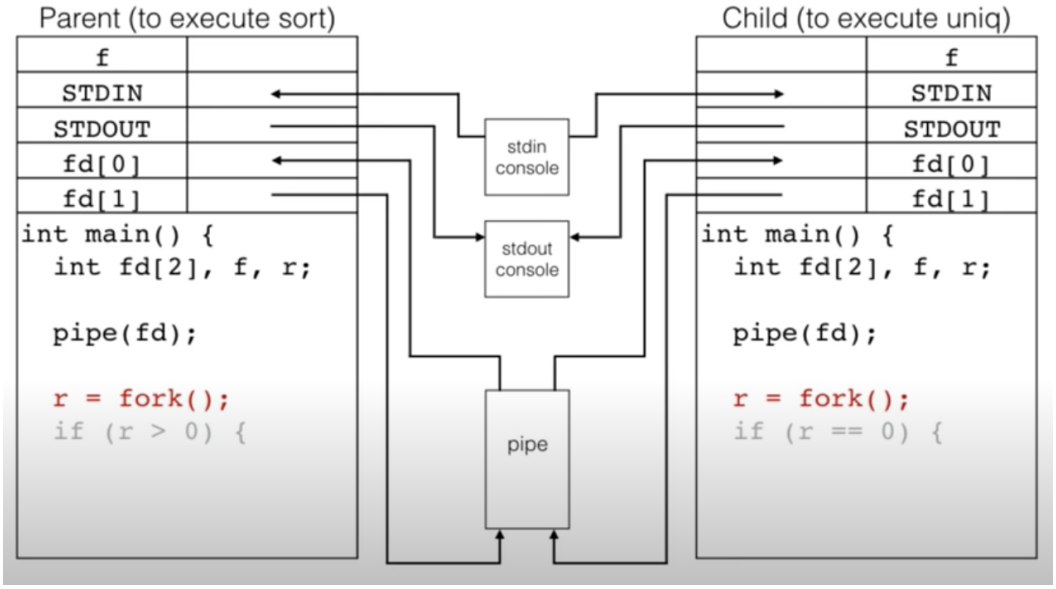
CSC209 Week 11 Notes

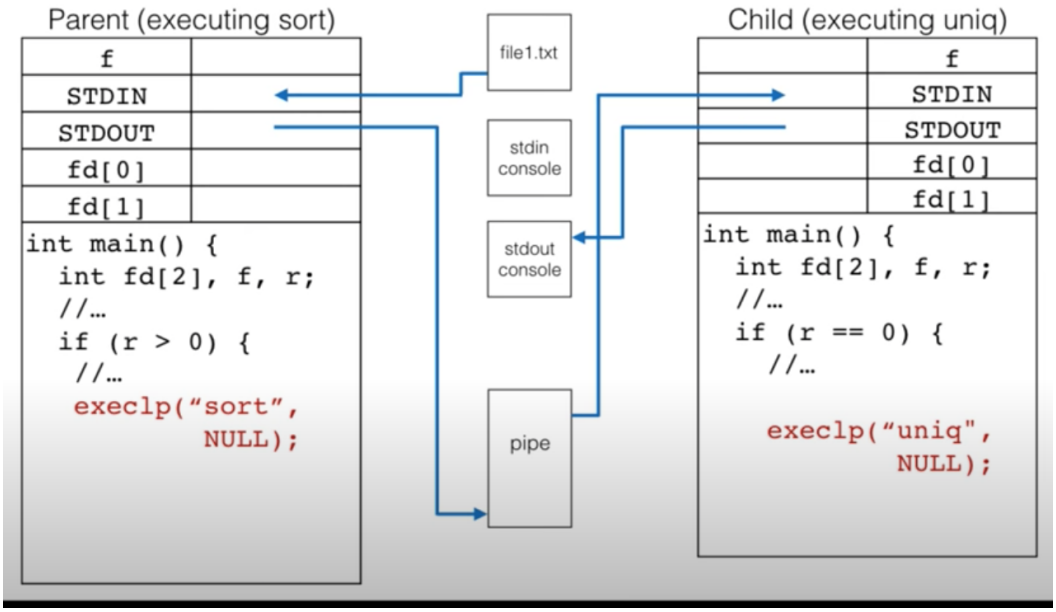
Hyungmo Gu

May 17, 2020

Processes 7 of 8

- Implementing the Shell Pipe Operator
  - piping
    - purpose is to send output from one end to input of anohter
    - done by using *dup2* and *pipe*
    - fd[0] and fd[1] must be closed after *dup2*
  - dup2
    - sets up redirection from fildes to fildes2
    - **Syntax:** int dup2(int fildes, int fildes2)
      - **fildes:** The source file descriptor
      - **fildes2:** The destination file descriptor
  - Example
    - Executing *sort* in parent and passing it to child for *uniq*
      - This is the same as 'sort <file1 | uniq'






```c
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include <sys/wait.h>dd

// equivalent to sort < file1 | uniq
void sort_by_parent(int *fd);
void uniq_by_child(int *fd);

int main() {
    int fd[2], r;

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

    r = fork();

    if (r < 0) {
        perror("fork");
        exit(1);
    }

    if (r > 0){
        sort_by_parent(fd);
    } else {
        uniq_by_child(fd);
    }
}

```

```
35  void sort_by_parent(int *fd) {
36      int filedes = open("file1.txt", O_RDONLY);
37
38      // reconfigure so all input from file1 are redirected to
    stdin
39      if (dup2(filedes, fileno(stdin)) == -1) {
40          perror("dup2.1");
41          exit(1);
42      }
43
44      // reconfigure so all output from stdout is redirected to
    write part of pipe
45      // this is to sent to uniq
46      if (dup2(fd[1], fileno(stdout)) == -1) {
47          perror("dup2.2");
48          exit(1);
49      }
50
51      // close read part of pipe
52      if (close(fd[0]) == -1) {
53          perror("close1");
54      }
55
56      // close write since it's redirected to stdout
57      if (close(fd[1]) == -1) {
58          perror("close2");
59      }
60
61      // close file since it won't be used directly
62      if (close(filedes) == -1) {
63          perror("close3");
64      }
65
66      // executes terminal's sort
67      execl("/usr/bin/sort", "sort", (char *) 0);
68  }
69
70  void uniq_by_child(int *fd) {
71      // reconfigure stdin so that it reads from pipe
72      if (dup2(fd[0], fileno(stdin))) {
73          perror("dup2");
74          exit(1);
75      }
76      // close the write pipe (see diagram)
77      if (close(fd[1]) == -1) {
78          perror("close");
79      }
80
81      // close the read pip since it will be read from stdin of
    pipe
82      if (close(fd[0]) == -1) {
83          perror("close");
84      }
85
```

```
86          execl("/usr/bin/uniq", "uniq", (char*) 0); // <- execute
    file from file path, and return 0 if successful
87          fprintf(stderr, "ERROR: exec should not return\n"); // <-
    run if uniq not run
88      }
89
```

Listing 1: pipe_example_1.c

```
1     >>> gcc -Wall pipe_example_1.c
2     >>> ./a.out
3     Fail T1
4     Fail T2
5     Fail T5
6     Pass
7
```

Shell Programming 6 of 6

- Shell operators Continued
  - **;:** Allows the program to run sequentially

```
1           prog1; prog2 #<- prog2 runs after prog1 ends
2
```

  - **&:** Allows the program to run simultaneously

```
1           prog1 & prog2; #<- prog1 and prog2 runs in parallel
2           prog1 & # <- prog1 runs in background, and cursor is
    returned immediately
3
```

  - **$!:** Gets the PID of latest program that's running in the background process
    - Don't use it more than a line! It's hard to read and error prone.

```
1           prog1 &
2           pid1=$!
3           prog2 &
4           pid2=$!
5
```

  - **#!:**
    - Don't use it more than a line! It's hard to read and error prone.
- Default executable
  - when file without extension is executed, shell recognizes it as a shell script, and bash program is run.

```
>>> ./hello
zsh: permission denied: ./hello
>>> chmod +x hello
>>> ./hello
./hello: line 1: hello: command not found

```