

## Assignment 2

### { Muhammad Hamad – MS-IC-21354 – Operating System}

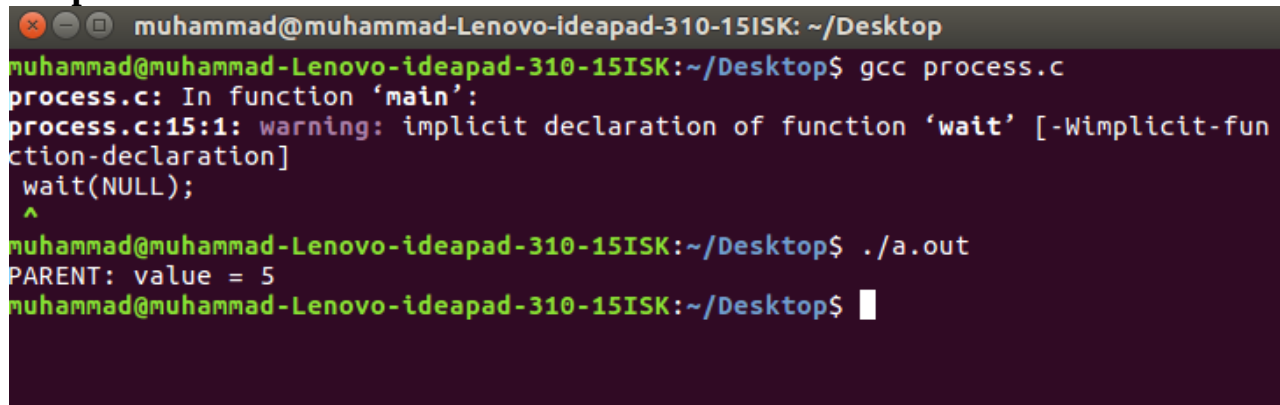
---

**3.1 :** Using the program shown in Figure 3.30, explain what the output will be at LINE A ?

```
#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>
int value = 5;
int main()
{
    pid_t pid;
    pid = fork();

    if (pid == 0) { /* child process */
        value += 15;
        return 0;
    }
    else if (pid > 0) { /* parent process */
        wait(NULL);
        printf("PARENT: value = %d\n",value); /* LINE A */
        return 0;
    }
}
```

**Output :**



The screenshot shows a terminal window with the following commands and output:

```
muhammad@muhammad-Lenovo-ideapad-310-15ISK: ~/Desktop
muhammad@muhammad-Lenovo-ideapad-310-15ISK:~/Desktop$ gcc process.c
process.c: In function 'main':
process.c:15:1: warning: implicit declaration of function 'wait' [-Wimplicit-function-declaration]
    wait(NULL);
    ^
muhammad@muhammad-Lenovo-ideapad-310-15ISK:~/Desktop$ ./a.out
PARENT: value = 5
muhammad@muhammad-Lenovo-ideapad-310-15ISK:~/Desktop$
```

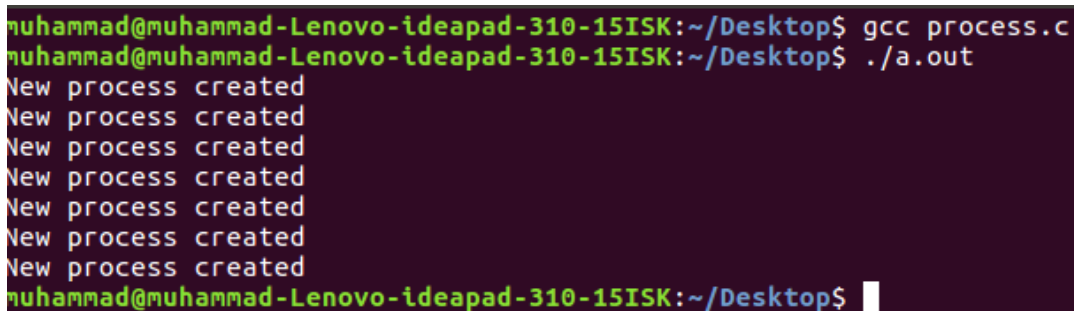
**Explanation:** When the `fork()` was called it created a duplicate process , and inside the duplicated process the condition `pid == 0` becomes true and `value` was incremented to 15 , but child updates its own copy of value it has nothing to do with parent process so when inside parent `pid > 0` becomes true it will still display `value=5` at line A .

---

**3.2 :** Including the initial parent process, how many processes are created by the program shown in Figure 3.31?

### Program :

```
#include <stdio.h>
#include <unistd.h>
int main()
{
    /* fork a child process */
    printf("New process created\n");
    fork();
    /* fork another child process */
    printf("New process created\n");
    fork();
    /* and fork another */
    printf("New process created\n");
    fork();
}
```



```
muhammad@muhammad-Lenovo-ideapad-310-15ISK:~/Desktop$ gcc process.c
muhammad@muhammad-Lenovo-ideapad-310-15ISK:~/Desktop$ ./a.out
New process created
New process created
New process created
New process created
New process created
New process created
New process created
muhammad@muhammad-Lenovo-ideapad-310-15ISK:~/Desktop$
```

### Output :

**Explanation :** As we can see from the output that 1 parent process and 7 child processes will be created .

---

**3.5 :** When a process creates a new process using the fork() operation, which of the following states is shared between the parent process and the child process?

- a) Stack
- b) Heap
- c) Shared memory segments

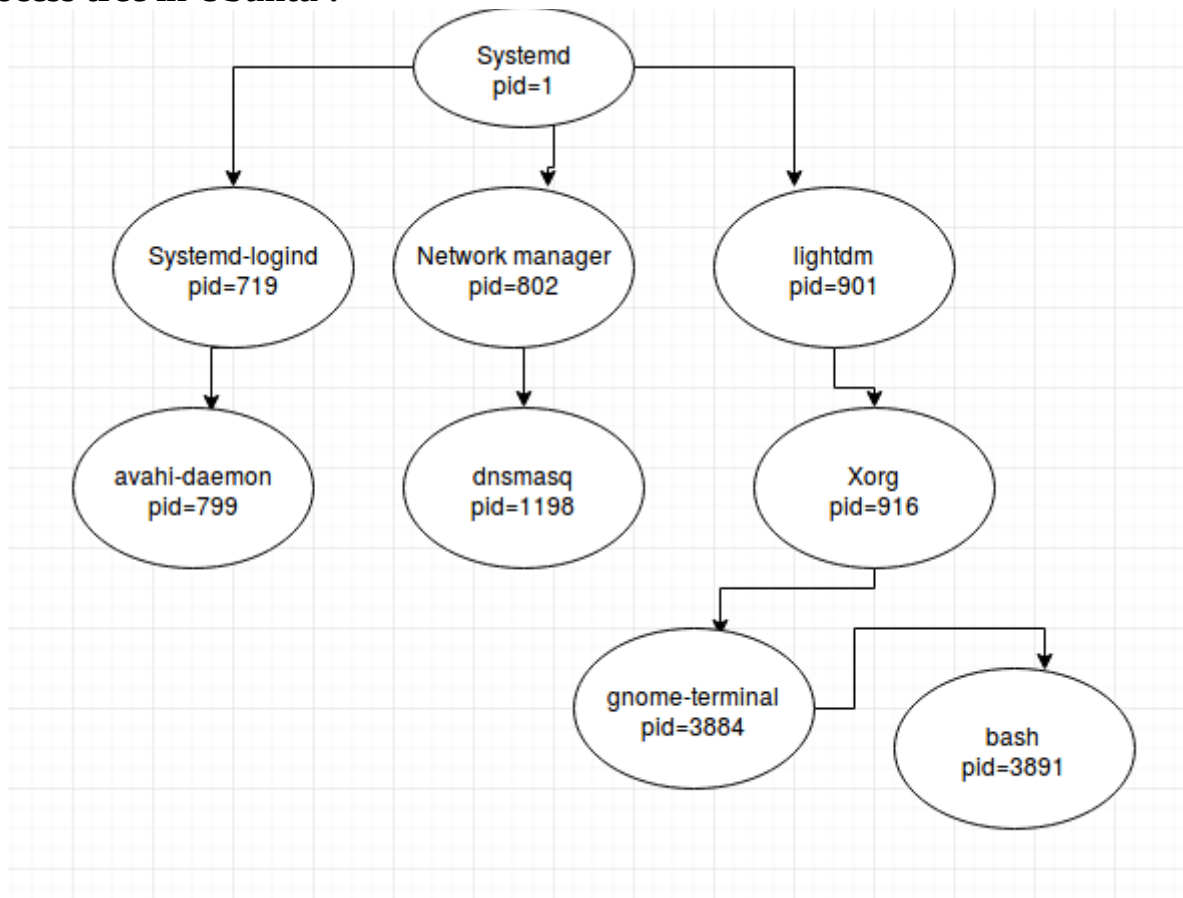
**Ans :** Only the shared memory segments are shared between the parent process and the newly forked child process. Copies of the stack and the heap are made for the newly created process.

---

**3.10 :** Construct a process tree similar to Figure 3.8. To obtain process information for the UNIX or Linux system, use the command `ps -ael` .

**Ans :** I wrote this command in Ubuntu terminal , and I got hundreds of processes with their process id's and I picked some of them and constructed the tree.

### Process tree in Ubuntu :



**3.14 :** Using the program in Figure 3.34, identify the values of pid at lines A , B ,C , and D . (Assume that the actual pids of the parent and child are 8620 and 8621, respectively.)

**Explanation :** When fork will be called it will return child process id that is 8621 will be stored in pid so we can say **in parent process pid=8621**, Inside the **child process** fork returns 0 and then condition **pid==0** becomes **true** and **pid1=getpid()**; will execute and **pid1=8621 in child process** ,Now when printf statements will execute they will display :

**child: pid=0**  
**child: pid1=8621**

Now when parent process will execute **pid1=getpid()** will save parent process id in pid1 that is **pid1=8620** , Now when printf statements will in parent process execute they will display :

**parent: pid=8621**  
**parent: pid1=8620**

## Program:

```
//3.14
#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>
int main()
{
    pid_t pid, pid1;
    /* fork a child process */
    pid = fork();
    if (pid < 0) { /* error occurred */
        fprintf(stderr, "Fork Failed");
        return 1;
    }
    else if (pid == 0) { /* child process */
        pid1 = getpid();
        printf("child: pid = %d\n",pid); /* A */
        printf("child: pid1 = %d\n",pid1); /* B */
    }
    else { /* parent process */
        pid1 = getpid();
        printf("parent: pid = %d\n",pid); /* C */
        printf("parent: pid1 = %d\n",pid1); /* D */
        wait(NULL);
    }
}
```

## Output :

```
muhammad@muhammad-Lenovo-ideapad-310-15ISK:~/Desktop$ ./a.out
parent: pid = 8621
parent: pid1 = 8620
child: pid = 0
child: pid1 = 8621
muhammad@muhammad-Lenovo-ideapad-310-15ISK:~/Desktop$
```

THE END