

DUP2 System Call – Duplication of File Descriptors

The **dup2()** system function is used to create a copy of an existing file descriptor

Stdin - standard input file descriptor - **scanf()**, **getc()** etc functions uses **stdin** file descriptor - also represented by the number **0**.

stdout: standard output file descriptor - **printf()** function uses **stdout** to print - also represented by the number **1**.

stderr: standard error file descriptor. also represented by the number **2**.

int dup2(int old_file_descriptor, int new_file_descriptor);

dup() system call creates a copy of a file descriptor. It uses the lowest-numbered unused descriptor for the new descriptor.



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

```
int main(void) {
    int number1, number2, sum;

    int input_fds = open("./input.txt", O_RDONLY);
    if (dup2(input_fds, STDIN_FILENO) < 0) {
        printf("Unable to duplicate file descriptor.");
        exit(EXIT_FAILURE);
    }

    scanf("%d %d", &number1, &number2);

    sum = number1 + number2;

    printf("%d + %d = %d\n", number1, number2, sum);

    return EXIT_SUCCESS;}
}
```



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

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

```
#include <sys/types.h>  #include <sys/stat.h>  #include <fcntl.h>
```

```
int main(void) { int number1, number2, sum;
```

```
    int input_fds = open("./input.txt", O_RDONLY);
```

```
    int output_fds = open("./output.txt", O_WRONLY );
```

```
    dup2(input_fds, STDIN_FILENO); //0
```

```
    dup2(output_fds, STDOUT_FILENO); //1
```

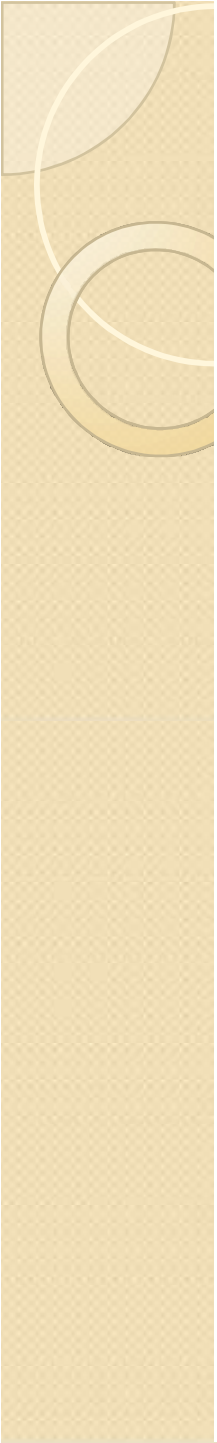
```
    scanf("%d %d", &number1, &number2);
```

```
    sum = number1 + number2;
```

```
    printf("%d + %d = %d\n", number1, number2, sum);
```

```
    return EXIT_SUCCESS;
```

```
}
```



```
int main()
{ // open() returns a file descriptor file_desc to a // the file
  "dup.txt" here"

  int file_desc = open("dup.txt", O_WRONLY | O_APPEND);

  if(file_desc < 0)      printf("Error opening the file\n");

      int copy_desc = dup(file_desc);

  write(copy_desc, "This will be output to the file named dup.txt\n",
    46);

  write(file_desc, "This will also be output to the file named dup.txt\n",
    51); return 0;

}
```

LS output in ALL CAPS using PIPES

```
#include<stdio.h>

#include<unistd.h>

#define MAX 512

int main(int argc, char* argv[])

{ int fd[2]; char buf[MAX]; int nb, i;

if (pipe(fd) == -1) { perror("Creating pipe"); exit(1); }

switch(fork())

{ case -1:

perror("Creating a process ");

exit(1);
```



case 0:

```
dup2(fd[1], 1);
```

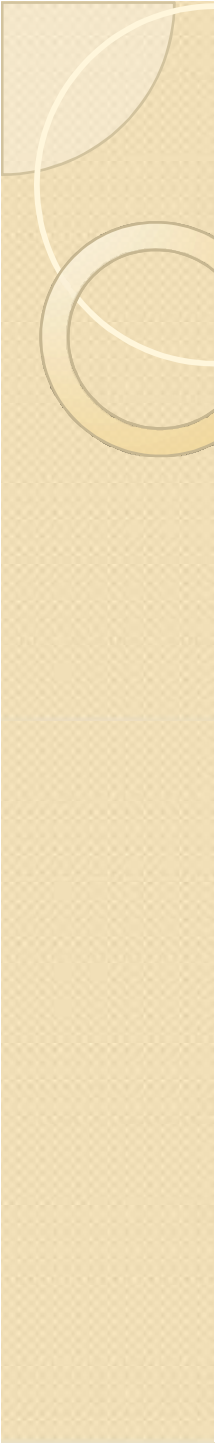
```
execvp("ls", argv); // output of ls command written on  
fd[1] courtesy the dup2 call.
```

```
perror("program ls");
```

```
exit(1);
```

default:

```
close(fd[1]);
```



```
while ((nb=read(fd[0], buf, MAX)) > 0)
{
for(i=0; i<nb; i++)
buf[i] = toupper(buf[i]);
printf("Test %s \n",buf);
if (write(1, buf, nb) == -1)
{ perror ("Writting to stdout");  exit(1); }
} // end of while
if (nb == -1){ perror("Reading from pipe");
exit(1); }
} // end of switch
} // main end
```

An alternate way of upper case output of LS Command

```
int main () {  
    char *cmd1[] = { "/bin/lS", "-al", "/", 0 };  
    char *cmd2[] = { "/usr/bin/tr", "a-z", "A-Z", 0 };  
  
    int pid; int pfd[2];  
  
    pipe (pfd); // other validation code u can repeat from earlier  
                discussion  
  
    switch (pid = fork())  
    { case 0:    /* child */  
  
        dup2(pfd[0], 0);  
  
        close(pfd[1]); /* the child does not need this end of the pipe */  
        execvp(cmd2[0], cmd2); // executes tr command on output of  
                                ls stored at the respective pipe end  
  
        perror(cmd2[0]);  
    }
```


An alternate way of upper case output of LS Command

```
default:      /* parent */
```

```
dup2(pfd[1], 1);
```

```
close(pfd[0]);
```

```
/* the parent does not need this end of the pipe */
```

```
execvp(cmdl[0], cmdl);
```

```
perror(cmdl[0]);
```

```
// Is executed and written to the pipe end of 1
```

```
case -l: perror("fork");
```

```
exit(1);
```

```
}
```

```
}
```