گزارش تمرین ۱

برای این تمرین باید ۲ process فرزند و والد داشته باشیم که ابتدا فرزند دستور / ls را اجرا کند سپس خروجی را توسط pipe به والد بدهد و والد دستور wc -l را روی داده های روی pipe اجرا کند.

- برای راحتی کار command و argument دستورها define شده اند.

```
#define COMMAND1 "ls"
#define ARG1 "/"
#define COMMAND2 "wc"
#define ARG2 "-l"
```

- در اینجا توسط تابع pipe() یک لوله دو طرفه برای نوشتن روی آن و خواندن از آن ایجاد شده است.

```
int Pipe[2];
pipe(Pipe);
```

- این دستور باعث ایجاد child میشود و id آن چون بعدا برای چک استفاده میشود نگه داشته شده.

```
childPID = fork()
```

- در این قسمت وظایف child مشخص شده.دستور dup2 باعث میشود خروجی stdout روی pipe نوشته شود. در ادامه pipe هایی که لازم نداریم را میبندیم.برای اجرای دستور / ls از تابع execlp استفاده شده است.

```
else if( childPID == 0)
{
    dup2( Pipe[1], 1 );

    close( origStdout );
    close( origStdin );
    close( Pipe[0] );
    close( Pipe[1] );

    printf("\n");
    execlp( COMMAND1, COMMAND1, ARG1, NULL );
    perror("Execl error");
}
```

تفاوت exec ها:

L vs V:

L: individual parameters in the call (variable argument list): execl(), execle(), execlp(), and execlpe()

V: as an array of char* execv(), execve(), execvp(), and execvpe()

E: let you additionally pass an array of char* that are a set of strings added to the spawned processes environment before the exec'ed program launches.

P: use the environment path variable to search for the executable file named to execute. The versions without the 'p' require an absolute or relative file path to be prepended to the filename of the executable if it is not in the current working directory.

```
else
 {
 dup2( Pipe[0], 0 );
 dup2( origStdout, 1 );

 close( Pipe[0] );
 close( Pipe[1] );
 close( origStdout );
 close( origStdin );

 execlp( COMMAND2, COMMAND2,ARG2, NULL );

 perror( "EXEC Error");
 }
```

- در این قسمت وظایف parent مشخص شده.دستور dup2 باعث میشود ورودی stdin قسمت خواندن pipe باشد. در ادامه pipe هایی که لازم نداریم را میبندیم.برای اجرای دستور wc -l از تابع execlp استفاده شده است.