

CSI 402 – Systems Programming – Handout 15.1

An Example for the pipe System Call

Note: This handout shows how you can use the Unix system call `pipe` to set up a pipe between a parent process and a child process. The pipe that is set up corresponds to executing the following Unix command:

```
grep this file.txt | wc -l
```

In the program shown below, the parent process sets up the pipe and runs the `wc -l` command. The child process runs the `grep this file.txt` command.

The program shows how the child process redirects its `stdout` to the pipe and the parent process redirects its `stdin` to the pipe so that the output of the child process becomes the input of the parent process.

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

int main(void) {
    int pfd[2];          /* Pipe's file descriptors. */
    pid_t child_pid;    /* Child's process id.      */

    /* Set up pipe. */
    if (pipe(pfd) == -1) {
        fprintf(stderr, "Call to pipe failed.\n"); exit(1);
    }

    /* Fork a child process and let it execute the grep command. */
    if ((child_pid = fork()) == 0) {

        /* The child process. First, redirect its stdout to pipe. */
        if (dup2(pfd[1], STDOUT_FILENO) == -1) {
            fprintf(stderr, "dup2 in child failed.\n"); exit(1);
        }

        /* Now that redirection of stdout to pipe has been accomplished, */
        /* remove the file descriptors for the pipe from the child's      */
        /* file descriptor table.                                          */

        if (close(pfd[0]) == -1) { /* Failed to close read end of pipe. */
            fprintf(stderr, "Couldn't close read end of pipe in child\n");
            exit(1);
        }
        if (close(pfd[1]) == -1) { /* Failed to close write end of pipe. */
            fprintf(stderr, "Couldn't close write end of pipe in child\n");
            exit(1);
        }
    }
}
```

(over)

```

    /* Exec the grep command. */
    execlp("grep", "grep", "this", "file.txt", NULL);

    /* If the following statement is reached, execlp must have failed. */
    fprintf(stderr, "Call to execlp in child failed.\n"); exit(1);
} /* End of child. */

else {
    /* The parent process. Make sure that fork worked. */
    if (child_pid == (pid_t) -1) {
        fprintf(stderr, "The call to fork failed.\n"); exit(1);
    }

    /* Fork was successful. Redirect stdin to pipe and exec the wc command. */
    if (dup2(pfd[0], STDIN_FILENO) == -1) {
        fprintf(stderr, "dup2 in parent failed.\n"); exit(1);
    }

    /* Now that redirection of stdin to pipe has been accomplished, */
    /* remove the file descriptors for the pipe from the parent's */
    /* file descriptor table. */

    if (close(pfd[0]) == -1) { /* Failed to close read end of pipe. */
        fprintf(stderr, "Couldn't close read end of pipe in parent\n");
        exit(1);
    }
    if (close(pfd[1]) == -1) { /* Failed to close write end of pipe. */
        fprintf(stderr, "Couldn't close write end of pipe in parent\n");
        exit(1);
    }

    /* Exec the wc command. */
    execlp("wc", "wc", "-l", NULL);

    /* If the following statement is reached, execlp must have failed. */
    fprintf(stderr, "execlp in parent failed.\n");
    exit(1);
} /* End of parent. */
return 0;
} /* End of main. */

```