122CS0067

Assignment-III

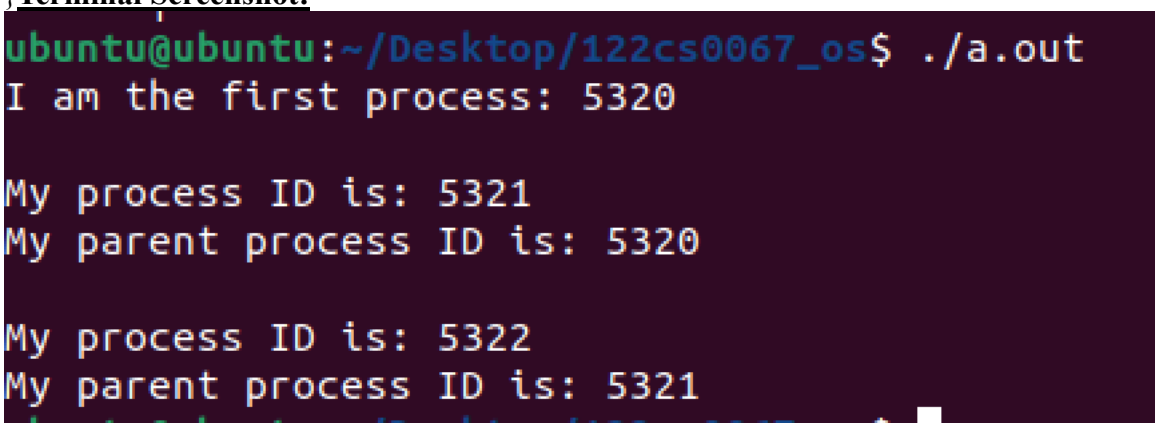
Name: **Amiya Chowdhury** Roll: **122CS0067** Date: **19/08/2024**

1. Write a program using fork() system call to create a hierarchy of 3 process such that P2 is the child of P1 and P1 is the child of P.

3_1.c

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

int main(){
int p1,p2;
printf("I am the first process: %d\n",p1=getpid());
pid_t p=fork();

if(p<0){
perror("Failed");
exit(1);
}

if(p==0){

   if(getppid()==p1){
   printf("\nMy process ID is: %d\n",p2=getpid());
   printf("My parent process ID is: %d\n",p1);
   }
   p=fork();

   if(p<0){
      perror("Failed");
      exit(1);
}

   if(p==0){
            if(getppid()==p2){
                     printf("\nMy process ID is: %d\n",getpid());
            printf("My parent process ID is: %d\n",p2);

        }}
}


return 0;
}
```

Terminal Screenshot:



122CS0067

2. Write a C program to create a tree of processes.

3_2.c

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


int main(void) {
   int a=fork();
   int b=fork();
   int c=fork();

   if (a > 0 && b > 0 && c>0) { //1
      printf("parent\n");
      printf("%d %d %d\n", a,b,c);
      printf(" my id is %d \n", getpid());
   }
   else if (a == 0 && b== 0 && c > 0)
   { //2
      printf("First child\n");
      printf("%d %d %d\n", a, b, c);
      printf("my id is %d  \n", getpid());
   }
   else if (a == 0 && b> 0 && c == 0)
   { //3
      printf("Second child\n");
      printf("%d %d %d\n", a, b, c);
      printf("my id is %d  \n", getpid());
   }
   else if (a > 0 && b== 0 && c == 0)
   { //4
      printf("Third child\n");
      printf("%d %d %d\n", a, b, c);
      printf("my id is %d  \n", getpid());
   }
   else if (a > 0 && b== 0 && c > 0)
   { //5
      printf("Fourth child\n");
      printf("%d %d %d\n", a, b, c);
      printf("my id is %d  \n", getpid());
   }
   else if (a > 0 && b> 0 && c == 0)
   { //6
      printf("Fifth child\n");
      printf("%d %d %d\n", a, b, c);
      printf("my id is %d  \n", getpid());
   }
   else if (a > 0 && b== 0 && c > 0)
   { //7
      printf("Sixth child\n");
      printf("%d %d %d\n", a, b, c);
      printf("my id is %d  \n", getpid());
   }
   else  {//8
      printf("Seventh child\n");
      printf("%d %d %d\n", a, b, c);
      printf(" my id is %d \n", getpid());
   }
   return 0;
```

122CS0067

}

//This creates a hierarchy of processes. It is difficult to store the pids in a tree as any global variable created has multiple instances across the child processes. I therefore sought to output all the processes created as a result of three fork statements.

Terminal Screenshot:

```
ubuntu@ubuntu:~/Desktop/122cs0067_os$ ./a.out
parent
5367 5368 5369
 my id is 5366
Fifth child
Seventh child
5367 5368 0
my id is 5369
ubuntu@ubuntu:~/Desktop/122cs0067_os$ 0 5370 5371
 my id is 5367
Fourth child
5367 0 5372
my id is 5368
Second child
Third child
0 5370 0
my id is 5371
5367 0 0
my id is 5372
First child
Seventh child
0 0 5373
my id is 5370
0 0 0
 my id is 5373
```

122CS0067

3. Write a program where two child processes (P1 and P2) execute different tasks in parallel (e.g., P1 counts from 1 to 10, while P2 prints the alphabet). Ensure that both processes are created by the parent process (P).

3_3.c

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

void pnum(){
for(int i=1;i<10;i++){
printf("P1::%d\n",i);
sleep(1);
}
exit(1);
}

void palpha(){
char a='A';
for(int i=0;i<26;i++){
printf("P2::%c\n",a++);
sleep(1);
}
exit(1);
}

int main(){

pid_t pid1,pid2;

pid1=fork();
if(pid1==0){
palpha();}

pid2=fork();
if(pid2==0){
pnum();}

waitpid(pid1,0,0);
waitpid(pid2,0,0);
return 0;
}
```

Terminal Screenshot:

```
ubuntu@ubuntu:~/Desktop/122cs0067_os$ ./a.out
P1::1
P2::A
P2::B
P1::2
P2::C
P1::3
P1::4
P2::D
P1::5
P2::E
P2::F
P1::6
P1::7
P2::G
P1::8
```

3. Explain the output of the following program?

```
#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>
int main(void)
{
 int i;     for (i=0; i < 3; i++) {
fork();
printf("hello\n");
}
return 0;
}
```

Solution:

Output: The program prints the “hello\n” statement fourteen times.

Explanation: In the program, the fork statement is called three times: for the first call, a single child process and the parent exist: there are two instances of print; for the second fork(), a child is created by the first child as well as the parent process, there a total of four processes running and so there are four instances of print; for the third fork each process spawns a child and there are a total of eight processes running, so there 8 instances of print.
In total, ‘hello’ is printed 2+4+8=14 times.