Assignment : 3

List of Topics: Interprocess communication, Exec Family

Exec() family of system calls:

Replaces the currently running process with a new process.

When we want a child process to perform something different than the parent process, we use any syscall from the exec() family.

- int execl(pathname, argo, ..., argn, 0);
  execl("/bin/ls", "ls", "-l", NULL);
- int execlp(cmdname, arg0, ..., argn, 0);
  execlp("ls", "ls", "-l", NULL);
- int execv(pathname, argv);
  char *argv[] = ("ls", "-l", NULL);
  execv("/bin/ls", argv);
- int execvp(cmdname, argv);
  char *argv[] = (argv[0], "-l", NULL);
  execvp("ls", argv);

Example: 1

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

int main(){
if(fork()==0)
{
        printf("child process\n");
        execlp("pwd","pwd",NULL);
}
else
{
        printf("parent process\n");
}
```

```
return 0;
}
```

Exercise:

1) Write a program, which runs two processes. The parent process prints all the destination IPs and the child process prints a list of all files and directories under the same directory in which C program is.
2) Write a program: The child process should create an empty file. After successful creation of an empty file, the parent process should append “This line is appended by the parent process” to the empty file using file I/O syscalls.

Interprocess Communication

PIPE:

A child and a parent process (or any two related processes) can communicate with each other using a pipe() system call.

Library: unistd.h

```
int pipe(int fd[2])
int fd[2];
int status
status=pipe(fd)
```

status=0 for success
-1 for error

fd[0] is open for reading
fd[1] is open for writing.

Basic Linux command to do communication between 2 process

```
#include <stdio.h>
#include <unistd.h>
#define MSGSIZE 16
char* msg1 = "hello, world #1";
```

```
int main()
{
        char inbuf[MSGSIZE];
        int p[2], i;

        // To check creation of pipe
        if (pipe(p) < 0)
                exit(1);

        /* write pipe */
        write(p[1], msg1, MSGSIZE);

        /* read pipe */
        read(p[0], inbuf, MSGSIZE);

        printf("% s\n", inbuf);

        return 0;
}
```

Exercise:

Data put into the pipe by one process can be read by another. E.g. one can create a pipe and write the stdout from one process to the pipe's write file descriptor and read from the pipe's read file descriptor to another program's stdin.

In the shell, this is typically seen in commands as ps -ef| grep <myuserid>

Write a c program (say myfilter.c) that creates a pipe between two processes given on the command line. If your program executable if a.out, and if you execute it with say the following two processes - **./a.out "ps -ef" "grep myuserid"**, you should get the expected output.

● myfilter has two commands to run (taken from command line argument) ● You will have

to parse the command line using tokenizer code

- program sets up a pipe using pipe() and then forks
- the parent and child close the pipe write and read descriptors respectively
- both processes run dup2() to duplicate stdin and stdout respectively to the pipe's descriptor
- Finally both parent and child run execvp() to load and run the commands

Submission:

1. Submit assignment with the report consists of an input file and output file with proper explanation of each output of all the **exercises** in pdf format.

2. Add all the outputs and a brief description of the commands used in the given demo scripts in the report

3. Submitted code in a report should be well commented.

Eg:

- ID_NO.pdf