**LAB # 4**

**Exec() System Call**

**OBJECTIVE**

Study the features of exec system call.

**THEORY**

**The 'exec( )'**

Forking provides a way for an existing process to start a new one, but what about the case where the new process is not part of the same program as parent process? This is the case in the shell; when a user starts a command it needs to run in a new process, but it is unrelated to the shell.

This is where the exec system call comes into play. exec will replace the contents of the currently running process with the information from a program binary. Thus the process the shell follows when launching a new program is to firstly fork, creating a new process, and then exec (i.e. load into memory and execute) the program binary it is supposed to run.

**The versions of exec are:**

|  |  |
| --- | --- |
| execl | (const char \*path, const char \*arg, ...); |
| execlp | (const char \*file, const char \*arg, ...); |
| execle | (const char \*path, const char \*arg , ..., char \* const envp[]); |
| execv | (const char \*path, char \*const argv[]); |
| execvp | (const char \*file, char \*const argv[]); |

**Example programs**

**Program 1**

main ( ) {

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

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

printf(“exec starts\n”);

execl(“/usr/guest/ex2”,”ex2”,(char\*)0);

printf(“this will not print\n”);

}

**Program 2**

main( ) {

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

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

printf(“exec ends\n”);}

**Program 3**

main(){

pid\_tpid;

pid=fork();

if(pid<0)

{

fprintf(stderr,”Fork Failed”);

return 1; }

else if(pid==0){

execlp(“/bin/ls”,”ls”,NULL);

}

else{

wait(NULL);

printf(“Child Complete”);

}

return 0;

}

**Exercise**

1. Describe all the versions of exec system call.
2. Write a program that will call another program from child process.