OS Project - 1

Varun Subramanya

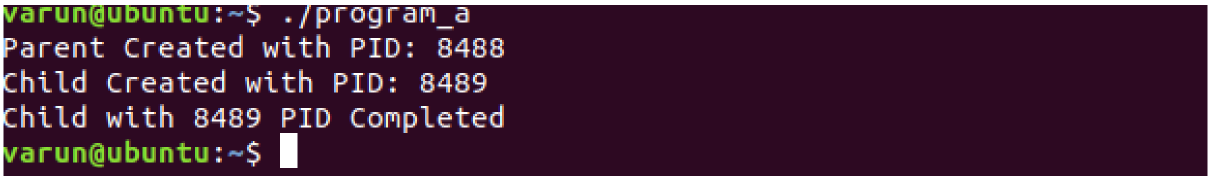
01705578

Program_a:

```
/*Author : *Varun Subramanya
Student ID: 01705578*/

/* The standard header files that ar required by the program*/

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


int main()
{
        int status = 0;
        pid_t pid;
        pid_t pid1;
/* At this point the program forks - There is a parent and a child when fork() is called */
        pid = fork();

        if (pid<0)
                {
                fprintf(stderr, "Fork Failed");
                exit (1);

/* Child Process*/                 }
        else if (pid ==0)
                {
                printf("Child Created with PID: %d\n", getpid()); // getpid() is a function that returns
the Process ID (PID) of the process.
```

```
                    exit (0); // returns 0 for a successful child creation

                    }

/* Parent process*/


          else

          {

          printf("Parent Created with PID: %d\n",getpid());

          pid1= wait(NULL); //The pid1 waits for the child to complete. Once completed it returns the
PID of the corrosponding process

          printf("Child with %d PID Completed\n",pid1);

          }

return 0;

}
```

Program a output:



```
varun@ubuntu:~$ ./program_a
Parent Created with PID: 8488
Child Created with PID: 8489
Child with 8489 PID Completed
varun@ubuntu:~$
```

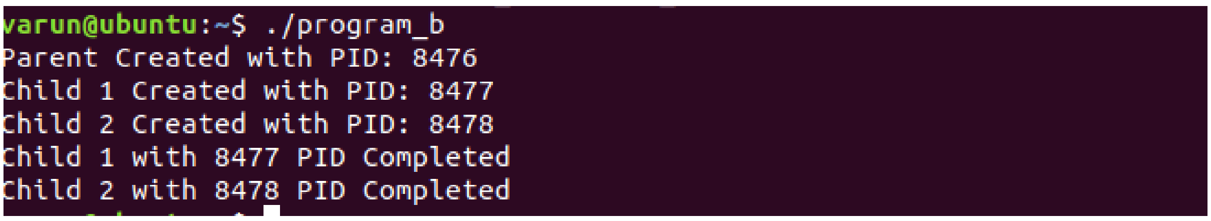
Program_b:

```
/*Author : *Varun Subramanya
Student ID: 01705578*/

/* In this program after the child1 ends the process is forked again for the second child.*/

/* The standard header files that ar required by the program*/
#include<stdio.h>
#include<sys/wait.h>
#include<unistd.h>
#include<stdlib.h>

int main()
{

        int status = 0;
        pid_t pid;
        pid_t pid1, pid2;

/* At this point the program forks - There is a parent and a child when fork() is called */
        pid = fork();

        int num = 1;
        int num2;

        if (pid<0)
                {
                fprintf(stderr, "Fork Failed");
                exit (1);
                }
```

```
/* Child Process*/
else if (pid ==0)
                  {
              printf("Child %d Created with PID: %d\n", num, getpid());



                  exit (0);
                  }



/* Parent process*/
         else
         {
         printf("Parent Created with PID: %d\n",getpid());
         pid1 = wait(NULL);
         pid = fork();
         num2= num+1;



         if (pid<0)
                  {
                  fprintf(stderr, "Fork Failed");
                  exit (1);
                  }
         else if (pid ==0)
                  {
                  printf("Child %d Created with PID: %d\n",num2, getpid());
                  exit (0);
                  }
         /* both the process are being returned are printed here*/
         else
```

```
            {
            pid2 = wait(NULL);
            printf("Child %d with %d PID Completed\n",num, pid1);
            printf("Child %d with %d PID Completed\n",num2, pid2);


            }
return 0;
}
}
```

Program output:

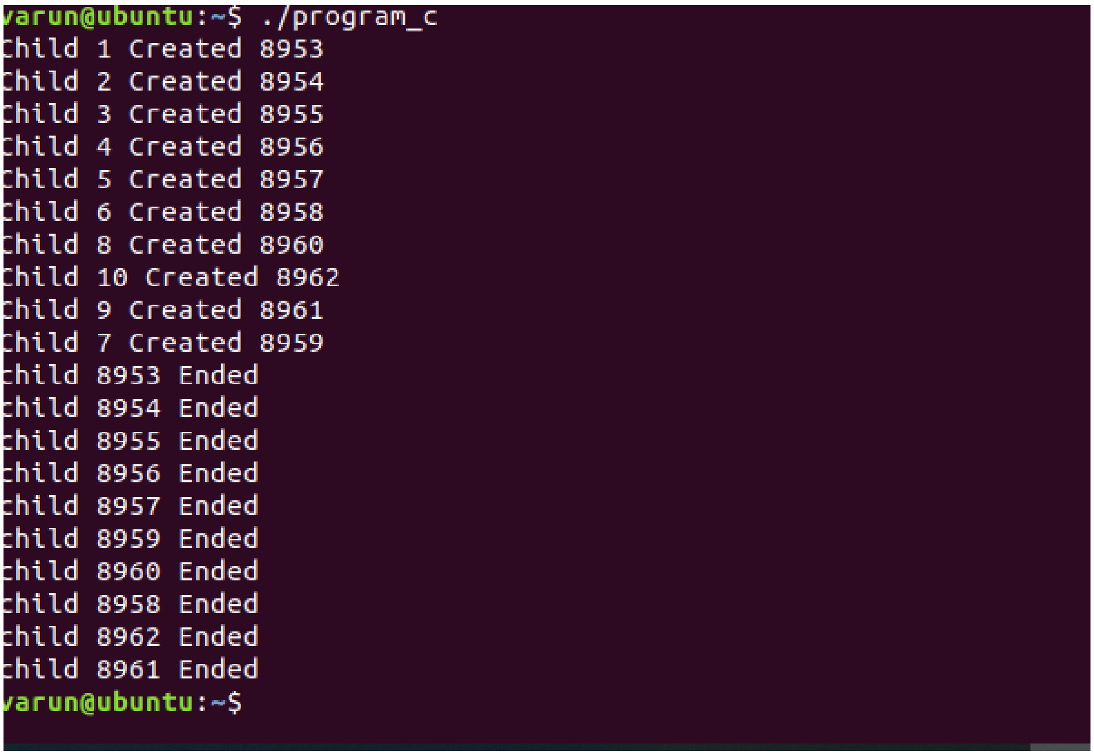


```
varun@ubuntu:~$ ./program_b
Parent Created with PID: 8476
Child 1 Created with PID: 8477
Child 2 Created with PID: 8478
Child 1 with 8477 PID Completed
Child 2 with 8478 PID Completed
```

Program _c:

```
/*Author : *Varun Subramanya

Student ID: 01705578*/


/* The standard header files that ar required by the program*/


#include<stdio.h>

#include<sys/wait.h>

#include<unistd.h>

#include<stdlib.h>


int main()

{

pid_t pid[10],pid1;

int o;


int i, count = 10, status;

int childNum;


/* Creating child processes */

for(i=0;i<count;i=i+1)

{

if((pid[i] = fork())< 0)

{

printf("\n Error");

exit (1);

}


else if (pid[i] == 0)


{
```

```
childNum = i+1;
printf("Child %d Created %d\n",childNum, getpid());
sleep (5);
exit (0);
}
}

/* parent process*/
/* wait for child to exit and remove pid values from the list */

o = count;
while (count>0)
        {
        pid1 = wait(&status);
        printf("child %d Ended\n", pid1);
        count= count - 1;
}
o= o - 1;
```

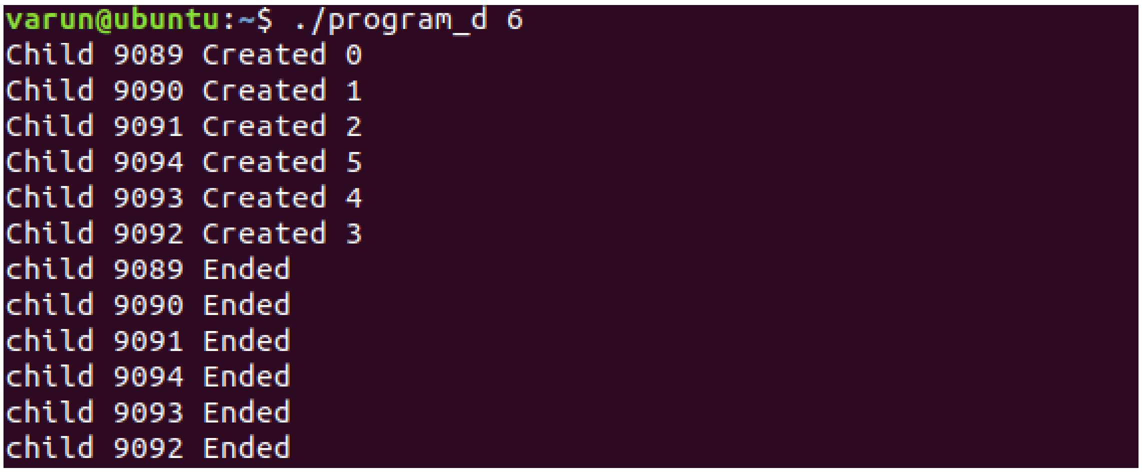


```
varun@ubuntu:~$ ./program_c
Child 1 Created 8953
Child 2 Created 8954
Child 3 Created 8955
Child 4 Created 8956
Child 5 Created 8957
Child 6 Created 8958
Child 8 Created 8960
Child 10 Created 8962
Child 9 Created 8961
Child 7 Created 8959
child 8953 Ended
child 8954 Ended
child 8955 Ended
child 8956 Ended
child 8957 Ended
child 8959 Ended
child 8960 Ended
child 8958 Ended
child 8962 Ended
child 8961 Ended
varun@ubuntu:~$
```

Program_d:

```
/*Author : *Varun Subramanya
Student ID: 01705578*/

/* The standard header files that ar required by the program*/
#include<stdio.h>
#include<sys/wait.h>
#include<unistd.h>
#include<stdlib.h>



int main(int argc, char *argv[])
{

unsigned int size = atoi(argv[1]);

pid_t pid[size], pid1;
int o;
int i, status;
int childNum;

/* Creating child processes */
for(i=0;i<size;i=i+1)
{
pid[i] = fork();

if(pid[i]< 0)



{
printf("\n Error");
exit (1);

}


else if (pid[i] == 0)


{

childNum = i+1;

printf("Child %d Created %d\n",getpid());

sleep(5);

exit (0);

}

}


/* parent process*/

/* wait for child to exit and remove pid values from the list */

sleep(5);

o = size;

while (size > 0)

        {

        pid1 = wait(&status);

        printf("child %d Ended \n", pid1);

        size= size - 1;

}

o = o-1;

}
```

Program_d output:



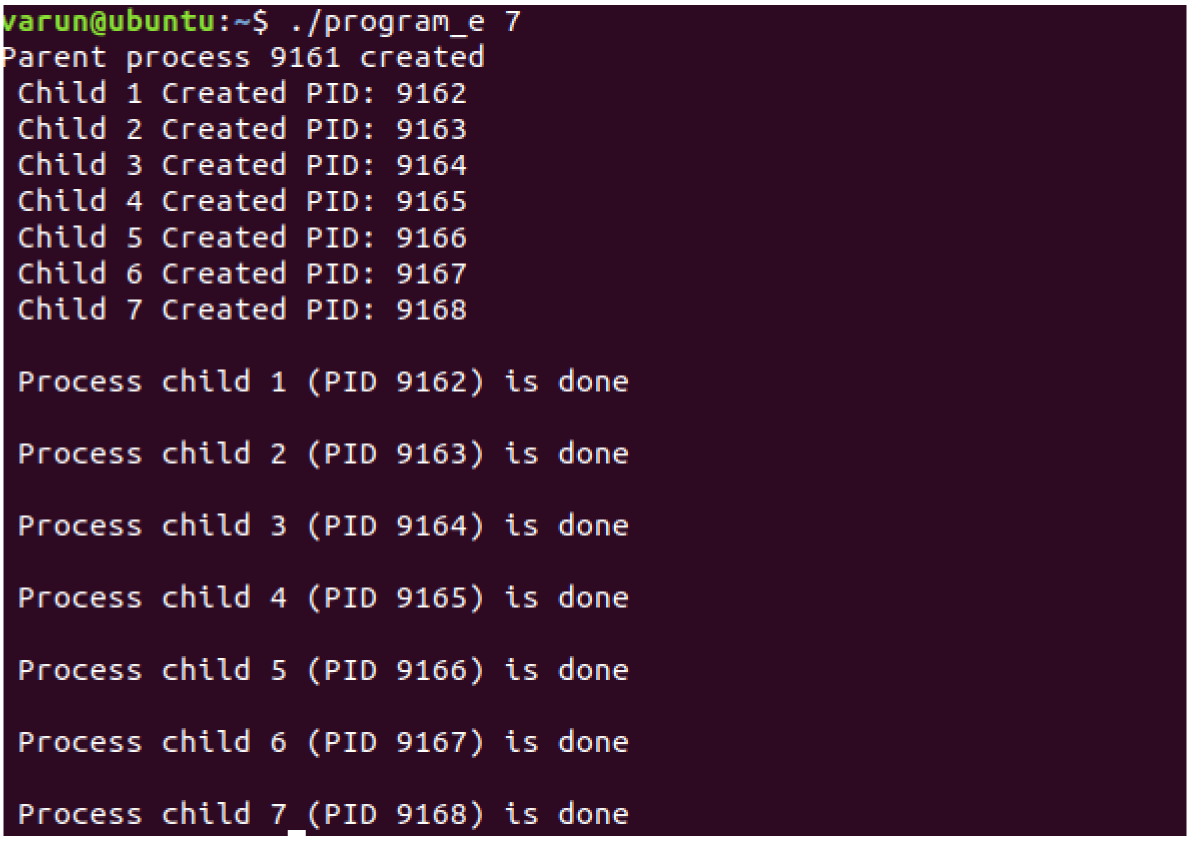
```
varun@ubuntu:~$ ./program_d 6
Child 9089 Created 0
Child 9090 Created 1
Child 9091 Created 2
Child 9094 Created 5
Child 9093 Created 4
Child 9092 Created 3
child 9089 Ended
child 9090 Ended
child 9091 Ended
child 9094 Ended
child 9093 Ended
child 9092 Ended
```

Program_e:

```
/*Author : *Varun Subramanya

Student ID: 01705578*/


/* The standard header files that ar required by the program*/


#include<stdio.h>

#include<sys/wait.h>

#include<unistd.h>

#include<stdlib.h>



int main(int argc, char *argv[])

{


unsigned int size = atoi(argv[1]);


pid_t pid[size], pid1;


int i = 0, x[i], status, y[i];

int childNum;

//childNum = i+1;

printf("Parent process %d created\n",getpid());

/* Creating child processes */

for(i=0;i<size;i=i+1)

{

        pid[i] = fork();


        if(pid[i] < 0)
```

```
            {
            printf("\n Error");
            exit (1);
            }

            else if (pid[i] == 0)

            {

            childNum = i+1;

            printf(" Child %d Created PID: %d \n",childNum, getpid());
            sleep (5);
            exit (0);
            }
            }

/* Child Exit Wait and Removal of PID */
/* Parent Process*/

int count = size;
            while (count >0)
            {
            pid1 = wait(&status);
            //pid1= wait(&stat
for (i =0 ; i<size; i++)
{
if (pid1 = pid[i])
            {

            childNum = i+1;
            printf("\n Process child %d (PID %d) is done\n", childNum, pid1);



}

count  --;

}

}


}
```

Program_e output:



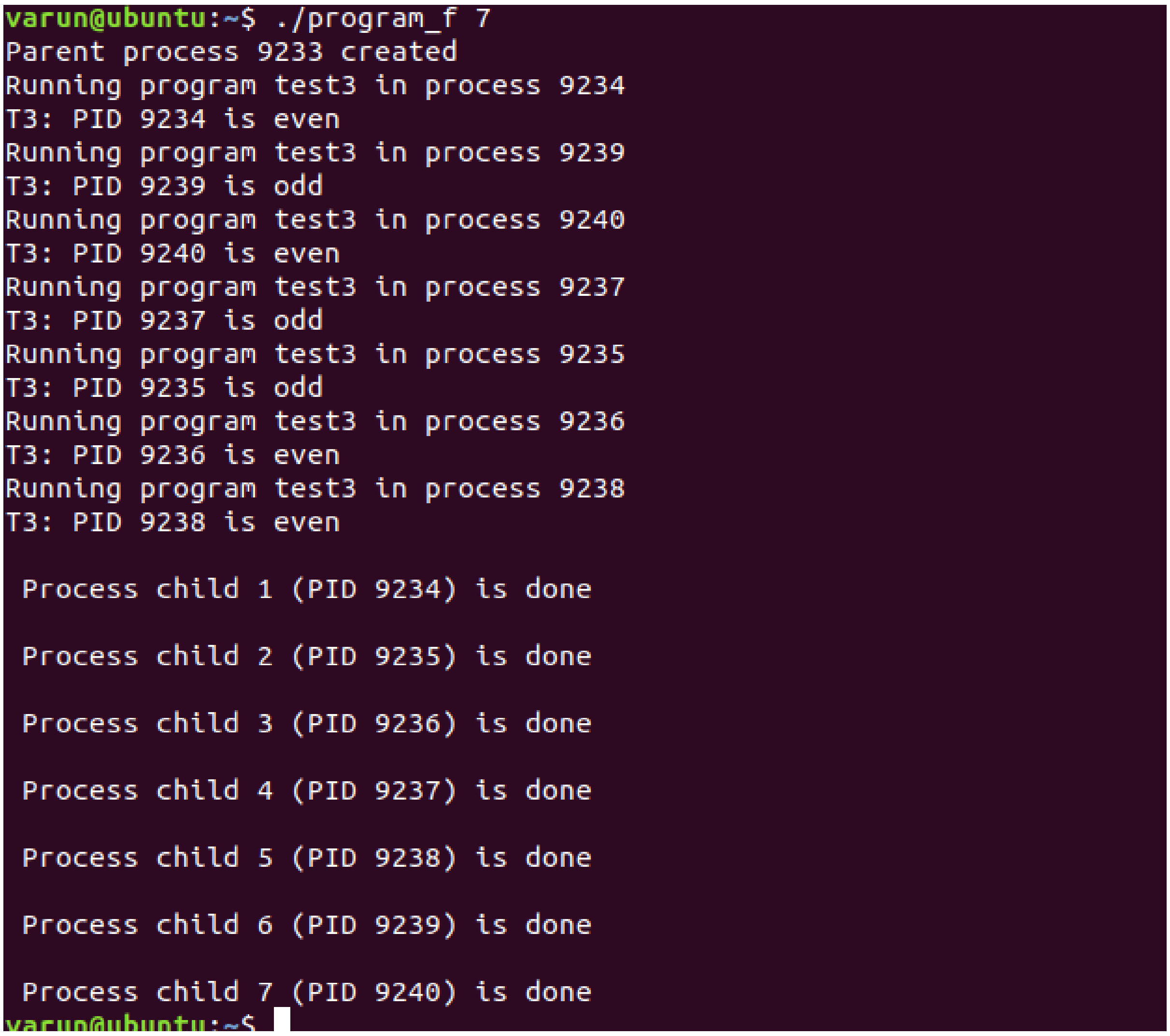
```
varun@ubuntu:~$ ./program_e 7
Parent process 9161 created
 Child 1 Created PID: 9162
 Child 2 Created PID: 9163
 Child 3 Created PID: 9164
 Child 4 Created PID: 9165
 Child 5 Created PID: 9166
 Child 6 Created PID: 9167
 Child 7 Created PID: 9168

 Process child 1 (PID 9162) is done

 Process child 2 (PID 9163) is done

 Process child 3 (PID 9164) is done

 Process child 4 (PID 9165) is done

 Process child 5 (PID 9166) is done

 Process child 6 (PID 9167) is done

 Process child 7 (PID 9168) is done
```

Program_f:

```
/* Varun Subramanya*/


#include<stdio.h>

#include<sys/wait.h>
```

```
#include<unistd.h>

#include<stdlib.h>



int main(int argc, char *argv[])

{


unsigned int size = atoi(argv[1]);


pid_t pid[size], pid1;


int i = 0, x[i], status, y[i];

int childNum;


//childNum = i+1;

printf("Parent process %d created\n",getpid());

/* Creating child processes */

for(i=0;i<size;i=i+1)

{

        pid[i] = fork();

        if(pid[i] < 0)




        {

        printf("\n Error");

        exit (1);

        }


        else if (pid[i] == 0)


        {
        childNum = i+1;
        //printf(" Child %d Created PID: %d \n",childNum, getpid());
        execlp("./test3","./test3",NULL);
        sleep (5);

//x[i] = getpid();
        //y[i] = childNum;
        //printf("%d\n", childNum);
        exit (0);
        }
        }

/* Child Exit Wait and Removal of PID */
/* Parent Process*/

int count = size;
        while (count >0)
        {
        pid1 = wait(NULL);
        sleep(5);
//pid1= wait(&status);
for (i =0 ; i<size; i++)
{
if (pid1 = pid[i])
        {

        childNum = i+1;
        printf("\n Process child %d (PID %d) is done", childNum, pid1);
        printf("\n");
}
count  --;
}


}


//while (1);
}


//pid1);


                //printf("%d \n", pid1);


/*//if (pid[size] == x[i])
//if (pid[i] = x[i])
if (pid[i] = x[i])
{
                //printf("\n %d and PID:%d", i, pid[i]);
                //i =0 ;
                childNum = i+1;
                printf("\n Process child %d (PID %d) is done", childNum, pid[i]);//pid1);
}
                //i--;*/
                //size--;
//}
```

Program output:



```
varun@ubuntu:~$ ./program_f 7
Parent process 9233 created
Running program test3 in process 9234
T3: PID 9234 is even
Running program test3 in process 9239
T3: PID 9239 is odd
Running program test3 in process 9240
T3: PID 9240 is even
Running program test3 in process 9237
T3: PID 9237 is odd
Running program test3 in process 9235
T3: PID 9235 is odd
Running program test3 in process 9236
T3: PID 9236 is even
Running program test3 in process 9238
T3: PID 9238 is even

 Process child 1 (PID 9234) is done

 Process child 2 (PID 9235) is done

 Process child 3 (PID 9236) is done

 Process child 4 (PID 9237) is done

 Process child 5 (PID 9238) is done

 Process child 6 (PID 9239) is done

 Process child 7 (PID 9240) is done
varun@ubuntu:~$
```

Program_g:

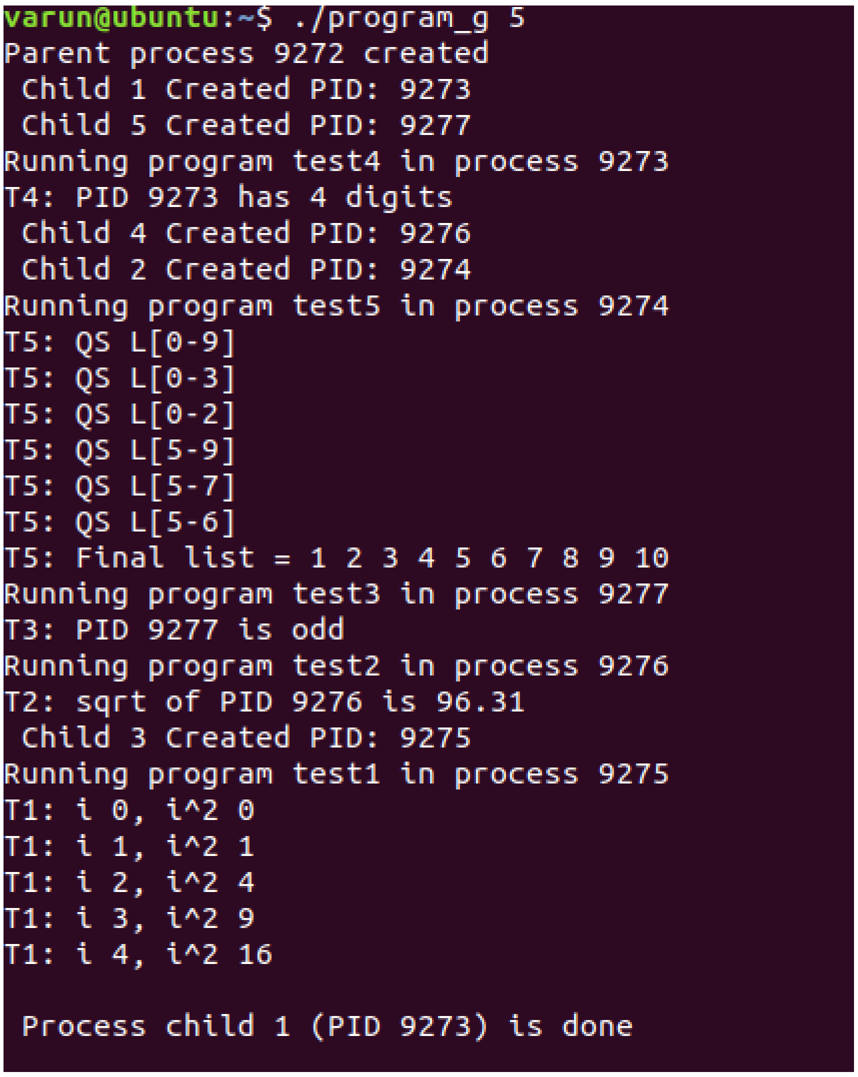
```
/* Varun Subramanya*/

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



int main(int argc, char *argv[])
{

unsigned int size = atoi(argv[1]);

pid_t pid[size], pid1;

int i = 0, x , status; //x[i], , y[i];
int childNum;
//childNum = i+1;
printf("Parent process %d created\n",getpid());
/* Creating child processes */
for(i=0;i<size;i=i+1)
{
        pid[i] = fork();
        if(pid[i] < 0)



        {
        printf("\n Error");
        exit (1);
```

```
		}

		else if (pid[i] == 0)

		{

		childNum = i+1;
		printf(" Child %d Created PID: %d \n",childNum, getpid());
//		for (z = 0; z <= childNum; z++)

x =getpid();


switch(x%5)
{
		case 0:
				execlp("./test1","./test1",NULL);
				break;
		case 1:
				execlp("./test2","./test2",NULL);
				break;
		case 2:
				execlp("./test3","./test3",NULL);
				break;
		case 3:
				execlp("./test4","./test1",NULL);
				break;
		case 4:
				execlp("./test5","./test5",NULL);
				break;
```

```
        default:
                printf("The PID Value is wrong/n");
                break;
}




        //execlp("./test3","./test3",NULL);
sleep (5);

//x[i] = getpid();
        //y[i] = childNum;
        //printf("%d\n", childNum);
exit (0);
        }
        }

/* Child Exit Wait and Removal of PID */
/* Parent Process*/

int count = size;
        while (count >0)
        {
        pid1 = wait(NULL);
        sleep(10);
//pid1= wait(&status);
for (i =0 ; i<size; i++)
{
if (pid1 = pid[i])
        {
```

```
        childNum = i+1;
        printf("\n Process child %d (PID %d) is done", childNum, pid1);
        printf("\n");

}
count  --;
}
exit (0);
}

//while (1);
}

//pid1);

        //printf("%d \n", pid1);

/*//if (pid[size] == x[i])
//if (pid[i] = x[i])
if (pid[i] = x[i])
{
        //printf("\n %d and PID:%d", i, pid[i]);
        //i =0 ;
        childNum = i+1;
        printf("\n Process child %d (PID %d) is done", childNum, pid[i]);//pid1);
}
        //i--;*/
        //size--;
//}
```

Program_g output:



```
varun@ubuntu:~$ ./program_g 5
Parent process 9272 created
 Child 1 Created PID: 9273
 Child 5 Created PID: 9277
Running program test4 in process 9273
T4: PID 9273 has 4 digits
 Child 4 Created PID: 9276
 Child 2 Created PID: 9274
Running program test5 in process 9274
T5: QS L[0-9]
T5: QS L[0-3]
T5: QS L[0-2]
T5: QS L[5-9]
T5: QS L[5-7]
T5: QS L[5-6]
T5: Final list = 1 2 3 4 5 6 7 8 9 10
Running program test3 in process 9277
T3: PID 9277 is odd
Running program test2 in process 9276
T2: sqrt of PID 9276 is 96.31
 Child 3 Created PID: 9275
Running program test1 in process 9275
T1: i 0, i^2 0
T1: i 1, i^2 1
T1: i 2, i^2 4
T1: i 3, i^2 9
T1: i 4, i^2 16

 Process child 1 (PID 9273) is done
```