Assignment No. 2:
Process control system calls: The demonstration of FORK, EXECVE and WAIT system calls along with zombie and orphan states.
A. Implement the C program in which the main program accepts the integers to be sorted. Main program uses the FORK system call to create a new process called a child process. Parent process sorts the integers using a sorting algorithm and waits for the child process using the WAIT system call to sort the integers using any sorting algorithm. Also demonstrate zombie and orphan states.

Printing the Process ID

getpid() to get Process Id
getppid() to get Parent Process Id of the current process
These functions are declared in <unistd.h> header file

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

int main()
{
        int p_id, p_pid;

        p_id = getpid(); /*process id*/
        p_pid = getpid(); /*parent process id*/

        printf("Process ID: %d\n", p_id);
        printf("Parent Process ID: %d\n", p_pid);

        return 0;
}
```

Output:

```
ubuntu@ubuntu-Vostro-460:~$ gcc sample.c -o sample
ubuntu@ubuntu-Vostro-460:~$ ./sample
Process ID: 8826
Parent Process ID: 8826
```

Example 2
Using the system call

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
int main()
{

   // make two process which run same
   // program after this instruction
   pid_t p = fork();
   if(p<0){
   perror("fork fail");
   exit(1);
   }
   printf("Hello world!, process_id(pid) = %d \n",getpid());
   return 0;
}
```

Output:

```
ubuntu@ubuntu-Vostro-460:~$ ./sample
Hello world!, process_id(pid) = 7769
Hello world!, process_id(pid) = 7770
```

Example 3
Using fork to duplicate a program's process

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
int main()
{
pid_t child_pid;
printf("The main program process ID is %d\n", (int) getpid());
child_pid=fork();
if(child_pid!=0)
{
printf("This is the parent process ID, with id %d\n", (int)
getpid());
printf("The child process ID is %d\n", (int) child_pid);
```

```
}
else
printf(“This is the child process ID, with id %d\n”, (int)
getpid());
return 0;
}
```

Output:

```
ubuntu@ubuntu-Vostro-460:~$ ./sample
Hello world!, process_id(pid) = 8051
Hello world!, process_id(pid) = 8052
```

Example 4:
Determining the exit status of a child

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/wait.h>
void show_return_status(void)
{
pid_tchildpid;
int status;
childpid = wait(&status); if (childpid == -1)
perror("Failed to wait for child");
else if (WIFEXITED(status))
printf("Child %ld terminated with return status %d\n", (long)childpid,
WEXITSTATUS(status));
}
```

Output:

```
ubuntu@ubuntu-Vostro-460:~$ ./sample
Hello world!, process_id(pid) = 8243
Hello world!, process_id(pid) = 8244
```

Example 5
A program that creates a child process to run ls -l

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

```
#include <unistd.h>
#include <sys/wait.h>
int main(void)
{
pid_tchildpid;
childpid = fork();
if (childpid == -1) {
perror("Failed to fork");
return 1;
}
if (childpid == 0) {
/* child code */
execl("/bin/ls", "ls", "-l", NULL);
perror("Child failed to exec ls"); return 1;
}
if (childpid != wait(NULL)) {
/* parent code */
perror("Parent failed to wait due to signal or error"); return 1;
}
return 0;
}
```

Output:

```
ubuntu@ubuntu-Vostro-460:~$ ./sample
Hello world!, process_id(pid) = 8451
Hello world!, process_id(pid) = 8452
```

Example 6

Making a zombie process

```
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>
int main()
{
pid_tchild_pid;
//create a child process
child_pid=fork();
if(child_pid>0) {
//This is a parent process. Sleep for a minute
sleep(60)
```

```
}
else
{
//This is a child process. Exit immediately.
exit(0);
}
return 0;
}
```

Output:

```
ubuntu@ubuntu-Vostro-460:~$ ./sample
Hello world!, process_id(pid) = 8601
Hello world!, process_id(pid) = 8602
```

Example 7

Demonstration of fork system call

```
#include<stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>
int main()
{
pid_tpid;
char *msg;
int n;
printf("Program starts\n");
pid=fork();
switch(pid)
{
case -1:
printf("Fork error\n");
exit(-1);
case 0:
msg="This is the child process";
n=5;
break;
default:
msg="This is the parent process";
n=3;
break;
```

```
}
while(n>0)
{
puts(msg);
sleep(1);
n--;
}
return 0;

}
```

Output:

```
ubuntu@ubuntu-Vostro-460:~$ ./sample
Hello world!, process_id(pid) = 8745
Hello world!, process_id(pid) = 8746
```

Example 8

Demo of multiprocess application using fork()system call

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

#define SIZE 1024
void do_child_proc(intpfd[2]);
void do_parent_proc(intpfd[2]);
int main()
{
intpfd[2];
intret_val,nread;
pid_tpid;
ret_val=pipe(pfd);
if(ret_val==-1)
{
perror("pipe error\n");
exit(ret_val);
}
pid=fork();
switch(pid)
```

```
{
case -1:
printf(“Fork error\n”);
exit(pid);
case 0:
do_child_proc(pfd);
exit(0);
default:
do_parent_proc(pfd);
exit(pid);

}
wait(NULL);
return 0;
}
void do_child_proc(intpfd[2])
{
intnread;
char *buf=NULL;
printf(“5\n”);
close(pfd[1]);
while(nread=(read(pfd[0],buf,size))!=0)
printf(“Child Read=%s\n”,buf);
close(pfd[0]);
exit(0);
}
void do_parent_proc(intpfd[2])
{
char ch;
char *buf=NULL;
close(pfd[0]);
while(ch=getchar()!=’\n’) {
printf(“7\n”);
*buf=ch;
buff+;
}
*buf=’\0’;
write(pfd[1],buf,strlen(buf)+1);
close(pfd[1]);
```

```
}
```

Output:

```
ubuntu@ubuntu-Vostro-460:~$ ./sample
Hello world!, process_id(pid) = 8929
Hello world!, process_id(pid) = 8930
```