Objectives

- To create parent-child processes using fork() system call.
- To create a process chain and prevent child processes from becoming zombie using wait() system call.
- To create a process fan and prevent child processes from becoming zombies using a wait() system call.
- To create a process tree and prevent child processes from becoming zombie using wait() system call.
- To execute a new process from a process using an exec() system call.

Pre-Lab Tasks

1. Testing of gcc compiler

i) type a source file with printf "helloworld" msg

ii) compilation statement : *$gcc -o helloworld.out helloworld.c*

1. Display of command line arguments

Type the code below in the file cmdargs.c, and then compile and execute with arguments from the command line and write the output

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

int i;

for(i=0;i<argc;i++)

     printf("%s\n",argv[i]);

}
```

In-Lab Tasks

header file*: unistd.h, stdio.h*

System call: *fork(), getpid(), getppid(),wait()*

Task1: Compile and run the following program and study its behavior

a) Write the output of the child process

b) Write the output of the parent process

c) *Mention any unusual thing noticed in the output of child process*

```
#include <stdio.h>

#include <unistd.h>


int main() {

int pid;

pid = fork();

if(pid > 0)


printf("I am parent: my process id is %d and my child
process id is %d\n", getpid(),pid);


else if (pid == 0)

printf("I am child: my process id is %d and my parent
process id is %d\n",getpid(), getppid());

else

printf("ERROR in executing fork()");


return 0;

}
```

Task2a: Compile and run the following code and write the output. From the output Also draw the process interconnected diagram to show process chain or fan or tree.

```
#include <stdio.h>

#include <unistd.h>


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

int pid;

int i;

for(i=0;i<4;i++){

     pid = fork();

     if(pid > 0)

          break;

     else if(pid == 0)

          continue;

     else

          printf("ERROR: In fork()");

}//end of for

printf("My process id is %d and my Parent process id is
%d\n",getpid(),getppid());

}//end of main
```

Task 2b: Modify the code to get the number of process value from the command line argument. Compile and run the program with number of process value other than 4 and write the output.

```
#include <stdio.h>

#include <unistd.h>


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

int pid;

int i;
```

```
printf("My process id is %d and my Parent process id is
%d\n",getpid(),getppid());

}//end of main
```

Output:

Task 2c: Write the modified code to prevent the parent process from terminating before the child processes through wait() system call and also write the output

system call: wait()

```
#include <stdio.h>

#include <unistd.h>


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

int pid;

int i;
```

```
printf("My process id is %d and my Parent process id is %d\n",getpid(),getppid());

}//end of main
```

Output:

Task 3a: Compile and run the following code and write the output. From the output Also draw the process interconnected diagram to show process chain or fan or tree.

```
#include <stdio.h>

#include <unistd.h>


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

int pid;

int i;

for(i=0;i<4;i++){

     pid = fork();

     if(pid > 0)

          continue;

     else if(pid == 0)

          break;

     else

          printf("ERROR: In fork()");

}//end of for

printf("My process id is %d and my Parent process id is
%d\n",getpid(),getppid());

}//end of main
```

Output:

Task 3b: Write the modified code to get the number of process value from the command line argument. Compile and run the program with a number of process value other than 4 and write the output.

```
#include <stdio.h>

#include <unistd.h>


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

int pid;

int i;











printf("My process id is %d and my Parent process id is
%d\n",getpid(),getppid());

}//end of main
```

Output:

Task 3c: Write the modified code to prevent the parent process from terminating before the child processes through wait() system call and also write the output

system call: wait()

```
#include <stdio.h>

#include <unistd.h>


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

int pid;

int i;











printf("My process id is %d and my Parent process id is
%d\n",getpid(),getppid());

}//end of main
```

Output:

Task 4a: Creating the Process Tree, write the code and output similarly as in process chain and process fan with n=4

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

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




printf("My process id is %d and my Parent process id is
%d\n",getpid(),getppid());
}//end of main
```

Output:

Task4c: In Process Tree with n=4, Prevent the parent process from terminating before child processes and write the modified code and output

system call: wait()

hint: `while(wait()>0);`

```
#include <stdio.h>

#include <unistd.h>

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

int pid;

int i;

printf("My process id is %d and my Parent process id is
%d\n",getpid(),getppid());

}//end of main
```

Output:

Task5: write the code to create a child process and execute ls -l command in the child process

system call: exec()

hint: explore the variants exec() call