CSCI 3150 Tutorial

Assignment One - Part III

HUANG Qun

SHB 120

qhuang@cse.cuhk.edu.hk

04/10/2012

Outline

- Redirection
- Pipes
- Signals

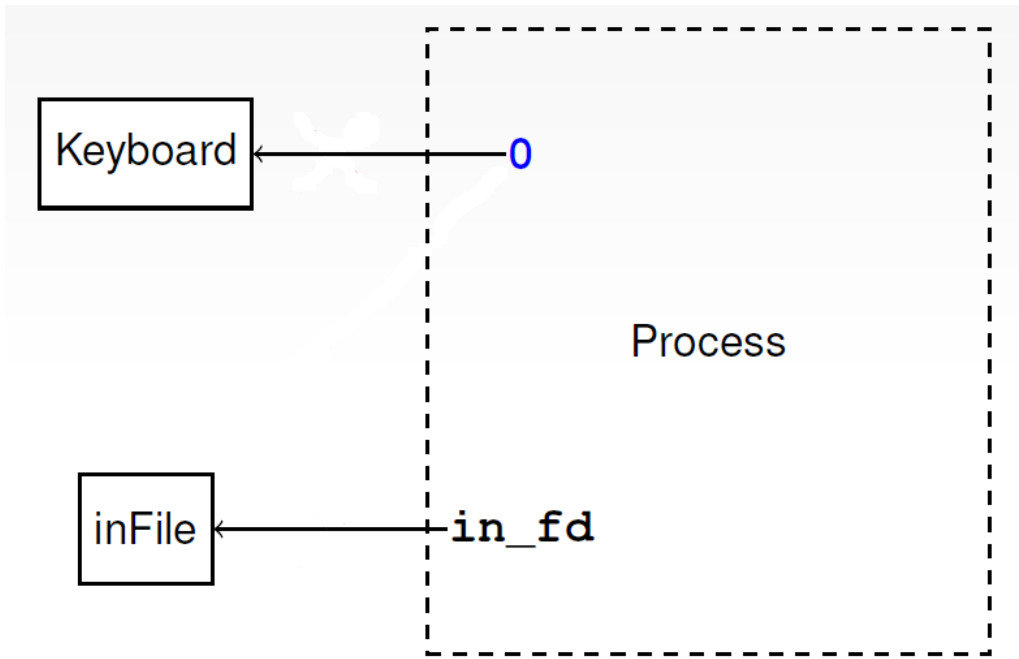
Redirection

STDIN, STDOUT, STDERR and File Descriptor

- As you may have already known, a process has three standard streams ready for read/write.
- These streams can be referenced by integers called file descriptors. The following names are defined in unistd.h.
  - STDIN_FILENO = 0 Standard input, default input file, usually associated with the keyboard
  - STDOUT_FILENO = 1 Standard output, default output file, usually print on the screen
  - STDERR_FILENO = 2 Standard error, responsible for showing error, usually print on the screen
- We will only redirect STDIN and STDOUT in our assignment.

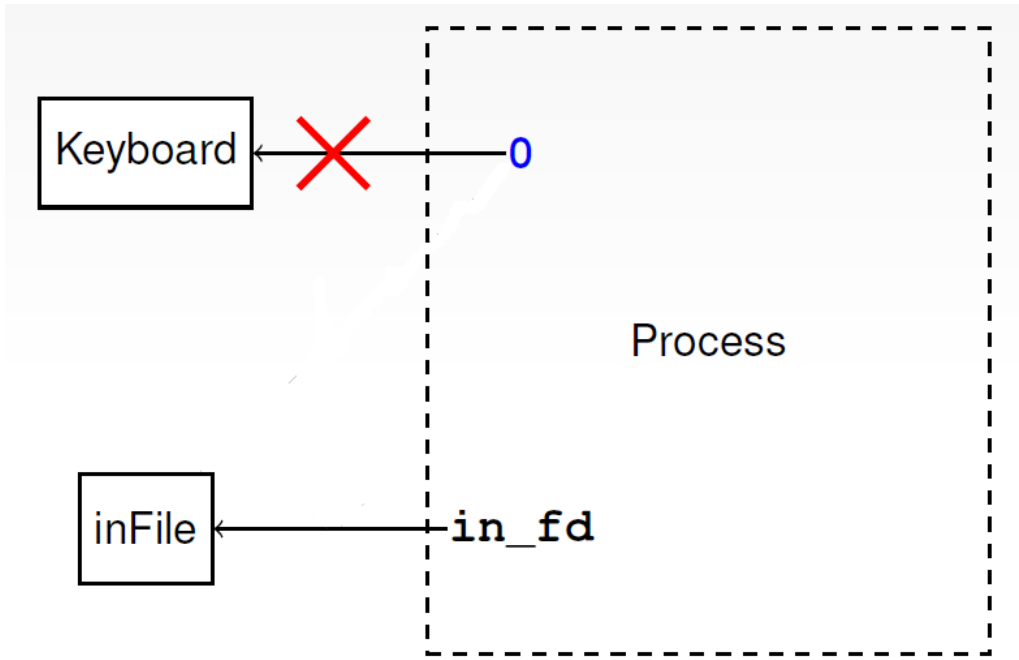
What does Input Redirection Mean?

- When a process reads from stdin, it reads from whatever that is pointed to by file descriptor STDIN_FILENO



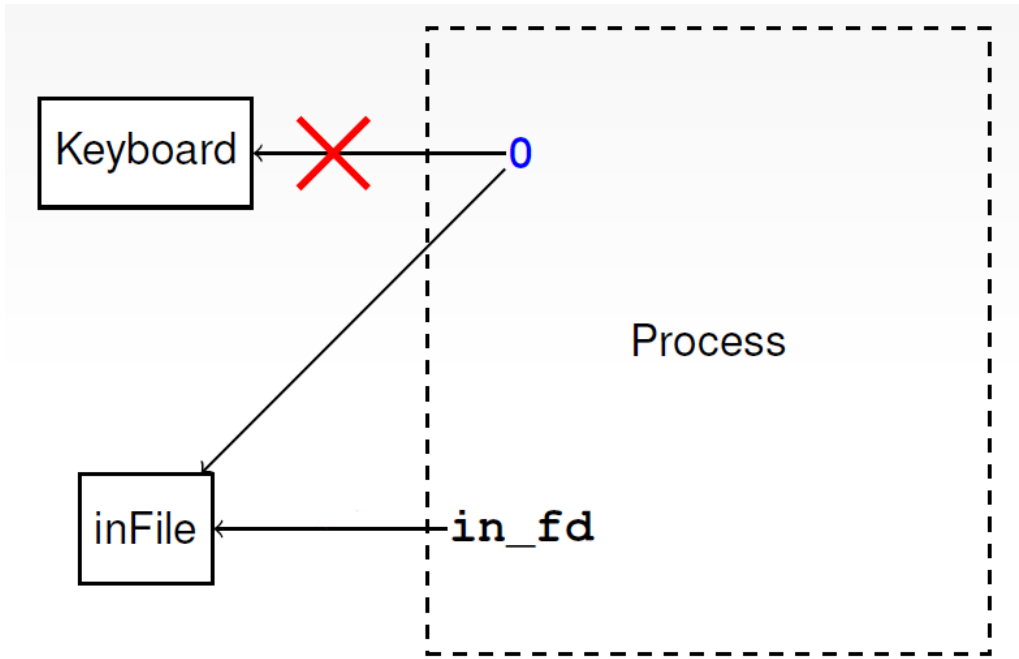
open the input file pointed by *in_fd*,
but the STDIN_FILENO still points to keyboard

What does Input Redirection Mean?



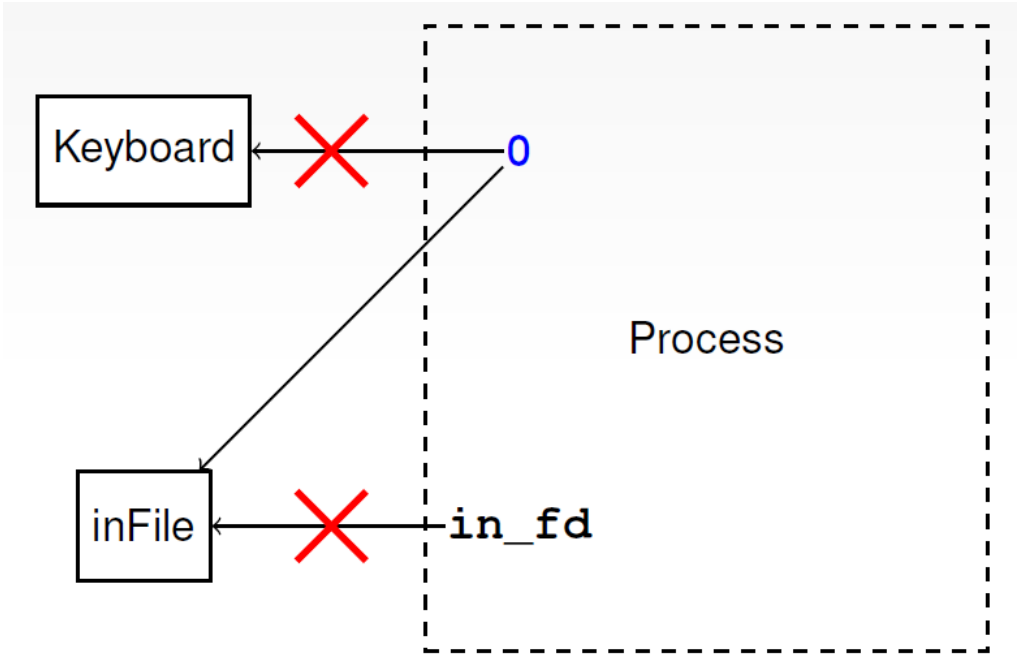
break the connection between STDIN_FILENO and keyboard

What does Input Redirection Mean?



let STDIN_FILENO point to the input file

What does Input Redirection Mean?



close *in_fd*, which breaks the connection between *in_fd* and the input file

Sample Code of Input Redirection

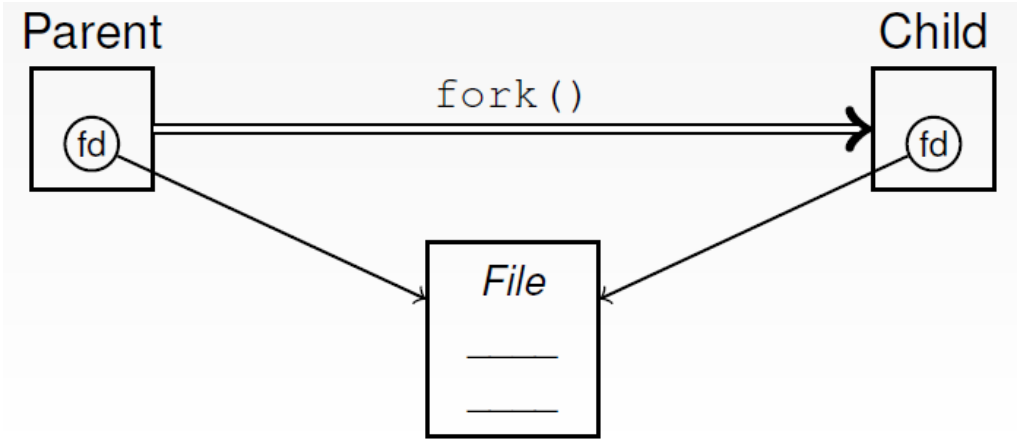
```
// open - open a file or device
int open(const char *pathname, int flags);

// dup2 - duplicate a file descriptor
int dup2(int oldfd, int newfd);
```

Example

```
in_fd = open(inFilename, O_RDONLY);
if (in_fd == -1) {/* error */}
if (dup2(in_fd, 0) == -1) {/* error */}
if (close(in_fd) == -1) {/* error */}
    /* our input file is associated with 0 already,
    * so we should dispose of the old extra reference
    * if it is not needed anymore */
```

Redirection

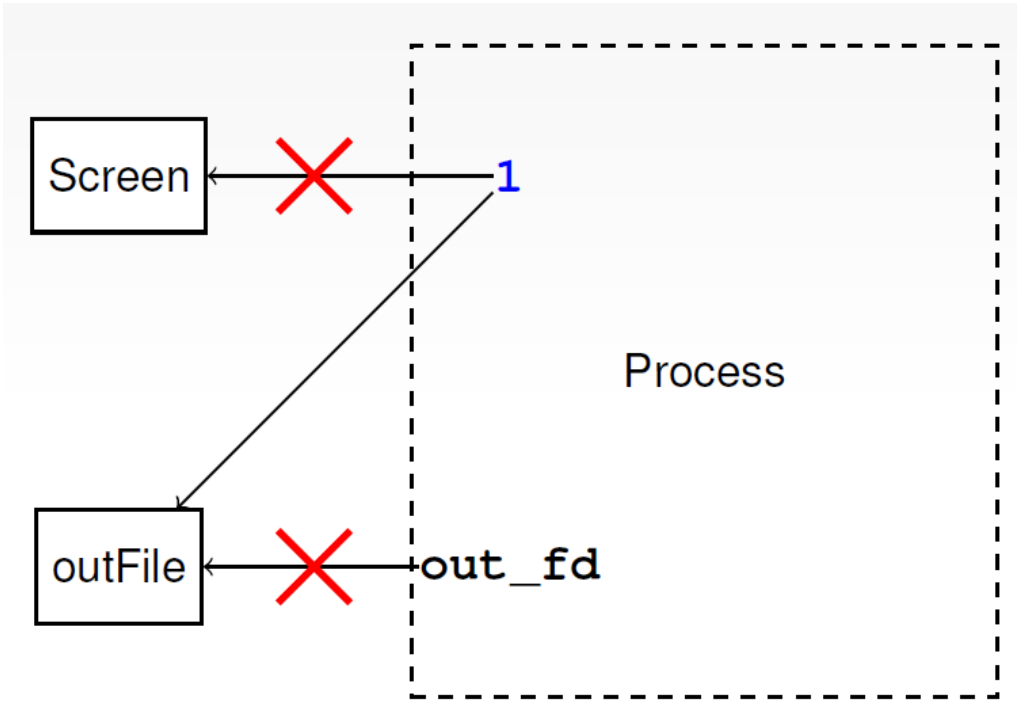
When to Redirect?



- **Important**: A child has same handles to files (known as file descriptors) as parent after *fork()*?
- On one hand, after *exec*()* the process is out of our control
- On other hand, we want only redirect input for the child
- So, handle the redirection between *fork()* and *exec*()* (in the child) [1]

[1] You can try handle the redirection before *fork* and see what will happen:)

Output Redirection



- When a process writes to stdout, it writes to whatever that is pointed to by file descriptor STDOUT_FILENO

Redirection

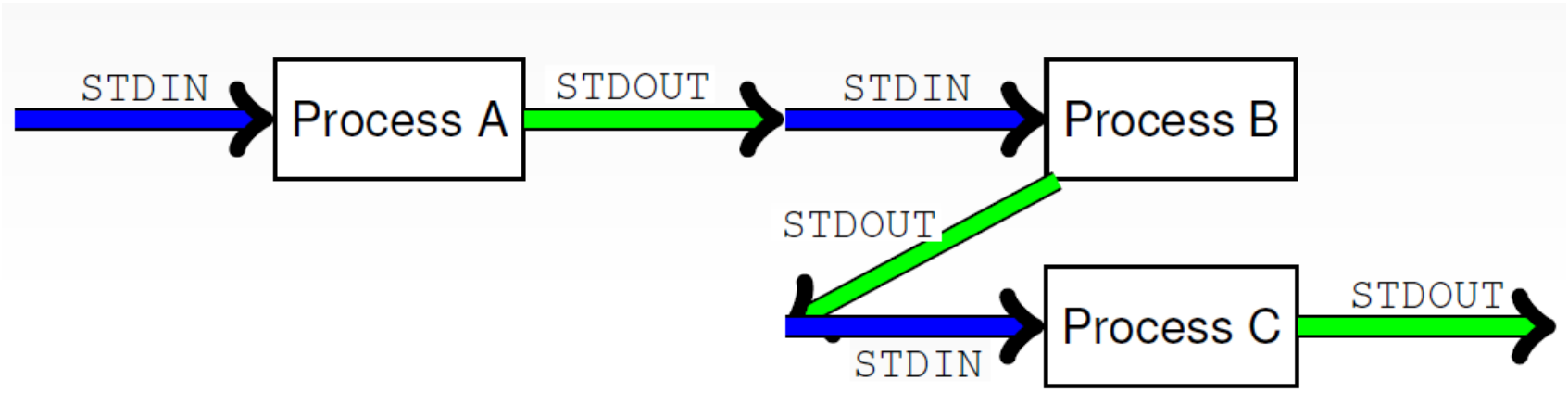
Compared to Input Redirection

- Similar to input redirection from file
- However, we need to do a little extra work when opening files
  - file opened for writing (O_WRONLY)
  - non-existent file has to be created (O_CREAT)
  - 2 different modes: append or truncate (O_APPEND or O_TRUNC)
  - permissions have to be set when creating files

Sample Code of Output Redirection

```
/* outMode = 0 for truncate, 1 for append */
if (outMode)
    out_fd = open(outFilename, O_WRONLY | O_CREAT
        | O_APPEND, S_IRWXU);
else
    out_fd = open(outFilename, O_WRONLY | O_CREAT
        | O_TRUNC, S_IRWXU);
if (out_fd == -1) {/* error */}
if (dup2(out_fd, 1) == -1) {/* error */}
if (close(out_fd) == -1) {/* error */}
```

Flow of Pipes



- Multiple Child process! (call *fork()* more than once)
- Both input and output redirections
- One writes to a buffer, another reads from it

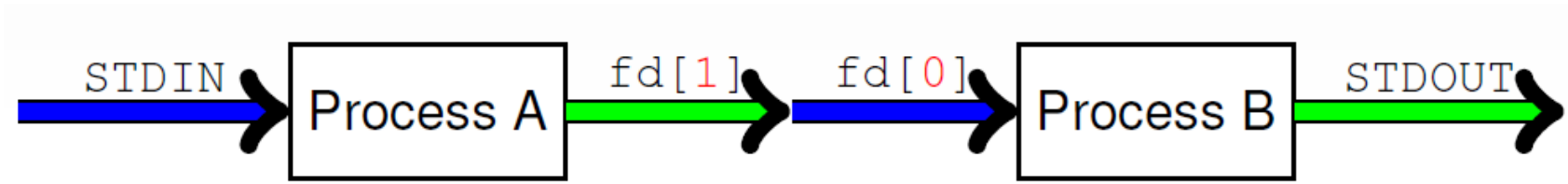
Sample Code of Pipes

```
// pipe - create pipe
int pipe(int filedes[2]);

// file descriptors for use in pipe
int fd[2];
if (pipe(fd) == -1) {/* error handling */}
// do something else

// for the first child process
if (dup2(fd[1], 1) == -1 ) {/* error handling */}
// do something else

// for the second child process
if (dup2(fd[0], 0) == -1 ) {/* error handling */}
// do something else
```

Example: *ls* | *cat*

```
int fd[2];
if (pipe(fd)== -1) {/* error handling */}
if ((pid1=fork())) {
    if (pid1 == -1) {/* error handling */}
    if ((pid2 = fork())) {
        // close fd[0] and fd[1]
        if (pid2 == -1) {/* error handling */}
        // call waitpid twice to wait for pid1 and pid2
    }
    else { // child 2
        if (dup2(fd[0], 0) == -1 ) {/* error handling */}
        // close fd[0] and fd[1]
        // execute "cat"
    }
}
else { // child 1
    if (dup2(fd[1], 1) == -1 ) {/* error handling */}
    // close fd[0] and fd[1]
    // execute "ls"
}
```

Pitfall



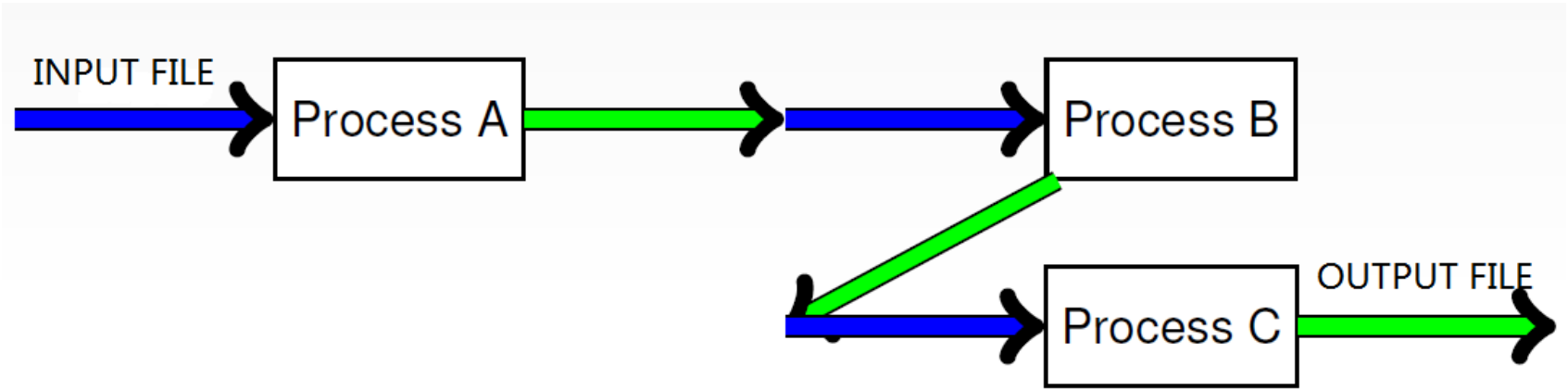
Note that fd[1] corresponds to output,
and fd[0] corresponds to input

Pipes

When to Call *pipe()*?

- We dont have much interprocess communications here
- If we *pipe()* in the child after *fork()*, other children have no way of using it
- So the only time to call *pipe()* is before *fork()*,
  i.e. *pipe()* in the parent first, then the child will have a copy of the (piped) file descriptors and call *dup2()*

Reconsider the Redirection



- When there are both pipe and I/O Redirection
- The first child is required to handle input redirection (if any)
- The last child is required to handle output redirection (if any)
- Remember that at most 2 *pipe()* (i.e. at most 3 child processes)in the assignment

Imporant: Remember to Clean Up!!!

```
// close - close a file descriptor
int close(int fd);
```

- Close all file descriptors that are not needed
- REMEMBER to close useless file descriptors in BOTH parent and child (if any) after *fork()*
- Or else you may see funny bugs . . .

An Example: The Parent forgets to close the pipe

Correct Result

```
qhuang@qhuang-vbox:~/TA/3150-2012-fall$ ./shell_correct
[3150 shell:/home/qhuang/TA/3150-2012-fall]$ ls
3150shell       exec.h    out.txt   shell_correct                tests
exec.c          main.c    parser.c  shell_no_close_fd_in_parent
exec_correct.c  Makefile  parser.h  shell_no_waitpid
[3150 shell:/home/qhuang/TA/3150-2012-fall]$ ls | grep main
main.c
[3150 shell:/home/qhuang/TA/3150-2012-fall]$
```

If the parent forget to close, the child cannot terminate here

```
qhuang@qhuang-vbox:~/TA/3150-2012-fall$ ./shell_no_close_fd_in_parent
[3150 shell:/home/qhuang/TA/3150-2012-fall]$ ls | grep main
main.c
```

If You use "*ps aux*" ...

```
qhuang    2563  0.0  0.0   1764   472 pts/1    S+   13:12   0:00 ./shell_no_clos
qhuang    2565  0.0  0.0   3324   804 pts/1    S+   13:12   0:00 grep main
```

How to Set up Signals

```
// signal - installs new signal handler for a signal
signal(int signum, sighandler_t handler);
```

Useful values for handler

- SIG_IGN Ignore the signal
- SIG_DFL Handle signal by default

Example

```
signal(SIGINT, SIG_IGN);
    // Current process now ignores Ctrl-C
signal(SIGINT, SIG_DFL);
    // Ctrl-C handling is restored
```

Restoring the Signals at Child Processes

- 6 signal handling routines are consider
- Only the parent process changes
- So, you must restore all the changes in the child processes