface includes a menu bar (File, Edit, View, VM, Tabs, Help), a toolbar, and a left sidebar with a "Library" pane. The main window displays a terminal window titled "GNU nano 2.2.6 File: testpid.cpp". The code in the terminal is as follows:

```
#include <unistd.h>
#include <sys/types.h>
#include <stdio.h>

int main() {
    pid_t parent_pid = getpid();
    printf("parent pid is %d\n", parent_pid);
    return 0;
}
```

Below the code, there is a status bar with various icons and text: "Get Help", "Exit", "WriteOut", "Justify", "Read File", "Where Is", "Prev Page", "Next Page", "Cut Text", "Uncut Text", "Cur Pos", "To Spell". At the bottom of the window, there is a message: "To direct input to this VM, click inside or press Ctrl+G."

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



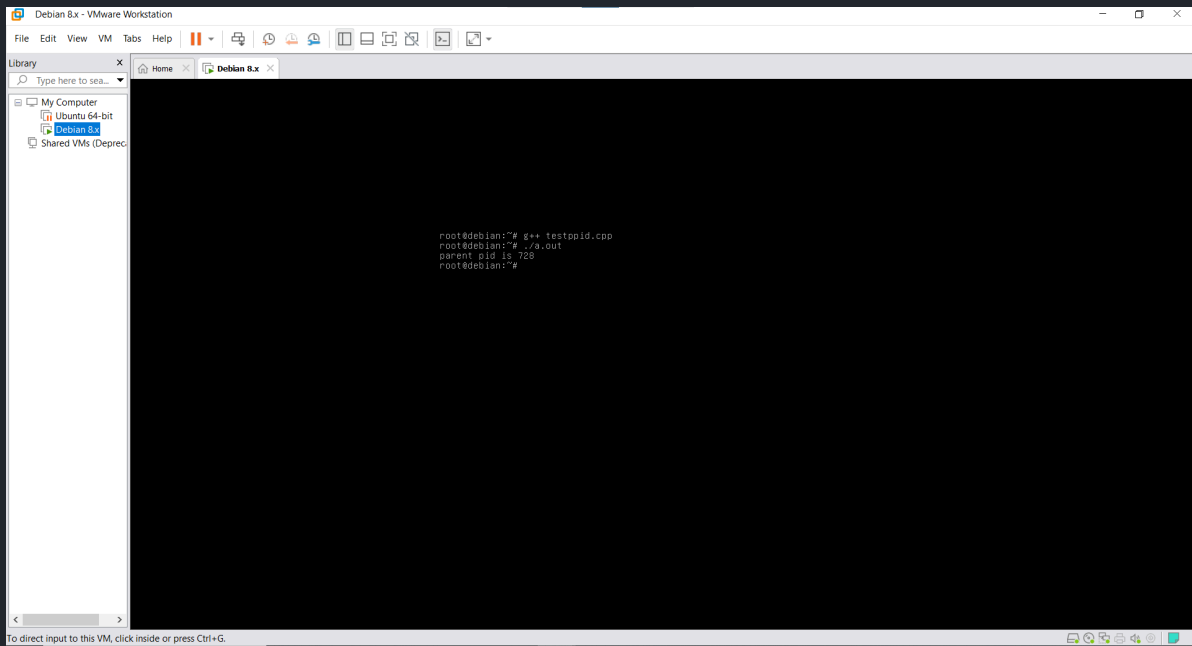
If we run the code, the result is 728. If we run `cat /proc/728/status | grep Name`, we will see its details:

The screenshot shows the same Debian 8.x VM window. The terminal window now displays the output of the command `cat /proc/728/status | grep Name`:

```
root@debian:~# cat /proc/728/status | grep Name
Name:  bash
root@debian:~#
```

The status bar and the bottom message are the same as in the previous screenshot.

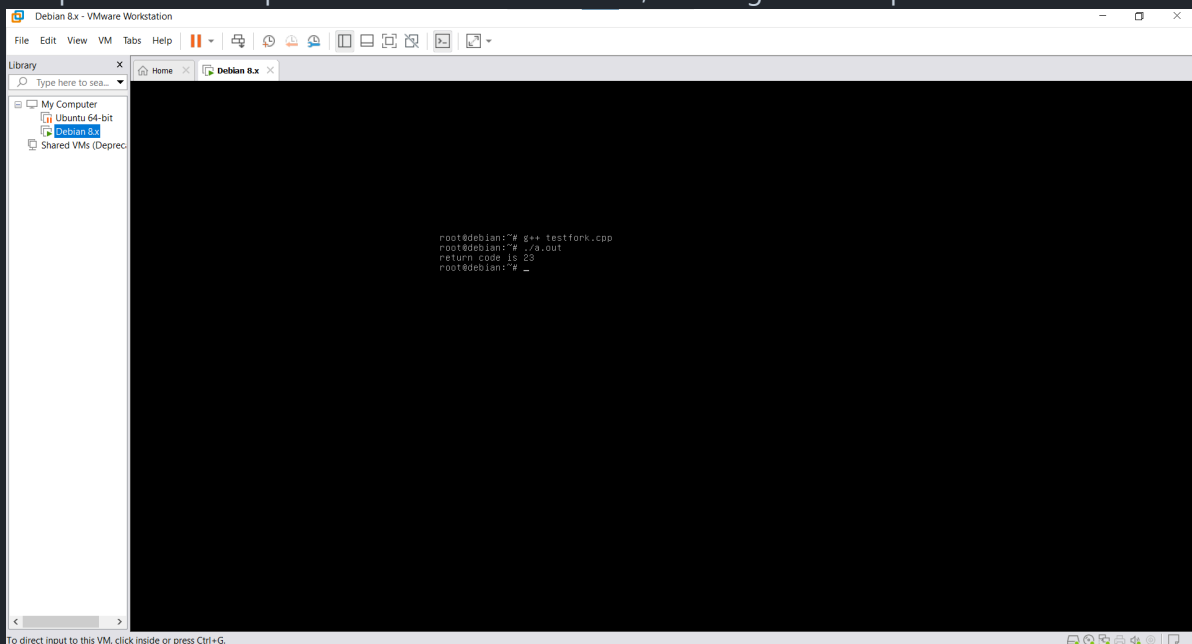
As you see, it's `bash` that runs our code and refers to an instance of running the Bash shell, which is a widely-used command interpreter on Linux. When we open a terminal on a Linux system, a Bash process is typically started. This process runs interactively, reads commands from the user and executes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



```
root@debian:~# g++ testpid.cpp
root@debian:~# ./a.out
parent pid is 728
root@debian:~#
```

### ✓ Describe the C program (fork program)

- ✓ 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:



```
root@debian:~# g++ testfork.cpp
root@debian:~# ./a.out
return code is 23
root@debian:~#
```

- ✓ Program showing that memory of the parent and the child is separate



```

GNU nano 2.2.6      File: 4-4-2-2.cpp      Modified
#include <stdio.h>
#include <sys/wait.h>
#include <unistd.h>

int variable = 0;

int main() {
    int ret = fork();
    if (ret == 0) {
        variable += 1;
        printf("we are in child process and variable = %d\n", variable);
        return 0;
    } else {
        ret = 0;
        wait(&ret);
        variable += 1;
        printf("we are in parent process and variable = %d\n", variable);
    }
    return 0;
}

Get Help  WriteOut  Read File  Prev Page  Cut Text  Cur Pos
Exit      Justify   Where Is  Next Page  Unbut Text To Spell
  
```

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:

```

root@debian:~# g++ 4-4-2-2.cpp
root@debian:~# ./a.out
we are in child process and variable = 1
we are in parent process and variable = 1
root@debian:~#
  
```

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

- ✓ Program printing different messages for parent and child process





The screenshot shows a VMware Workstation window titled "Debian 8.x - VMware Workstation". The interface includes a menu bar (File, Edit, View, VM, Tabs, Help), a toolbar, and a sidebar with a "Library" pane. The main window displays a terminal window with the GNU nano 2.2.6 editor open. The editor shows a C++ program named "4-4-2-3.cpp". The code is as follows:

```
GNU nano 2.2.6 File: 4-4-2-3.cpp Modified

#include <stdio.h>
#include <sys/wait.h>
#include <unistd.h>

int main() {
    int ret = fork();
    if (ret == 0) {
        printf("I am the child\n");
        return 23;
    } else {
        int rc = 0;
        wait(&rc);
        printf("I am the parent\n");
    }
    return 0;
}
```

At the bottom of the terminal window, there is a status bar with various icons and a message: "To direct input to this VM, click inside or press Ctrl+G."

The output is:

The screenshot shows the same VMware Workstation window as above, but the terminal window now displays the output of the program. The output is as follows:

```
root@debian:~# g++ 4-4-2-3.cpp
root@debian:~# ./a.out
I am the child
I am the parent
root@debian:~#
```

The status bar at the bottom of the terminal window still shows the message: "To direct input to this VM, click inside or press Ctrl+G."

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



Program for the last task of this section



```

GNU nano 2.2.6 File: 4-4-2-4.cpp Modified
#include <stdio.h>
#include <sys/wait.h>
#include <unistd.h>
int main() {
    int ret = fork();
    printf("After first fork\n");
    fork();
    printf("After second fork\n");
    fork();
    printf("After third fork\n");
    if (ret == 0) {
        printf("I am the child\n");
        return 23;
    } else {
        int rc = 0;
        wait(&rc);
        printf("I am the parent\n");
    }
    return 0;
}
Get Help WriteOut Read File Prev Page Cut Text Cur Pos
Exit Justify Where Is Next Page Uncut Text To Spell

```



```

root@debian:~# g++ 4-4-2-4.cpp
root@debian:~# ./a.out
After first fork
After first fork
After second fork
After second fork
After second fork
After second fork
After third fork
After third fork
After third fork
After third fork
After third fork
After third fork
After third fork
After third fork
I am the parent
I am the child
I am the child
I am the parent
I am the child
I am the parent
I am the child
I am the parent
I am the child
I am the parent
root@debian:~# _

```

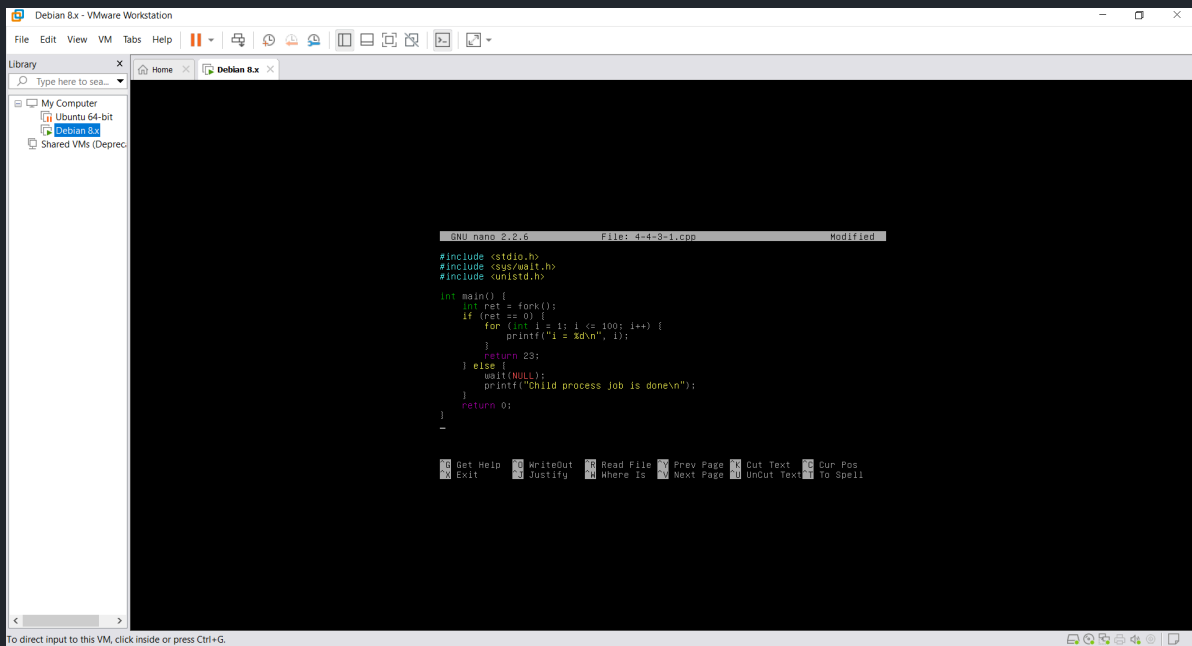
After the first fork, we have 2 processes, So the phrase **After first fork** is printed 2 times. After the second fork, we have 4 processes (each of the first 2 processes creates another process), So the phrase **After second fork** is printed 4 times. After the third fork we have 8 processes, So the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

✓ Program using `wait` and counting from 1 to 100

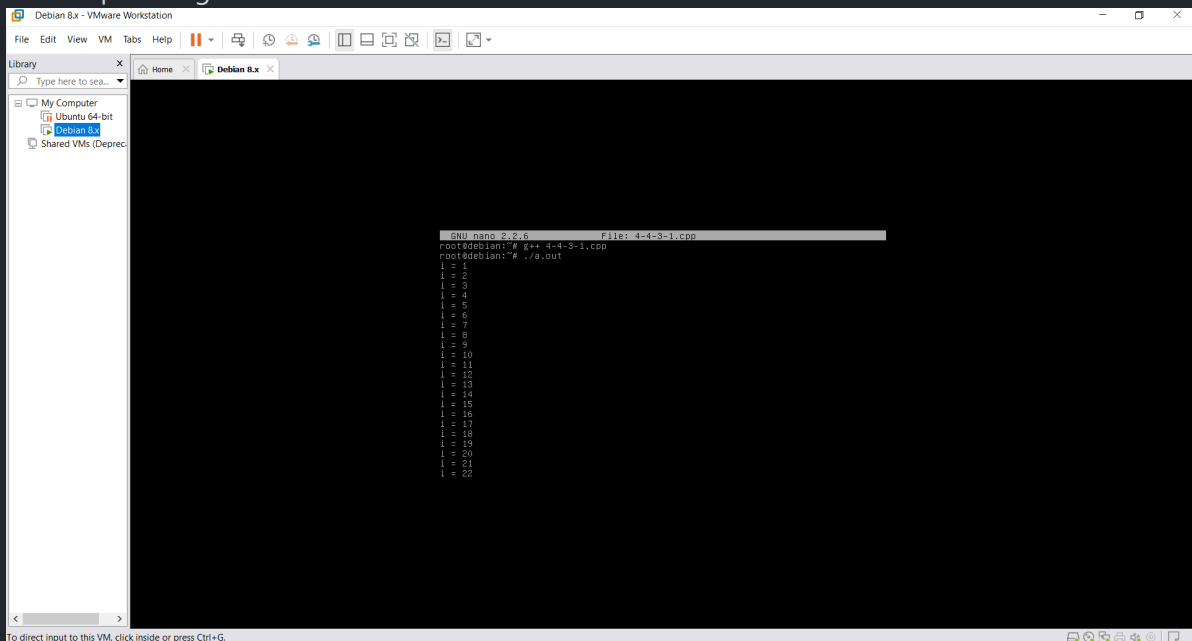


```
GNU nano 2.2.6 File: 4-4-3-1.cpp Modified
#include <stdio.h>
#include <sys/wait.h>
#include <unistd.h>

int main() {
    int ret = fork();
    if (ret == 0) {
        for (int i = 1; i <= 100; i++) {
            printf("i = %d\n", i);
        }
        return 23;
    } else {
        wait(NULL);
        printf("Child process job is done\n");
    }
    return 0;
}
```

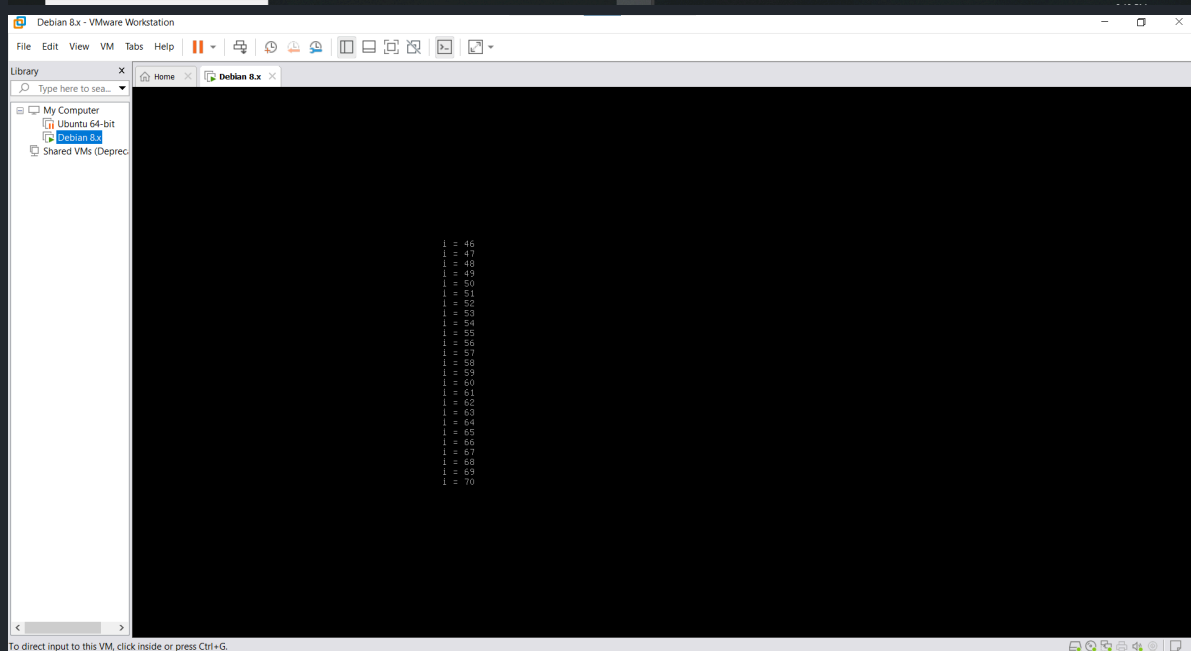
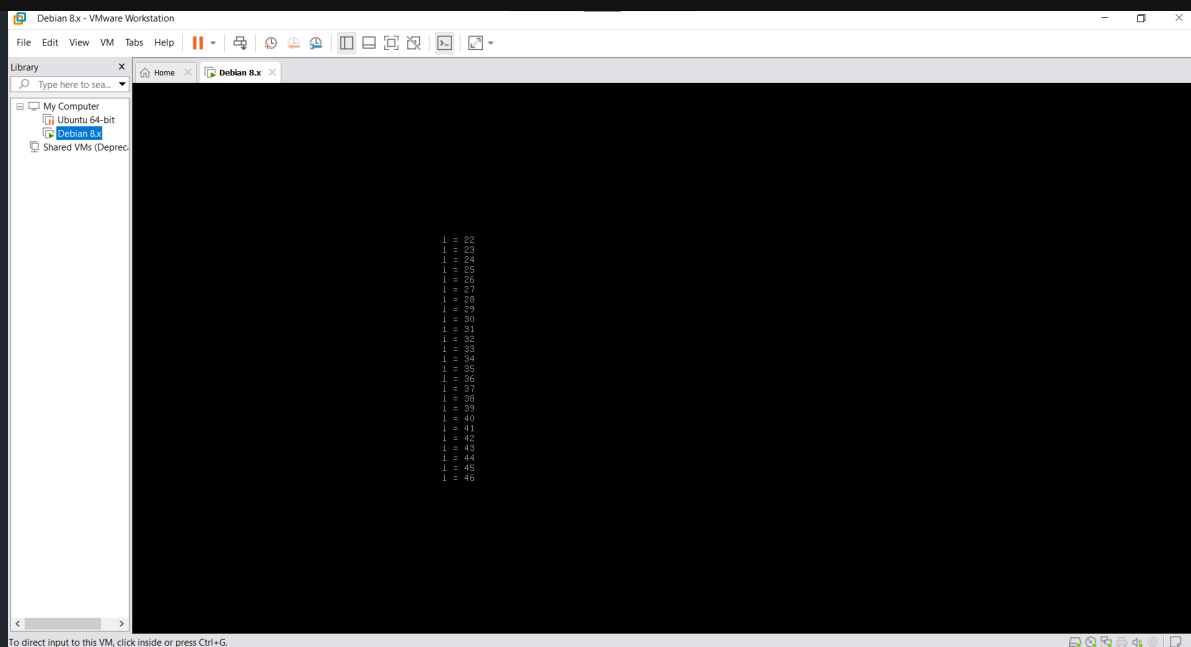
The code is in file `4-4-3-1.cpp`.

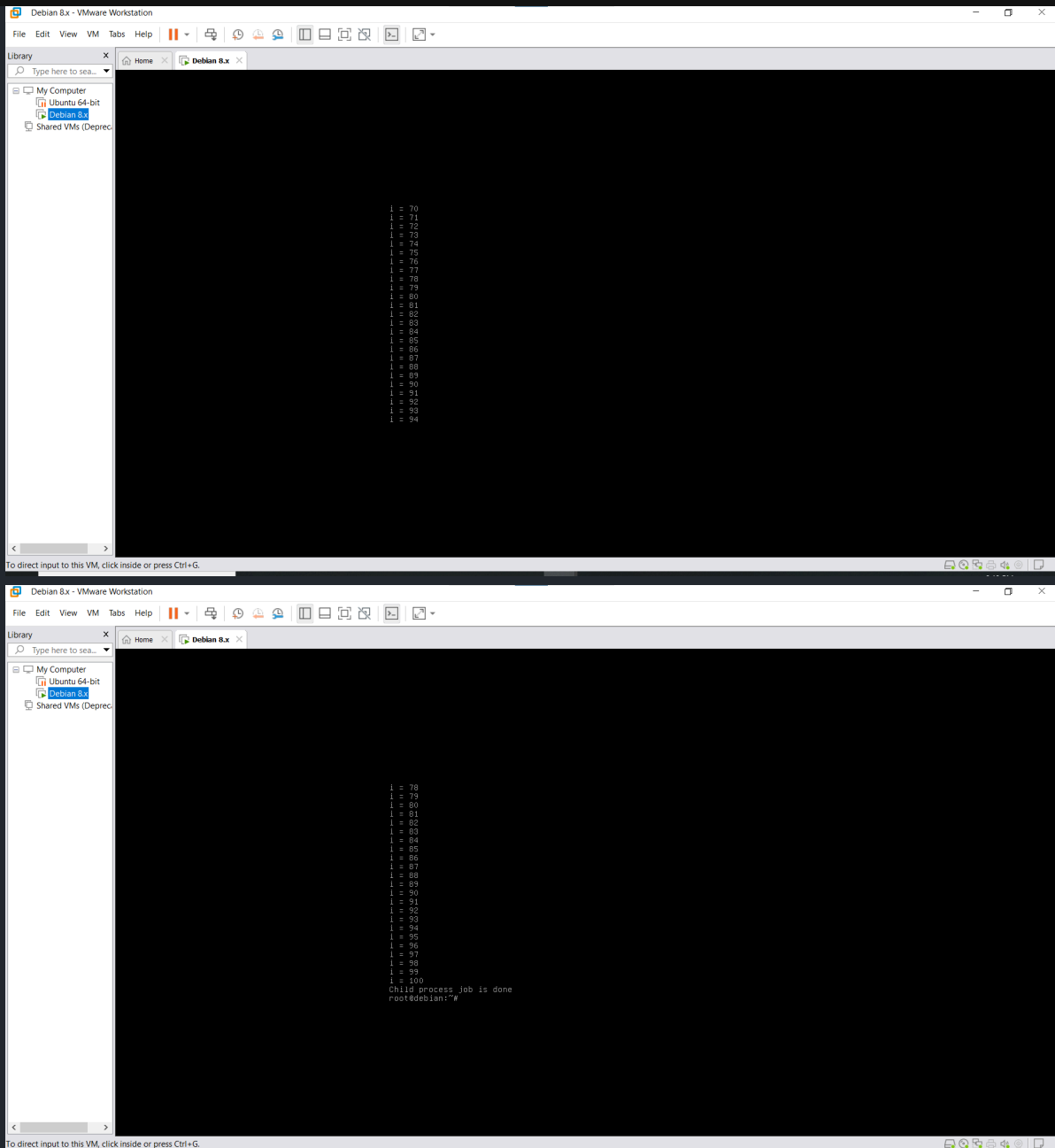
The output images are:



```
GNU nano 2.2.6 File: 4-4-3-1.cpp
root@debian:~# g++ 4-4-3-1.cpp
root@debian:~# ./4.out
i = 1
i = 2
i = 3
i = 4
i = 5
i = 6
i = 7
i = 8
i = 9
i = 10
i = 11
i = 12
i = 13
i = 14
i = 15
i = 16
i = 17
i = 18
i = 19
i = 20
i = 21
i = 22

```





- ✓ 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.

- ✓ Program showing process adoption



```

GNU nano 2.2.6      File: 4-4-3-2.cpp      Modified
#include <stdio.h>
#include <sys/wait.h>
#include <unistd.h>
int main() {
    int ret = fork();
    if (ret < 0) {
        printf("Failed to fork a process");
        return 1;
    } else if (ret == 0) {
        printf("My parent pid before adoption is : %d\n", getpid());
        sleep(5);
        printf("My parent pid after adoption is : %d\n", getpid());
        return 0;
    } else {
        printf("The parent pid is : %d\n", getpid());
        sleep(10);
        printf("The parent job is done\n");
    }
    return 0;
}
Get Help  WriteOut  Read File  Prev Page  Cut Text  Cur Pos
Exit      Justify   Where Is  Next Page  Uncut Text To Spell

```

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



```

root@debian:~# g++ 4-4-3-2.cpp
root@debian:~# ./a.out
The parent pid is : 766
My parent pid before adoption is : 766
The parent job is done
root@debian:~# My parent pid after adoption is : 1
-

```

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



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.

✓ Program which forks and executes `ls` command



```

GNU nano 2.2.6      File: 4-4-4.cpp
#include <stdio.h>
#include <sys/wait.h>
#include <unistd.h>

int main() {
    int ret = fork();
    if (ret < 0) {
        printf("Failed to fork a process");
        return 1;
    } else if (ret == 0) {
        execlp("ls", "ls", "-g", "-h", (char *)NULL);
        return 0;
    } else {
        wait(NULL);
    }
    return 0;
}

[ Wrote 17 lines ]

root@debian:~# g++ 4-4-4.cpp

```

The code is in file `4-4-4.cpp` and I used `exec1p` function which is explained above. The output is:



```
root@debian:~# g++ 4-4-4.cpp
root@debian:~# ./a.out
10193 634
-rw-r--r-- 1 root 422 Jul 15 10:51 4-4-2-2.cpp
-rw-r--r-- 1 root 283 Jul 15 10:58 4-4-2-3.cpp
-rw-r--r-- 1 root 409 Jul 15 11:03 4-4-2-4.cpp
-rw-r--r-- 1 root 324 Jul 15 11:16 4-4-3-1.cpp
-rw-r--r-- 1 root 530 Jul 15 11:32 4-4-3-2.cpp
-rw-r--r-- 1 root 332 Jul 16 08:13 4-4-4.cpp
-rw-r--r-- 1 root 544 Jul 16 08:13 a.out
drwxr-xr-x 25 root 4.0k Jul 3 13:22 linux-3.16.56
-rw-r--r-- 1 root 2.4M May 8 2018 linux_3.16.56-1+deb8u1.debian.tar.xz
-rw-r--r-- 1 root 138K May 8 2018 linux_3.16.56-1+deb8u1.dsc
-rw-r--r-- 1 root 79M Apr 20 2018 linux_3.16.56.orig.tar.xz
-rw-r--r-- 1 root 310 Jul 15 07:08 testpid.cpp
-rw-r--r-- 1 root 272 Jul 15 07:15 testpid.c
root@debian:~#
```

## Source Code Submission

please submit all your codes in a zip file

 [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 