PRACTICAL 1

Objective:

Write a program (using fork() and/or exec() commands) where parent and child execute:

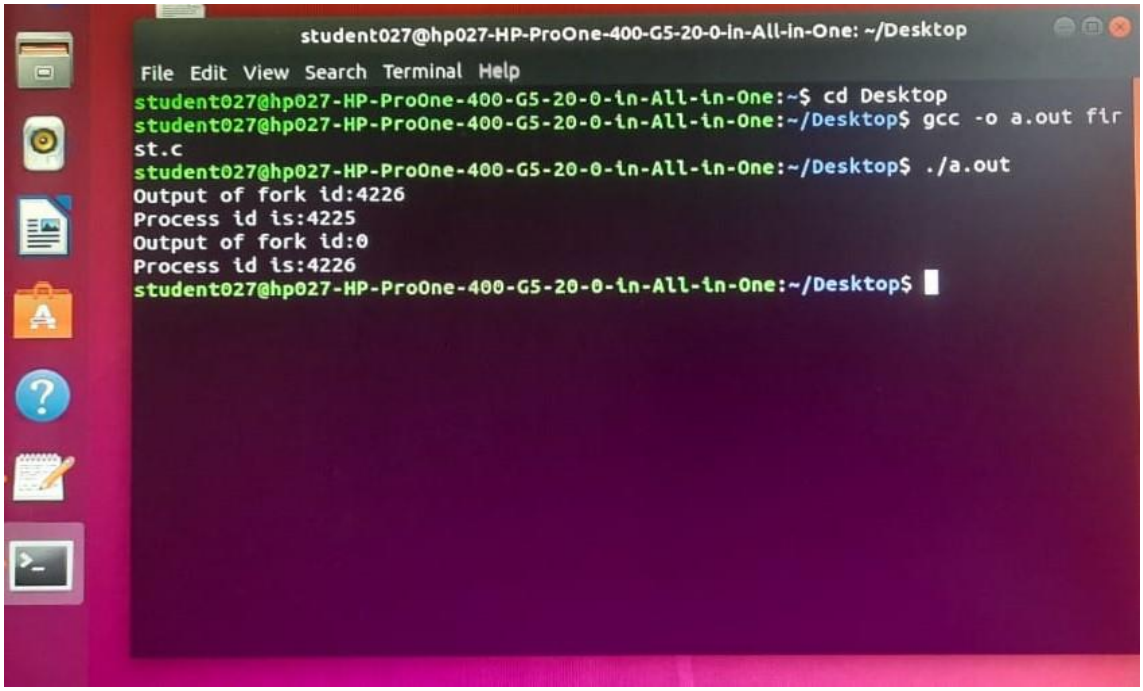
a) same program, same code.

Code:

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int main()
{
        pid_t pid,p;
        p = fork();
        pid = getpid();
        if(p<0)
        {
                fprintf(stderr, “Fork Failed ”);
                return 1;
        }
        printf(“Output of fork id: %d\n” ,p);
        printf(“Process id is: %d \n” ,pid);
        return 0;
}
```

Output:



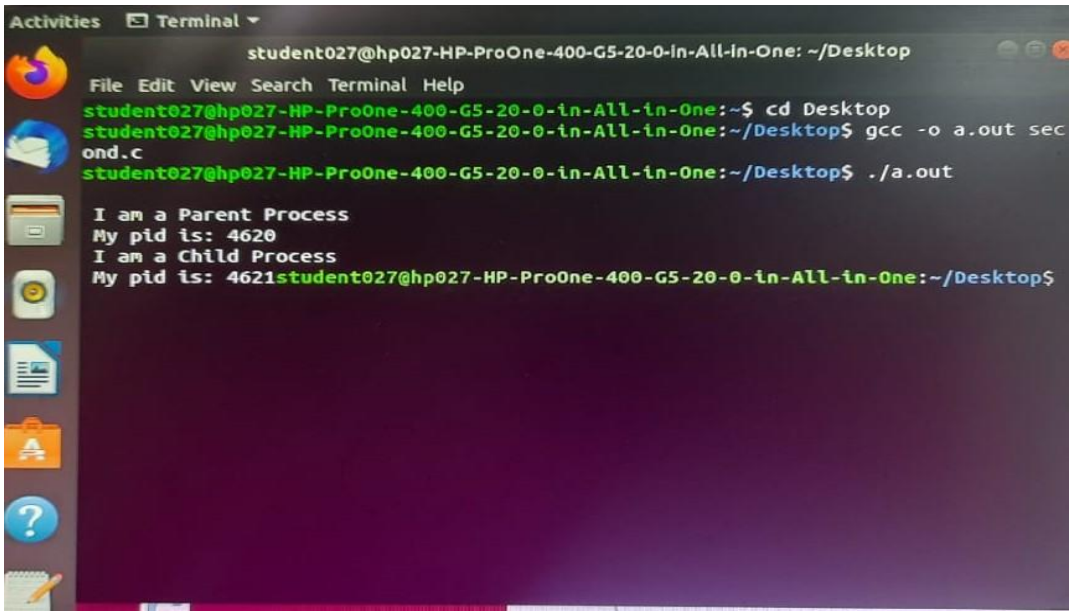
b) same program, different code.

Code:

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int main()
{
        pid_t pid,p;
        p = fork();
        if(p<0)
        {
                fprintf(stderr, “Fork Failed ”);
                exit(1);
        }
        else if(pid==0)
        {
                printf(“\nI am a Child Process”);
                printf(“\nMy pid is: %d \n” ,getpid());
                exit(0);
        }
        else
        {
                printf(“\nI am a Parent Process”);
                printf(“\nMy pid is: %d \n” ,getpid());
                exit(1);
        }
}
```

Output:

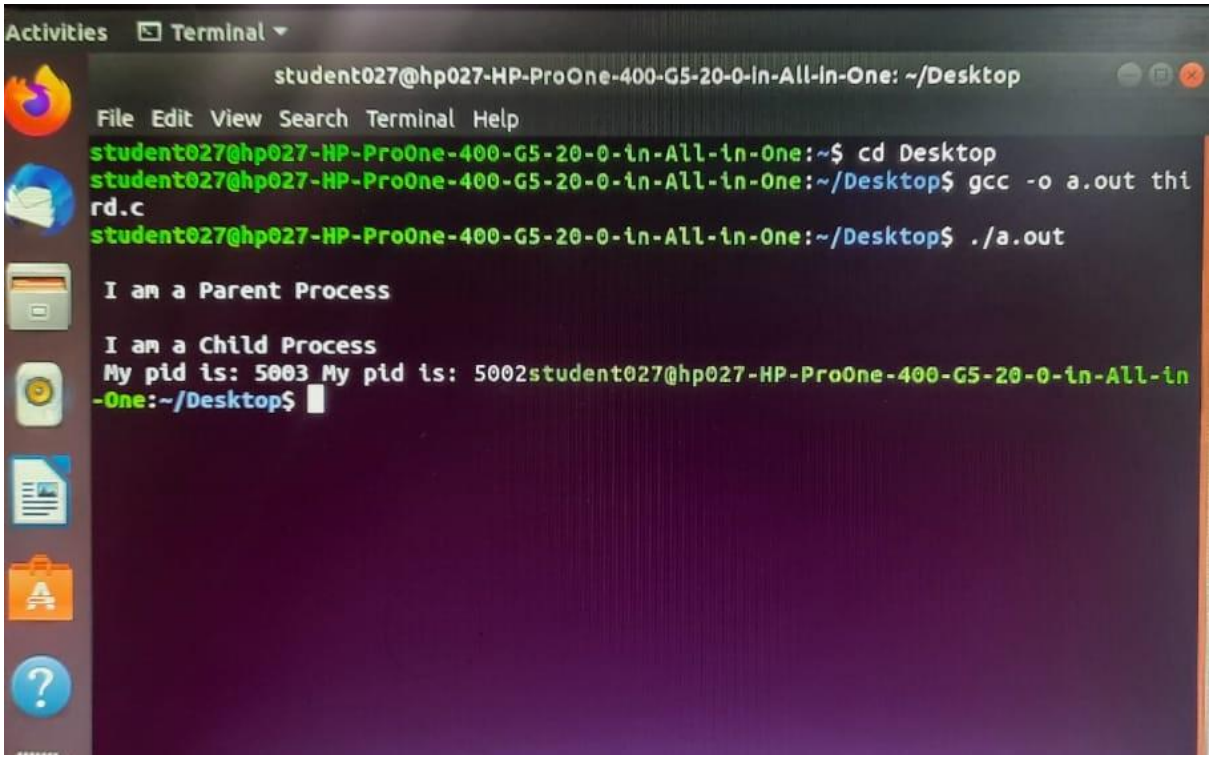
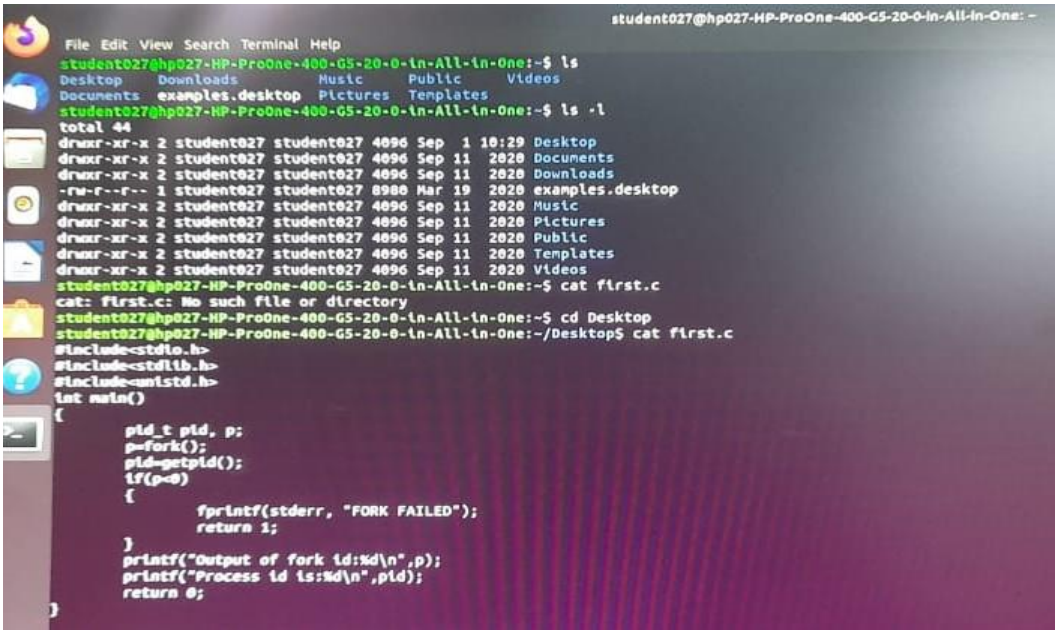
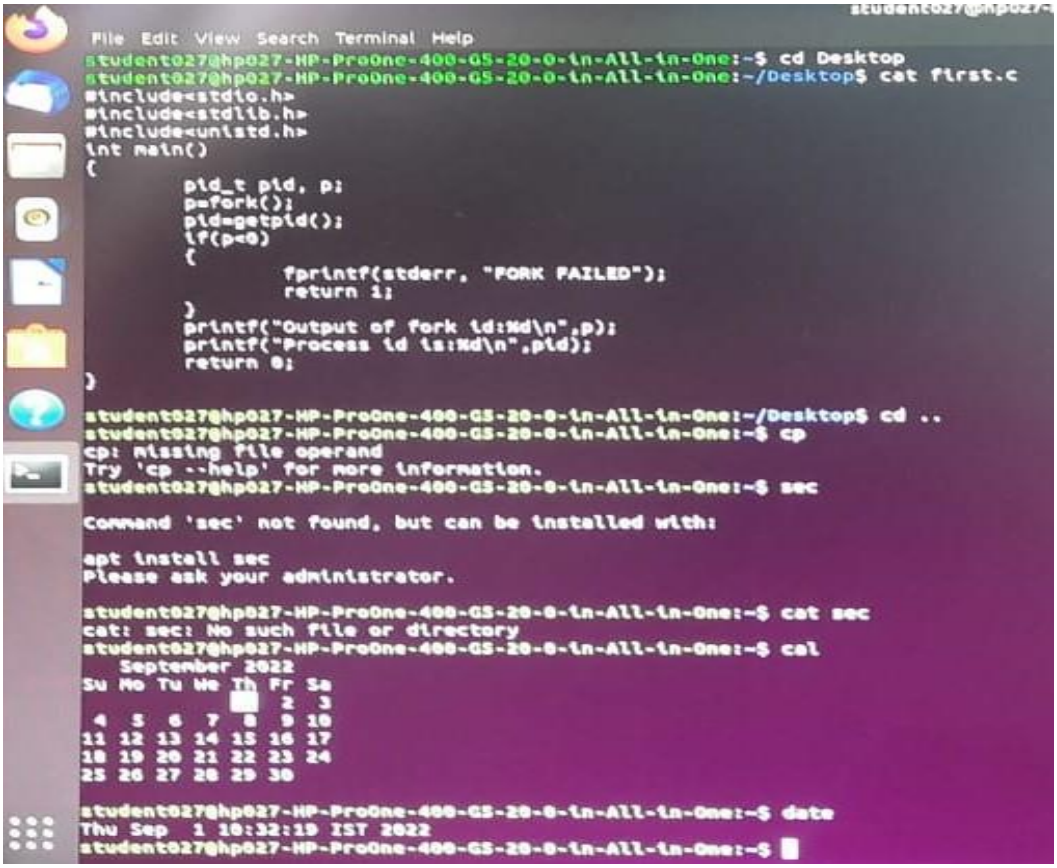


c) before terminating, the parent waits for the child to finish its task.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int main()
{
        pid_t pid,p;
        p = fork();
        pid = getpid();
        if(p<0)
        {
                fprintf(stderr, “Fork Failed ”);
                return 1;
        }
        printf(“Output of fork id: %d\n” ,p);
        printf(“Process id is: %d \n” ,pid);
        return 0;
}
```

Output:



➢UNIX COMMANDS (date, cal, ls, cp, cat, ls-l)