Lab 3 - Process and Signal

1. Write a C program to block a parent process until the child completes using a wait system call.

Code:

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

int main(int argc, char const *argv[])
{
        pid_t pid;
        int status;
        pid = fork();
        if(pid<0){
                perror("fork failed\n");
                exit(1);
        }
        else if(pid==0){
                        printf("Child running.......\nChild completed\n");
        }
        else{
          wait(&status);
          printf("Parent resumed\n");
          printf("Child returned : %d\n",status);
        }

        return 0;
}
```

O/P:

```
180905032@selab-19:~/OS/lab2/part1$ cc -o q1 q1.c
180905032@selab-19:~/OS/lab2/part1$ ./q1
Child running.......
Child completed
Parent resumed
Child returned : 0
180905032@selab-19:~/OS/lab2/part1$
```

2. Write a C program to load the binary executable of the previous program in a child process using the exec system call.

Code:

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

int main(int argc, char const *argv[])
{
        pid_t pid;
        int status;
        pid = fork();
        if(pid<0){
                perror("fork failed\n");
                exit(1);
        }
        else if(pid==0){
                    printf("Child process\nBeginning execution of q1.c\n");
                    printf("****************\n");
                         execlp("../part1/q1","q1",NULL);

        }
        else{
        wait(NULL);
                   printf("****************\n");
        printf("Child completed\n");

        }

        return 0;
}
```

O/P:

```
180905032@selab-19:~/OS/lab2/part1$ cc -o q2 q2.c
180905032@selab-19:~/OS/lab2/part1$ ./q2
Child process
Beginning execution of q1.c
****************
Child running.......
Child completed
Parent resumed
Child returned : 0
****************
Child completed
180905032@selab-19:~/OS/lab2/part1$
```

3. Write a program to create a child process. Display the process IDs of the process, parent and child (if any) in both the parent and child processes.

Code:

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

int main(int argc, char const *argv[])
{
        pid_t pid,c_id,p_id;
        int status;
        pid = fork();
        if(pid<0){
                perror("fork failed\n");
                exit(1);
        }
        else if(pid==0){
                c_id=getpid();
                        p_id=getppid();
                        printf("Child process with pid %d\n",c_id);
                        printf("PID of parent : %d\n",p_id);
        }
        else{
        wait(NULL);
         c_id=getpid();
                        p_id=getppid();
         printf("Parent process with pid %d\n",c_id);
                        printf("PID of parent : %d\n",p_id);
                        printf("PID of child : %d\n",pid);
                }

        return 0;
}
```

O/P:

```
180905032@selab-19:~/OS/lab2/part1$ cc -o q3 q3.c
180905032@selab-19:~/OS/lab2/part1$ ./q3
Child process with pid 4440
PID of parent : 4439
Parent process with pid 4439
PID of parent : 3767
PID of child : 4440
180905032@selab-19:~/OS/lab2/part1$
```

4. Create a zombie (defunct) child process (a child with exit() call, but no corresponding wait() in the sleeping parent) and allow the init process to adopt it (after parent terminates). Run the process as a background process and run the "ps" command.

Code:

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

int main(int argc, char const *argv[])
{
	pid_t pid;
	int status;
	pid = fork();
	if(pid<0){
		perror("fork failed\n");
		exit(1);
	}
	else if(pid==0){
			printf("Child started...\nChild completed\n");
	}
	else{

	printf("Parent process %d\n",getpid());
	while(1);
	}

	return 0;
}
```

O/P:

```
Terminal
180905032@selab-19: ~/OS/lab2/part1
180905032@selab-19:~/OS/lab2/part1$ cc -o q3 q3.c
180905032@selab-19:~/OS/lab2/part1$ ./q3
Child process with pid 4440
PID of parent : 4439
Parent process with pid 4439
PID of parent : 3767
PID of child : 4440
180905032@selab-19:~/OS/lab2/part1$ cc -o q4 q4.c
q4.c: In function 'main':
q4.c:23:2: error: expected expression before '}' token
  }
  ^
180905032@selab-19:~/OS/lab2/part1$ cc -o q4 q4.c
180905032@selab-19:~/OS/lab2/part1$ ./q4
Parent process
Child started...
Child completed
^C
180905032@selab-19:~/OS/lab2/part1$ cc -o q4 q4.c
180905032@selab-19:~/OS/lab2/part1$ ./q4
Parent process 4647
Child started...
Child completed
4288 ?        00:00:00 kworker/0:2
4365 ?        00:00:02 Web Content
4408 ?        00:00:00 kworker/0:1
4453 ?        00:00:00 kworker/1:1
4489 ?        00:00:00 kworker/u8:1
4527 ?        00:00:00 Web Content
4566 ?        00:00:00 kworker/3:2
4580 ?        00:00:00 kworker/2:0
4604 pts/17   00:00:00 bash
4620 ?        00:00:00 kworker/0:0
4647 pts/2    00:03:36 q4
4648 pts/2    00:00:00 q4 <defunct>
4650 ?        00:00:00 kworker/3:0
4669 ?        00:00:00 kworker/u8:2
4675 pts/17   00:00:00 ps
180905032@selab-19:~/OS/lab2/part1$
```

```
180905032@selab-19:~/OS/lab2/part1$ ps -A
  PID TTY          TIME CMD
    1 ?        00:00:01 systemd
    2 ?        00:00:00 kthreadd
    3 ?        00:00:00 ksoftirqd/0
    5 ?        00:00:00 kworker/0:0H
    7 ?        00:00:00 rcu_sched
    8 ?        00:00:00 rcu_bh
    9 ?        00:00:00 migration/0
   10 ?        00:00:00 watchdog/0
   11 ?        00:00:00 watchdog/1
   12 ?        00:00:00 migration/1
   13 ?        00:00:00 ksoftirqd/1
```

```
4580 ?        00:00:00 kworker/2:0
4604 pts/17   00:00:00 bash
4620 ?        00:00:00 kworker/0:0
4647 pts/2    00:03:36 q4
4648 pts/2    00:00:00 q4 <defunct>
4650 ?        00:00:00 kworker/3:0
4669 ?        00:00:00 kworker/u8:2
4675 pts/17   00:00:00 ps
180905032@selab-19:~/OS/lab2/part1$
```