Assignment #2

Name: Mohsin Khan

Reg #2017-kiu-2588

Course: Operating System

**solve the below given System calls code in GCC.**

**Fork**

**Exit**

**Exec.**

**Fork()**

#include <stdio.h>

#include <sys/types.h>

#include <unistd.h>

int main()

{

    // make two process which run same

    // program after this instruction

    fork();

    printf("Hello world!\n");

    return 0;

}

**Output:**

Hello world!

Hello world!

**Exec() Fork() Exit()**

// C program to illustrate  use of fork() &

// exec() system call for process creation

#include <stdio.h>

#include <sys/types.h>

#include <unistd.h>

#include <stdlib.h>

#include <errno.h>

#include <sys/wait.h>

int main(){

   pid\_t  pid;

   int ret = 1;

   int status;

   pid = fork();

   if (pid == -1){

      // pid == -1 means error occured

      printf("can't fork, error occured\n");

      exit(EXIT\_FAILURE);

   }

   else if (pid == 0){

      // pid == 0 means child process created

      // getpid() returns process id of calling process

      // Here It will return process id of child process

      printf("child process, pid = %u\n",getpid());

      // Here It will return Parent of child Process means Parent process it self

      printf("parent of child process, pid = %u\n",getppid());

      // the argv list first argument should point to

      // filename associated with file being executed

      // the array pointer must be terminated by NULL

      // pointer

      char \* argv\_list[] = {"ls","-lart","/home",NULL};

      // the execv() only return if error occured.

      // The return value is -1

      execv("ls",argv\_list);

      exit(0);

   }

   else{

      // a positive number is returned for the pid of

      // parent process

      // getppid() returns process id of parent of

      // calling process

// Here It will return parent of parent process's ID

      printf("Parent Of parent process, pid = %u\n",getppid());

      printf("parent process, pid = %u\n",getpid());

        // the parent process calls waitpid() on the child

        // waitpid() system call suspends execution of

        // calling process until a child specified by pid

        // argument has changed state

        // see wait() man page for all the flags or options

        // used here

        if (waitpid(pid, &status, 0) > 0) {

            if (WIFEXITED(status) && !WEXITSTATUS(status))

              printf("program execution successful\n");

            else if (WIFEXITED(status) && WEXITSTATUS(status)) {

                if (WEXITSTATUS(status) == 127) {

                    // execv failed

                    printf("execv failed\n");

                }

                else

                    printf("program terminated normally,"

                       " but returned a non-zero status\n");

            }

            else

               printf("program didn't terminate normally\n");

        }

        else {

           // waitpid() failed

           printf("waitpid() failed\n");

        }

      exit(0);

   }

   return 0;

}