


Process & Pipe II

1. Process Pipe II

1.1 CSCI 330

CSCI 330

UNIX and Network Programming



Process & Pipe
Part 2

Video

1.2 Unit Overview

Unit Overview

- Process Management
 - create new process
 - change what a process is doing
- Pipe concept
 - Pipe for inter-process communication

1.3 System Calls

System Calls

- fork
 - create a new process
- wait
 - wait for a process to terminate
- exec
 - execute a program
- pipe
 - establish communication channel
- dup
 - duplicate file descriptor

1.4 System Call: fork example

System Call: fork example

```
forkLoopWait.cxx x
16 int main() {
17     pid_t pid;
18     // fork will make 2 processes
19     pid = fork();
20     if (pid == -1) { perror("fork"); exit(EXIT_FAILURE); }
21
22     if (pid == 0) {
23         // Child process: fork() returned 0
24         for (int j = 0; j < 5; j++) {
25             cout << "child: " << j << endl;
26         }
27     } else {
28         // Parent process: fork() returned a positive number
29         // wait for child to end
30         wait(NULL);
31         for (int i = 0; i < 5; i++) {
32             cout << "parent: " << i << endl;
33         }
34     }
35     cout << "end of process:" << getpid() << endl;
36     return 0;
37 }
38
```

1.5 System Call: exec

System Call: exec

- family of functions that replace current process image with a new process image
 - actual system call: **execve**
 - library functions
 - **execl, execlp, execl**
 - **execv, execvp**
- arguments specify new executable to run and its arguments and environment

1.6 C Library Function: *execlp*

C Library Function: *execlp*

```
int execlp(const char *cmd, const char *arg, ...)
```

- starts executable for command specified in **cmd**
- new executable runs in current process
- **cmd** executable is found via path
- arguments are specified as list, starting at argv[0],
terminated with (char *NULL)
- return -1 on error

1.7 Together: *fork* and *exec*

Together: *fork* and *exec*

- UNIX does not have a system call to spawn a new additional process with a new executable
- instead:
 - fork to duplicate current process
 - exec to morph child process into new executable

1.8 Together: fork and exec

Together: fork and exec

```
ForkRunCommand.cpp x
16 int main() {
17     string command, argument;
18     // prompt user for command to run
19     cout << "Enter command to run (with one argument): ";
20     cin >> command >> argument;
21
22     int pid, rs, status;
23     // fork will make 2 processes
24     pid = fork();
25     if (pid == -1) { perror("fork"); exit(EXIT_FAILURE); }
26
27     if (pid == 0) {
28         // Child process: exec to command with argument
29         cout << "before exec to: " << command << " " << argument << endl;
30
31         rs = execlp(command.c_str(), command.c_str(), argument.c_str(), (char*) NULL);
32
33         if (rs == -1) { perror("execlp"); exit(EXIT_FAILURE); }
34     } else {
35         // Parent process: wait for child to end
36         wait(&status);
37     }
38     cout << "end of process: " << getpid() << endl;
39     return 0;
40 }
41
42
```

1.9 UNIX Pipe

UNIX Pipe



- can create a software pipeline:
 - set of processes chained by their standard IO
- output of one process becomes input of second process

command line example:

ls | wc

implemented via **pipe** system call

1.10 System Call: pipe

System Call: pipe

```
int pipe(int pipefd[2])
```

- creates a channel to transport data
- has direction: one side to write, one side to read
 - available via 2 file descriptors `pipefd[2]`
 - read side `pipefd[0]`
 - write side `pipefd[1]`
- can be used synchronize producer and consumer of data

1.11 System Calls: pipe and fork

System Calls: pipe and fork

- Idea: read and write end of pipe in different processes
 - single process creates pipe
 - fork creates two processes
- parent process:
 - close read end of pipe
 - write to write end of pipe
- child process:
 - close write end of pipe
 - read from read end of pipe

1.12 System Call: pipe & fork example

System Call: pipe & fork example

```
pipeFork.cxx
16 int main() {
17     int pipefd[2], rs;
18     char buffer[256];
19
20     // create pipe
21     rs = pipe(pipefd);
22     if (rs == -1) { perror("pipe"); exit(EXIT_FAILURE); }
23     cout << "pipe created\n";
24
25     // fork into 2 processes
26     rs = fork();
27     if (rs == -1) { perror("fork"); exit(EXIT_FAILURE); }
28
29     if (rs == 0) { // child process
30         // close write end of pipe
31         close(pipefd[1]);
32         // read from read end of pipe
33         read(pipefd[0], buffer, sizeof(buffer));
34         cout << "pipe contained: " << buffer << endl;
35     } else { // parent process
36         // close read end of pipe
37         close(pipefd[0]);
38         // write to write end of pipe
39         write(pipefd[1], "Hello", 6);
40         wait(NULL);
41     }
42     return 0;
43 }
44
45
```

1.13 Question: how to implement “|”

Question: how to implement “|”

- shell allows pipe on command line
 - connects output of one command to input of second command

Example: **ls | wc**

- implemented with system calls:
 - fork, pipe, dup, exec

1.14 Command One: ls

Command One: ls

- shell creates pipe
- shell forks:
 - parent keeps going
 - child
 - closes standard output
 - dups pipefd[1]
 - execs to command one: ls

1.15 Command Two: wc

Command Two: wc

- shell forks again:
 - parent keeps going
 - child
 - closes standard input
 - dups pipefd[0]
 - execs to command two: wc
- parent (shell) waits for children to complete

1.16 Illustration: shell pipe implementation

Illustration: shell pipe implementation

- Idea:
 - create 2 child processes
 - 2 processes communicate via pipe
 - each process exec's into new executable
- Example: **ls | wc**
 - child process 1: runs **ls**
 - child process 2: runs **wc**

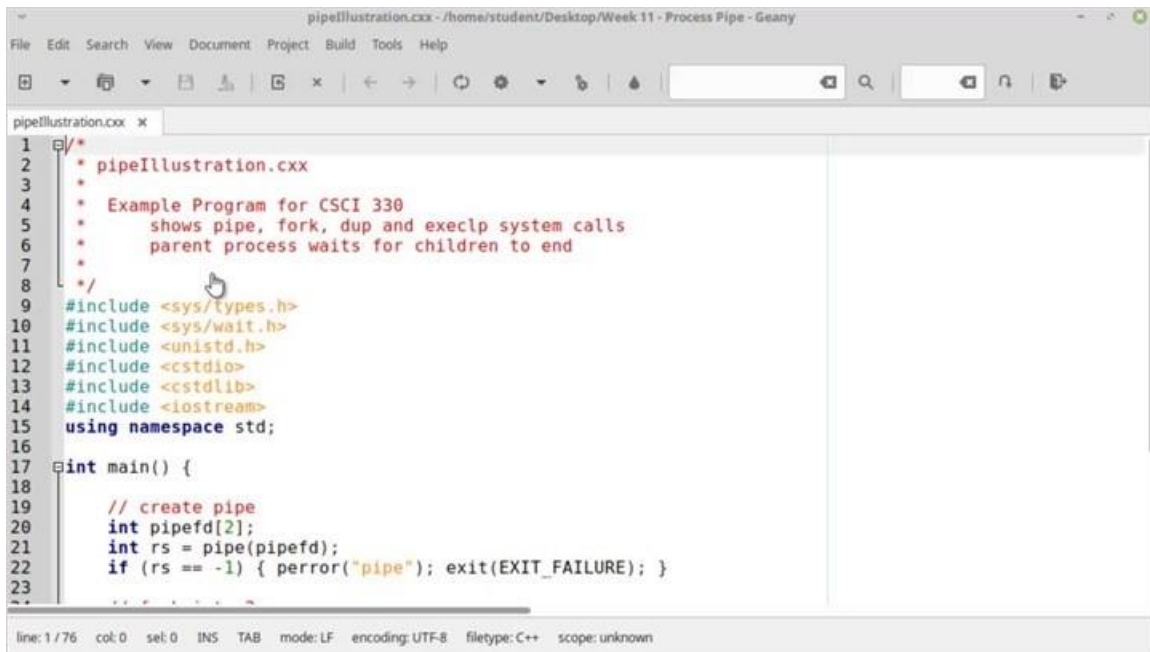
1.17 Illustration

Illustration

- create pipe
- run command 1 in child
 - with pipefd[1] as output
- run command 2 in child
 - with pipefd[0] as input
- wait for children

```
pipeImplementation.c
18 int main() {
19     // create pipe
20     int pipefd[2];
21     int rs = pipe(pipefd);
22     if (rs == -1) { perror("pipe"); exit(EXIT_FAILURE); }
23
24     // fork into 2 processes
25     rs = fork();
26     if (rs == -1) { perror("fork"); exit(EXIT_FAILURE); }
27     if (rs == 0) { // child process 1
28         // close read end of pipe, keep write end open
29         close(pipefd[0]);
30         // close standard output
31         close(STDOUT_FILENO);
32         // duplicate write end of pipe into standard output
33         dup(pipefd[1]);
34         // close write end of pipe
35         close(pipefd[1]);
36         // run first command
37         rs = execlp("ls", "ls", (char*)NULL);
38         if (rs == -1) { perror("execlp"); exit(EXIT_FAILURE); }
39     } else { // parent process
40         // close write end of pipe, keep read end open
41         close(pipefd[1]);
42         // fork into 2 processes
43         rs = fork();
44         if (rs == -1) { perror("fork"); exit(EXIT_FAILURE); }
45         if (rs == 0) { // child process 2
46             // close standard input
47             close(STDIN_FILENO);
48             // duplicate read end of pipe into standard input
49             dup(pipefd[0]);
50             // close read end of pipe
51             close(pipefd[0]);
52             // run second command
53             rs = execlp("wc", "wc", (char*)NULL);
54             if (rs == -1) { perror("execlp"); exit(EXIT_FAILURE); }
55         } else { // parent process
56             // close read end of pipe
57             close(pipefd[0]);
58             wait(NULL); // for first child
59             wait(NULL); // for second child
60             cout << "Parent done\n";
61         }
62     }
63     return 0;
64 }
```

1.18 Pipe Illustration



```
1  * pipeIllustration.cxx
2  *
3  * Example Program for CSCI 330
4  * shows pipe, fork, dup and execlp system calls
5  * parent process waits for children to end
6  *
7  */
8
9  #include <sys/types.h>
10 #include <sys/wait.h>
11 #include <unistd.h>
12 #include <cstdio>
13 #include <cstdlib>
14 #include <iostream>
15 using namespace std;
16
17 int main() {
18     // create pipe
19     int pipefd[2];
20     int rs = pipe(pipefd);
21     if (rs == -1) { perror("pipe"); exit(EXIT_FAILURE); }
22 }
23
```

line: 1 / 76 col: 0 sel: 0 INS TAB mode: LF encoding: UTF-8 filetype: C++ scope: unknown

1.19 Summary

Summary

- Process Management
 - create new process
 - change what a process is doing
- Pipe concept
 - Pipe for inter-process communication