CS170 Operating Systems

Discussion Section Week 2

- My office hour:
    - Wed: 4pm to 6 pm
    - Fri: 2pm to 4 pm
    - Place: CSIL
    - I will sit in the first line, near the door.

- Started coding on first project?

Project1 - Shell

Important Milestones:

1. Implement basic shell without four signs
2. Implement ">" and "<"
3. Implement basic "|"
4. Handle multiple "|"
5. Implement "&" and handle CTRL-C and D
6. Handle all 4 signs mixed together

Project1 - Shell Grading

Important Milestones:

1. Implement basic shell without four signs-30%
2. Implement ">" and "<" - 15%
3. Implement basic "|" - 20%
4. Handle multiple "|" - 15%
5. Implement "&", handle CTRL-Cand D- 10%
6. Handle all 4 signs mixed together- 10%

Outline

- Implement basic shell without four signs
- Implement ">" and "<"
- Implement basic "|"
- Handle multiple "|"
- Implement "&" and handle CTRL-C and D
- Handle all 4 signs mixed together

Basic Shell

- Print a prompt

Basic Shell

- Print a prompt
- Get the command
    - Use fgets or gets
    - Syntax:

      gets(char *s)

      fgets(chat *s, int len, FILE *fp)
    - Use "stdin" instead of FILE *fp for fgets
    - Returns 0 on "EOF"
        - Can you handle Ctrl-D using this?
    - Which one should you prefer - fgets() or gets() ?

Basic Shell

- Print a prompt
- Get a command
- Parse the command
  - Scan each character using a loop to get the tokens

Basic Shell

- Print a prompt
- Get a command
- Parse the command

Execute the command

- Use fork()
- Execute the command in the child process
- Use execvp() to take advantage of path
  - execvp(const char *cmd,char *const argv[])
  - What are the first and last elements of argv?
- Parent waits for child to terminate unless "&" is used

Something like this..

```
pid_t pID = fork();
if(pID == 0){
      ret = execvp(command_name, command_args);
}
else if(pID < 0){
      printf("Failed to fork\n");
      exit(1);
}
else{
      waitpid(-1, &status, 0);
}
```

Outline

- Implement basic shell without four signs
- Implement ">" and "<"
- Implement basic "|"
- Handle multiple "|"
- Implement "&" and handle CTRL-C and D
- Handle all 4 signs mixed together

Input and Output Redirection

- "<" means input to the command is read from file instead of stdin
- ">" means output of the command is written to file instead of stdout
- You would want to use open(), dup2(), close()

File Descriptors

- What are file descriptors?

File Descriptors

- File can be accessed using File descriptor
- It is non negative integer for each file

- Reserved:
  - 0(stdin), 1(stdout), 2(stderr)

- Rest of the files use fd>2 when you use open()

Open()

- Open file:
- Defined in unistd.h
- Syntax
  - open("filename",options,permissions)
- Returns file descriptor.
  - If fd<0, indicates error opening file

Open()

Example:

```
   #include <fcntl.h>
   #include <unistd.h>
   int infilefd;
   infilefd=open("infile.txt",O_RDONLY, 0);
```

Open() useful options

- O_CREAT: create if not exists
- O_RDONLY: read only
- O_WRONLY: write only
- O_TRUNC: Truncate the contents of file before writing
- O_RDWR: read and write

- Specify multiple options using "|"(or)
  - outfilefd=open("file.txt", O_CREAT|O_WRONLY|O_TRUNC, 644);

close()

- int close(int fd)

Example:

```
close(outfilefd)
```

dup2()

```
#include <unistd.h>
int dup2(int filedes, int filedes2);
```

- dup2 closes the entry filedes2 of the file descriptor table, and then
- copies the pointer of entry filedes into filedes2.
- In other words it changes the pointer in: filedes2 to the pointer in filedes.

dup2(fd,fd2)

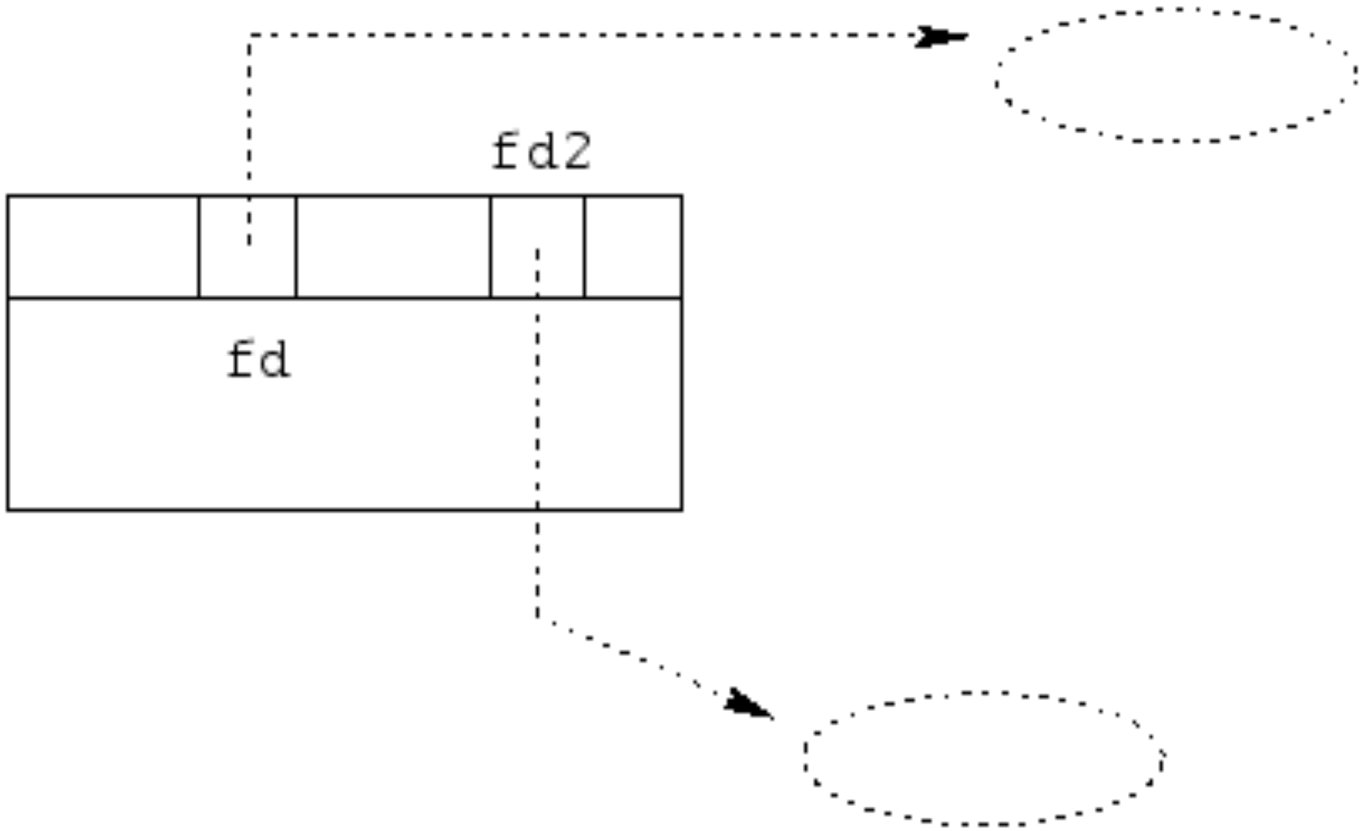


Figure 2: Before a call to `dup2(fd,fd2)`

dup2(fd,fd2)

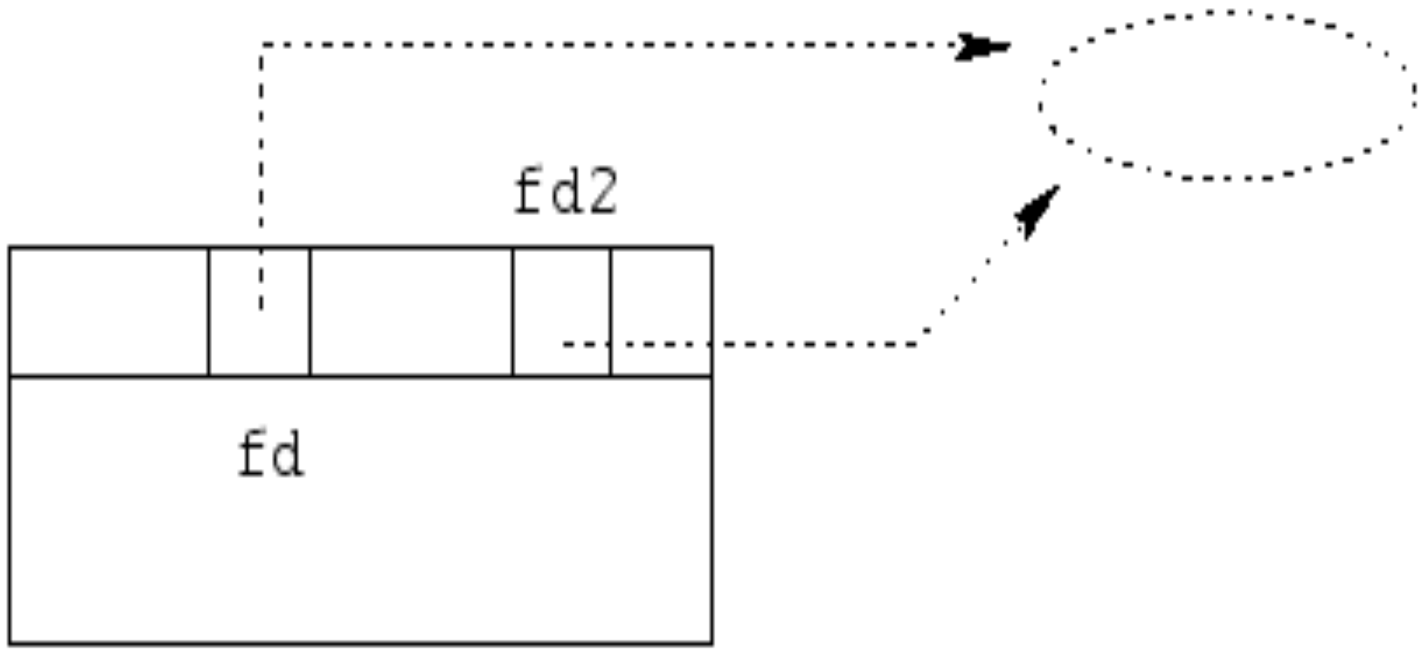


Figure 3: After a call to `dup2(fd,fd2)`

Implementing "<" and ">"

- Open the file
  - Eg. newfd=open(.....)
- Assign newfd to 0(if "<") or 1(if ">") using dup2
  - Eg. dup2(newfd,0)
- Execute the command
  - The command reads from 0 which is your file now
- Close your file
  - Eg. close(newfd)
- Restore original file descriptors

Implementing "<" and ">"

- Your shell must support:
  - cat file1
  - cat < file1
  - cat > file1
  - cat < file1 > file2
  - cat > file1 < file2

Outline

- Implement basic shell without four signs
- Implement ">" and "<"
- Implement basic "|"
- Handle multiple "|"
- Implement "&" and handle CTRL-C and D
- Handle all 4 signs mixed together

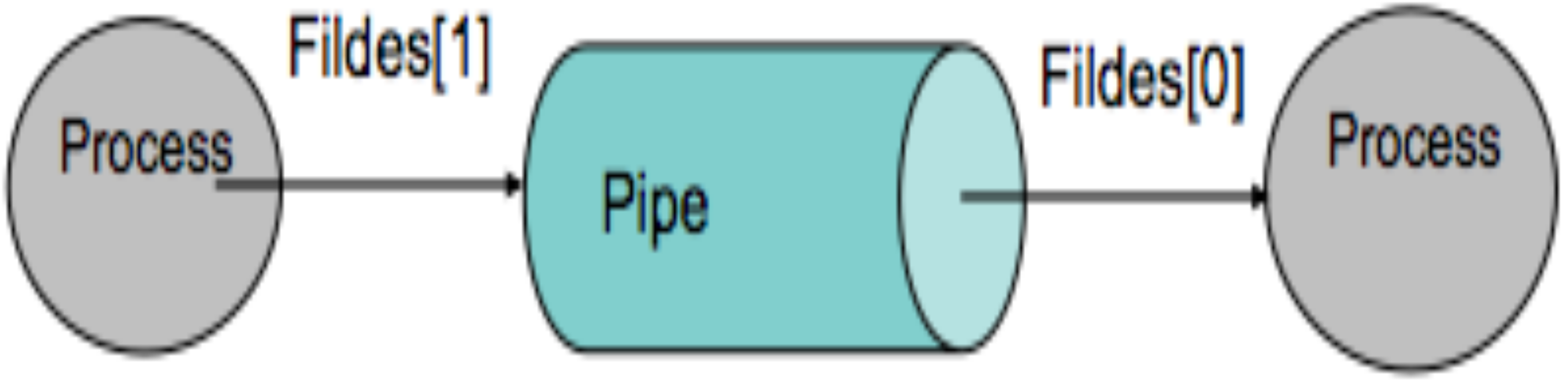
Pipes

- Pipes are interprocess communication buffers (of course they are files).

-
  ```
  #include <unistd.h>
  int pipe(int filedes[2]);
  ```
- On success :
  - fildes[0] – descriptor for read
  - fildes[1] – descriptor for write
  - return value 0
- On failure : return value -1

Pipes

```
int filedes[2];
pipe(filedes);
```



Simple Example (parent_proc | child_process)

```
int file_pipes[2];
Char some_data[] = “123”;
pipe(file_pipes) ;
fork_result = fork();
…
if (fork_result == 0) { // Child
        close(file_pipes[1]);
        data = read(file_pipes[0], buffer, BUFSIZ);
        printf("Read %d bytes: %s\n", data, buffer);
        exit(EXIT_SUCCESS);
 }
else { //Parent
        close(files_pipes[0]);
        data= write(file_pipes[1],some_data,strlen(some_data)+1);
        printf("Wrote %d bytes\n", data);
 }
```

Implementing Pipe: ls -l | sort -n -k 5

```
int fd[2];
pid_t childpid;
pipe(fd);
if ((childpid = fork()) == 0) {  /* ls is the child */
   dup2(fd[1], STDOUT_FILENO);
   close(fd[0]);
   close(fd[1]);
   execute "ls -l"
   ....
} else {  /* sort is the parent */
   dup2(fd[0], STDIN_FILENO);
   close(fd[0]);
   close(fd[1]);
   execute "sort ...."
   ....
}
```

Outline

- Implement basic shell without four signs
- Implement ">" and "<"
- Implement basic "|"
- Handle multiple "|"
- Implement "&" and handle CTRL-C and D
- Handle all 4 signs mixed together

Handle multiple "|"

- **ls** -al | grep '.c' | cut -c30-| sort -n
- Parse the first command (everything before the first '|').
  - fork, and child will become *ls -al*

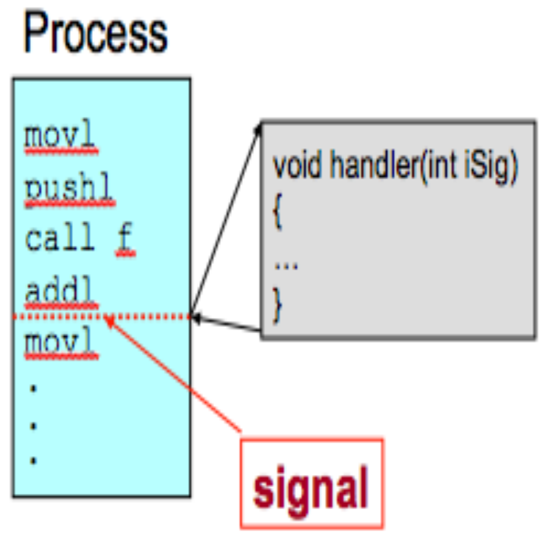
    Eg. command becomes ls -al | something
  - Have child create *pipe()*, and *fork()*.
    - child execs *ls*.
    - grandchild processes *something* (in the same way).
- Think of a data structure!!

Outline

- Implement basic shell without four signs
- Implement ">" and "<"
- Implement basic "|"
- Handle multiple "|"
- Implement "&" and handle CTRL-C and D
- Handle all 4 signs mixed together

Signals

- **OS** communicate to an **application process** using signals
- Signal: A notification of an event
    - Event gains attention by OS
    - OS stops application process immediately
    - Executes signal handler completely
    - Resume process from where it left



Example

- User types Ctrl-c
  - Event gains attention of OS
  - OS stops the application process immediately, sending it a 2/SIGINT signal
  - Signal handler for 2/SIGINT signal executes to completion
  - Default signal handler for 2/SIGINT signal exits process

Signal Handling

- Every signal has a handler associated with it
  - Most default handler exit the process

- **sighandler_t signal(int iSig, sighandler_t pfHandler);**
  - Installs function **pfHandler** as the handler for signals of type **iSig**
  - **pfHandler** is a function pointer:
    typedef void (*sighandler_t)(int);

Signal Handling Example

```
#include <stdio.h>
#include <signal.h>
void myHandler(int iSig) {
   printf("In myHandler with argument %d\n", iSig);
}

//invokes myhandler whenever CTRL-C is pressed
signal(SIGINT,myhandler)
```

Signal Handling options

```
signal(SIGINT,SIG_IGN) //Ignore the signal

signal(SIGINT,SIG_DFL)
//Apply default signal handler(which is exit)
```

Implementing '&'

1. Notice that the '&' sign can only exist as the last token, otherwise it is misplaced

2. Parent does not wait for child to complete.

3. Child sends SIGCHLD on completion

Avoid "zombies"

- When process finish execution, it has exit status to report to parent.
- So process exist in process table but has completed execution=> zombie

- Parent receives SIGCHLD when child exits
- Parent calls wait() to collect child status and removes the process

Avoid "zombies"

- Wait options:
  - WCONTINUED: The waitpid() function shall report the status of any continued child process specified by pid whose status has not been reported.
    - It suspends execution of calling thread
  - WNOHANG: The waitpid() function shall not suspend execution of the calling thread if status is not immediately available for one of the child processes specified by pid.

Handling Ctrl + D

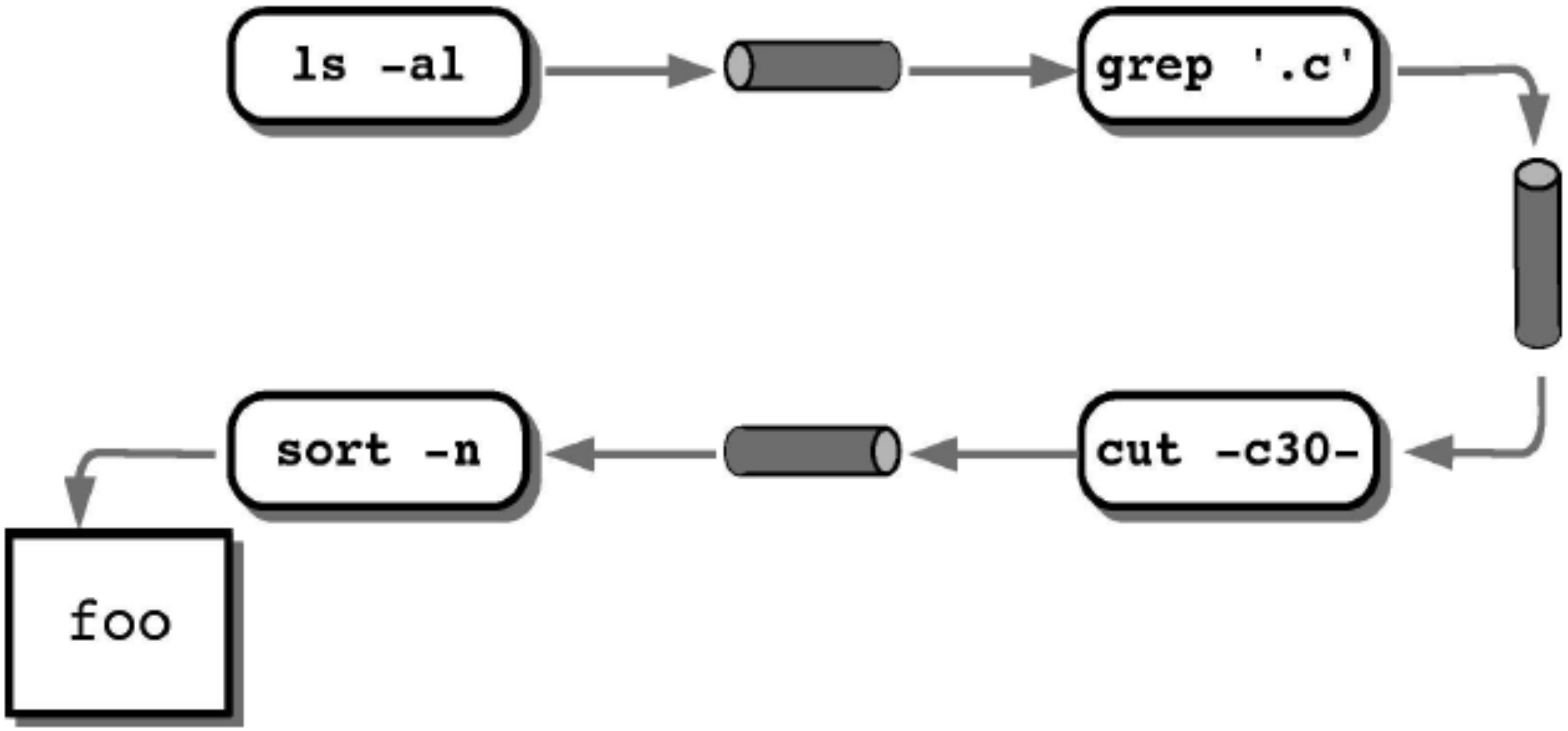
- Use fgets and read from stdin
  - char s[100]
  - fgets(s,len(s),stdin)
  - Ctrl-D is EOF, fgets returns 0 on EOF

Outline

- Implement basic shell without four signs
- Implement ">" and "<"
- Implement basic "|"
- Handle multiple "|"
- Implement "&" and handle CTRL-C and D
- Handle all 4 signs mixed together

Putting it together!

- **ls** -al | grep '.c' | cut -c30-| sort -n > foo &



Testing

1. Compare the output by executing on terminal.

2. I will update sample input and output files on my TA page for each milestone.
http://cs.ucsb.edu/~manish/Teaching

3. Grading would be done using secret test files and not the ones I post.

Interesting Question

```
int main()
{
     fd = open(dir);
     printf("fd =%d",fd);
     if(fd<2)
          perror("open");
}
```

I get fd = -2 and the perror("open") prints "Success"!!!

Give a plausible explanation for this behavior.

Questions or Comments?