Practical – 10 Input and Output Redirection

If file is opened twice ,two file descriptor points to separate file table entries.

STDIN_FILENO -> standard input file
STDOUT_FILENO -> standard output file

Integer value	**Name**
0	Standard input (stdin)
1	Standard output (stdout)
2	Standard error (stderr)

Each process has its own file descriptor table
close(STDOUT_FILENO)
open("f1.txt",O_WRONLY.);
printf(""standard output redirection"); -> this will written to f1.txt instead of standard o/p file

dup() :
These system calls create a copy of the file descriptor .
it creates an alias for the provided file descriptor. dup always uses the smallest available file descriptor.
Thus, if dup() first thing in program, then it assign file descriptor 3 (dup uses 3 because 0, 1, and 2 are already taken by default).

Close() Must be used before dup() if stadarad input or output is redirected

#include <unistd.h>
Syntax : int dup(int filediscriptor

Program 1 :

```
//dup.c
#include<unistd.h>
#include<stdio.h>
#include<fcntl.h>
int main()
{
  int old_fd, new_fd ,redfd;
  char buf[10];
  old_fd=open("test.txt",O_RDWR);
  printf("File descriptor is %d\n",old_fd);
  new_fd=dup(old_fd);  //duplicate file discriptor
  printf("New file descriptor is %d\n",new_fd);
```

```
  read(old_fd,buf,10);  //old filediscriptor is also work with file
  printf("\n%s",buf);
 }
```

Program 2

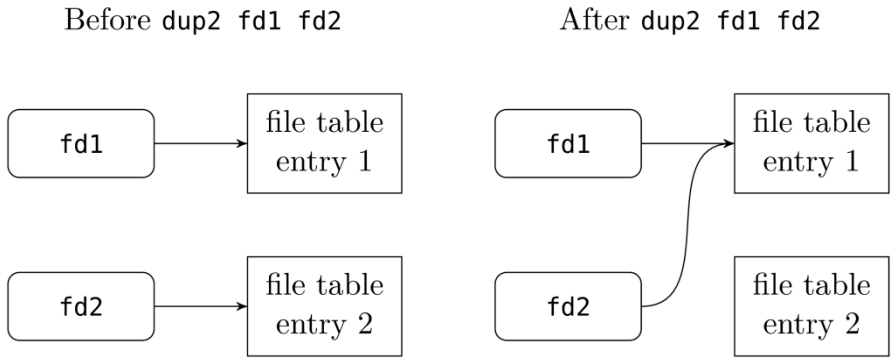
```
// output redirection using dup()

/Redirection using dup()
#include<sys/stat.h>
#include<fcntl.h>
#include<stdio.h>
#include <unistd.h>

int main()
{
        int fd1, fd2 , nfd1;

       char *argv[]={"cat",NULL};
      fd1 = open("one.txt",O_RDONLY);

      fd2 = open("out.txt",O_CREAT|O_WRONLY|O_TRUNC,0664);
        close(STDIN_FILENO);
        nfd1=dup(fd1);
       printf("fd1 = %d\n",nfd1);
        close(STDOUT_FILENO);
        dup(fd2);
       printf("fd1 = %d\n",fd2);
        execvp(argv[0],argv);

        printf("Command failed...\n");
        close(fd1);
        close(fd2);

}
```

Dup2():

The dup2() system call is similar to dup() but the basic difference between them is that instead of using the lowest-numbered unused file descriptor, it uses the descriptor number specified by the user

Syntax : int dup2(int fildes, int fildes2);

int fildes : The source file descriptor. This remains open after the call to dup2.
int fildes2 : The destination file descriptor. This file descriptor will point to the same file as filedes after this call returns.
return value: dup2 returns the value of the second parameter (fildes2) upon success. A negative return value means that an error occurred.



Program 3

```
//This program perform standard input and output redirection but using dup2()
#include<stdio.h>
#include<unistd.h>
#include<fcntl.h>



int main()
{
        int fd1,fd2,exitstatus;
       char *argv[]={"wc","-lc",NULL};

                fd1=open("one.txt",O_RDONLY);
                fd2=open("two.txt",O_WRONLY|O_TRUNC|O_CREAT,0644);
                dup2(fd1,0);
                dup2(fd2,1);
                execvp(argv[0],argv);

}
```

Program 4

```
//This program uses pipe for cat f1.txt | wc -l
//Child writes to the file and parent reads from the pipe

#include<stdio.h>
#include<unistd.h>
#include<stdlib.h>
int main()
{
        int fd[2],n,in,out;

        pipe(fd);

        switch(fork())
        {
                case -1:
                        printf("Fork error\n");
                        exit(1);
                case 0:
                        close(fd[0]);
                        in=dup2(fd[1],STDOUT_FILENO);
                        execlp("cat","cat","one.txt",NULL);
                      close(fd[1]);
                        break;
                default:
                        close(fd[1]);
                        out=dup2(fd[0],STDIN_FILENO);
                        execlp("wc","wc","-c","-l",NULL);
                        close(fd[0]);
        }

        close(in);
        close(out);

}
```

Exercise

1. Write a program which will work like cat f1.txt | head -2

Hint : use pipe and redirection