**Task#1:**

**fork() function** command creates another process, i.e. the new cretaed process is duplication of the parent process and is known as child process. Parent process is the ne which called fork() function. fork() function returns a PID (process ID) of the child in the parent process.

**exec() function** command replaces the current program space with some other program as provided by the user. Once it is called, the code lines after the exec() are totally replaced by a set of other instructions as indicated by the user.

**Task#2:**

**Program#1:**

int main()

{

                int pid;

                pid = getpid();

                printf(“Process ID is %d\n”, pid);

                return 0;

}

getpid() function returns the process id of the current process and it is being stored in the variable pid. Printf shows the pid(process id of the current process) on the screen

**Program#2:**

int main()

{

                long i;

                printf(“Process ID is %d\n”, getpid());

                for(i=0; i<=400;i++)

                {

                                printf(“i is %d\n”, i);

                }

                return 0;

}

getpid() function returns the process id of the current process and printf shows it on the screen after printing the “Process ID is”, “\n” moves the cursor to next line and the the process id is printed.

The for-loop will show 401 times the line “i is” and value of i will be from 0-400

**Program#3:**

int main()

{

                int ppid;

                ppid = getppid();

                printf(“Parent Process ID is %d\n”, ppid);

                return 0;

}

getppid returns the process id of the parent(the one who created the current process) of the current process.

**Program#4:**

int main()

{

                fork();

                printf(Hello World\n”);

                return 0;

}

fork() command creats a new child process. The child process is duplicated of the parent process. Here, all the code after fork() function is duplicated to a child process. This program prints “Hello World” one time for child process and the return to parent process and again prints Hello World. In total Hello World is printed twice.

**Prgoram#5:**

int main()

{

                printf(“This is to demonstrate the fork() \n”);

                fork();

                return 0;

}

fork() command creats a new child process. The child process is duplicated of the parent process. Here, all the code after fork() function is duplicated to a child process. But this time one ouput is shown on the monitor as printf line is before the fork command so it will run only in case of parent process.

**Program#6:**

int main()

{

                fork();

                fork();

                printf(“Hello World\n”);

                return 0;

}

In this program 1st fork creates a child and second fork creates another child of the child i.e grand child, thus creating a hierarchy. 1st the parent is executed then 1st chuild then chid of 1st child, since each process containts printf line so Hello World is printed 4 times on the screen

**Program#7:**

int main()

{

                fork();

                printf(“The PID is %d\n”, getpid());

                return 0;

}

Fork() command creates a child, and getpid 1st prints the process id of the current process(1st time it is the child and then the parent process)

**Program#8:**

int main()

{

                int pid;

                pid = fork();

                if(pid > 0)

                                printf(“Parent Process ID is  %d\n”, pid);

                return 0;

}

Once a fork() call is made, it returns some value. This value is 0 in case the pid (variable) is in the child’s process. While in the parent’s process, the value of pid is the ID of child which is not 0. If(pid>0)Conditon is only true, once it is in parent process So, the print message will only be printed once

**Program#9:**

int main()

{

                int pid;

                pid = fork();

                if(pid == 0)

                                printf(“Child Process\n”);

                return 0;

}

Fork() command creates a child process and If(pid==0) Conditon is only true, once it is in child process So, the print message will only be printed once

**Program#10:**

int main()

{

                int pid;

                pid = fork();

                if(pid==0)

                {

                                printf(“I am the child, my process ID is   %d\n”, getpid());

                                printf(“The child’s parent  process ID is   %d\n”, getppid());

                }

                else

                {

                                printf(“I am the parent, my process ID is   %d\n”, getpid());

                                printf(“The parent  process ID is   %d\n”, getppid());

                }

                return 0;

}

In this program, the else part executes first. This shows that whenever fork() call is made, first the parent program is given the CPU resource.

First the pid of the parent is printed and then ppid (parent pid).

After that, pid and ppid of child is printed.

**Task#3:**

**Program#11:**

int main()

{

                printf(“Before exec my id is %d \n”, getpid());

                printf(“My parent process id is  %d \n”, getppid());

                printf(“exec strats\n”);

                execl(“Program#12”, “ex2”, (char\*)0);   //set path

                printf(“This will not print\n”);

                return 0;

}

**Program#12:**

int main()

{

                printf(“After the exec my process id is %d\n”, getpid());

                printf(“My parent  process id is %d\n”, getppid());

                printf(“exec  ends\n”);

                return 0;

}

Program11 starts noraml execution and 1st three lines are executed and messages are displayed on the screen that contains process id, parent process id and exec starts line. At 4th line the program11 code is replaced by the code of program12 and following lines are executed

printf(“After the exec my process id is %d\n”, getpid());

            printf(“My parent  process id is %d\n”, getppid());

             printf(“exec  ends\n”);

 The process id and parent process id values are the same for both as lines are executing in the same program.