Understanding exec() system call

Goal: In this lab you will write programs using the exec() family of system calls.

The TAs will discuss the review problems (source files given) and the programming exercise. To finish, you will have to write two programs and demonstrate them to a TA.

This lab will introduce you to the exec() family of system calls. The exec family of functions are used to replace the current process image with a new process image.

There are several different flavours of the **exec()** call, but they all perform essentially the same task. The following is a list of all six **exec()** variants and the parameter numbers and types they take:

```
int execl(pathname, argo, ..., argn, 0);
int execv(pathname, argv);
int execlp(cmdname, arg0, ..., argn, 0);
int execvp(cmdname, argv);
int execle(pathname, arg0, ..., arga, 0, envp);
int execve(pathname, argv, envp);
```

Review Problems

The code is given in following files with suitable comments.
lab2-prog2.c
lab2-prog3.c

Follow the comments and attempt understanding these programs based on the obtained output.

Assignment Exercise 1

Write a program prog_exec that takes a command-line argument of the path to the program (absolute or relative), forks a child that tries to exec() the given program, and reports on its success or failure. A child that exits with a status of 0 is assumed to be success, non-zero is a failure. If the exec() fails, the child should print why (via perror()), and exit with a non-zero status.

This program does not have to support command line arguments to the other program. The working of the program is shown below.

Parent	**Child**
---- before the fork ----	
check the command line args	----
fork()	
---- after the fork ----	
wait() for child	exec the given program
---- after the exit ----	
report on child's success	----

Example Runs

```
$ ./prog_exec
usage: prog_exec command

$ ./prog_exec command with args
usage: prog_exec command

$ ./prog_exec non-existant
non-existant: No such file or directory
Process 2359 exited with an error value.

$ ./prog_exec ls
ls: no such file or directory
Process 2361 exited with an error value.

$ ./prog_exec /bin/ls
Makefile RCS forkit forkit.c prog_exec prog_exec.c
Process 2369 succeeded.

$ ./prog_exec /bin/false
Process 2371 exited with an error value.

$ ./prog_exec /bin/true
Process 2373 succeeded.
```

As seen in the example above, /bin/true always exits with a successful exited code, and /bin/false always exits with an unsuccessful one.

Assignment Exercise 2

Write a C program which will execute the following series of Linux commands:

```
cp /etc/fstab .
sort fstab -o myfstab
cat myfstab
```

Hint: The best way to guarantee a particular order of execution for the three commands is to **fork()** child processes to execute each of the commands before executing the next. For e.g.,

```
if (!fork())
      {
            execlp("cp", ……………….);
            printf("error executing cp\n");
            exit(1);
      }

if (!fork())
      {
            execlp("sort",……………………….);
            printf("error executing sort\n");
            exit(1);
      }
……………..
```