**Lab 3 - Process and Signal**

**Lab Exercises**

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

#include<sys/types.h>

#include<sys/wait.h>

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

{

// printf("Q1 program executing\n");

int status;

pid\_t pid;

pid = fork();

if(pid == -1)

{

printf("Error creating child process\n");

}

else if(pid == 0)

{

printf("I'm the child process\n");

exit(0);

}

else

{

wait(&status);

printf("I'm the parent process\n");

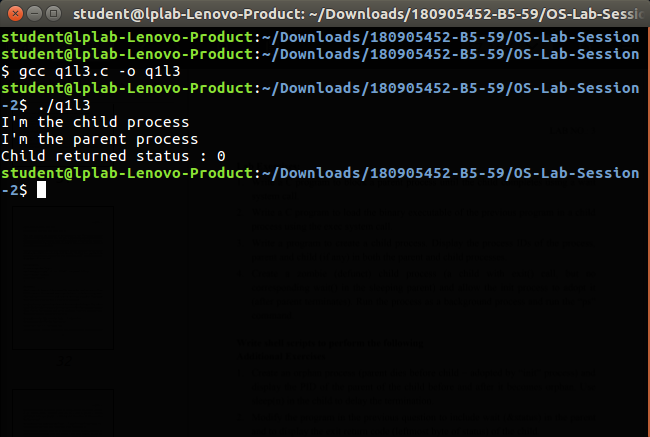
printf("Child returned status : %d\n",status);

}

return 0;

}

Output



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

#include<stdio.h>

#include<stdlib.h>

#include<unistd.h>

#include<dirent.h>

#include<sys/types.h>

#include<sys/wait.h>

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

{

pid\_t pid;

pid = fork();

if(pid < 0)

{

fprintf(stderr,"Error in creating child process\n");

exit(-1);

}

else if(pid == 0)

{

execlp("./q1l3.o","q1l3",NULL);

}

else

{

wait(NULL);

printf("Child complete\n");

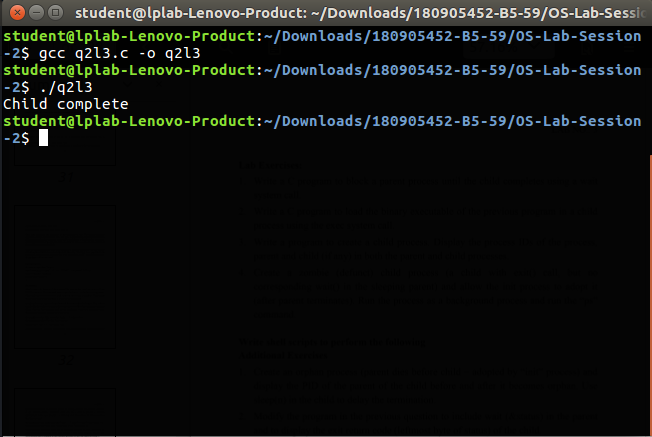
exit(0);

}

return 0;

}

Output



1. 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.

#include<stdio.h>

#include<stdlib.h>

#include<unistd.h>

#include<dirent.h>

#include<sys/types.h>

#include<sys/wait.h>

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

{

pid\_t pid;

pid = fork();

if(pid < 0)

{

printf("Error creating child process\n");

exit(-1);

}

else if(pid == 0)

{

printf("In child process\n");

printf("PID = %d\n",getpid());

printf("Parent PID = %d\n",getppid());

printf("child PID = %d\n",pid);

}

else

{

wait(NULL);

printf("\nIn the parent process\n");

printf("PID = %d\n",getpid());

printf("Parent PID = %d\n",getppid());

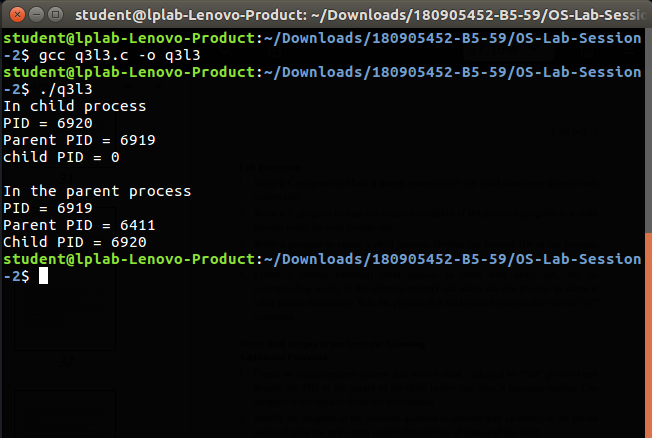
printf("Child PID = %d\n",pid);

}

return 0;

}

Output



1. 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.

#include<stdio.h>

#include<stdlib.h>

#include<unistd.h>

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

{

pid\_t pid;

pid = fork();

if(pid < 0)

{

printf("Error\n");

exit(-1);

}

if(pid == 0)

{ //child process

printf("child process\n");

exit(0);

}

else

{ //parent process

sleep(2);

printf("parent process\n");

}

return 0;

}

Output

