UNIVERSITY *of* WEST FLORIDA

COP4634: Systems & Networks I

Zombies

Redirection

Programs

- Compiled, executable code
- Stored on disk
- **Passive** entity
- Doesn't **do** anything
- Read from disk
- Written to memory
- Execution begun

$\left.\begin{array}{l}\\ \\ \\ \end{array}\right\}$

Done by loader

no longer program

now a process

Preventing Zombies

- Parent must outlive children
- Two useful system calls:
  - `wait()` – blocks until <u>any</u> child terminates
  - `waitpid()` – blocks until <u>a specific</u> child terminates
- What are the parameters?

man page of wait(2)

```
> man -s 2 wait
WAIT(2)     Linux Programmer's Manual         WAIT(2)

NAME
 wait, waitpid - wait for process termination

SYNOPSIS
 #include <sys/types.h>
 #include <sys/wait.h>

 pid_t wait(int *status);
 pid_t waitpid(pid_t pid, int *status, int options);
…
```

Parameters to wait(2)

- Address of a *status*
- Status is the return value
- Example:

```
int pid, status;
…
pid = wait(&status);
…
```

System call suspends execution of calling process until one of its children terminates. If status is not NULL, it indicates the status of the child process upon return.

Params for waitpid(2)

- PID of child to watch
- Address of a *status*
- Some options
- Example:

```
int cpid, pid, status;
…
cpid = fork();
…
pid = waitpid(cpid, &status, WNOHANG);
…
```

WNOHANG: return immediately if child’s status information is not available.

Use in the myshell program

- Use before parent terminates
- Wait for ALL children to terminate
- Read the man page and chose wisely

How can all this be used in the `myshell`?

myshell.c

1. Prompt and parse input (parse.c)
2. Call `fork()` to create children as directed by user.
3. For each child:
   1. Child calls `exec*()` to run other program
   2. Child should never return from `exec*()`
4. Parent waits for children to finish
5. Parent returns to 1

Basic Check

```
> ./myshell
$$$ ./collatz 4 1000
<output of all collatz instances>
$$$ ./prime 4 1000000
<output of all prime instances>
$$$ exit
>
```

What to do & Not to do

- Do not write `prime`, `collatz`, etc.
- Do not write system programs
- Do write `myshell` to use
  - `fork()` to create a new process
  - `exec*()` to run a pre-existing program
  - `wait()` to prevent zombies
- Should be able to run ANY user program from your shell.

I/O Redirect

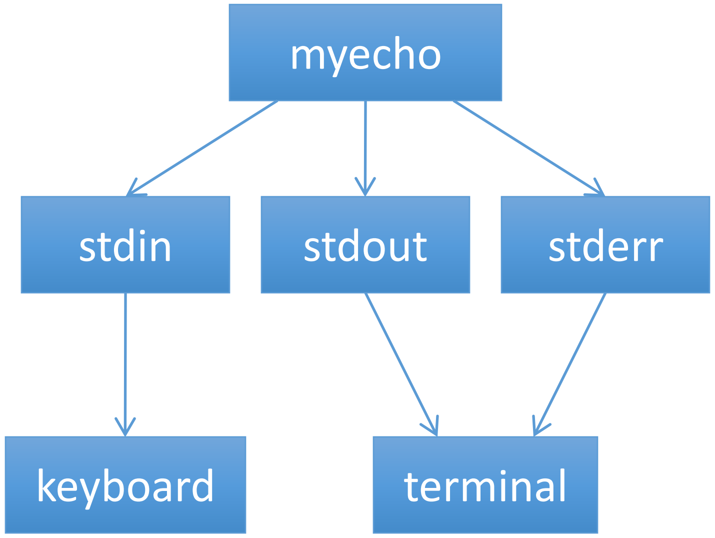
- Three files open for a process by default:
  - `stdin` refers to the keyboard
  - `stdout` refers to the monitor (window)
  - `stderr` refers to the monitor (window)
- Programs can read `stdin` / write `stdout`

```
printf("Hello, world\n");
gets(str);
fgets(str, 256, stdin);
```

Program Design

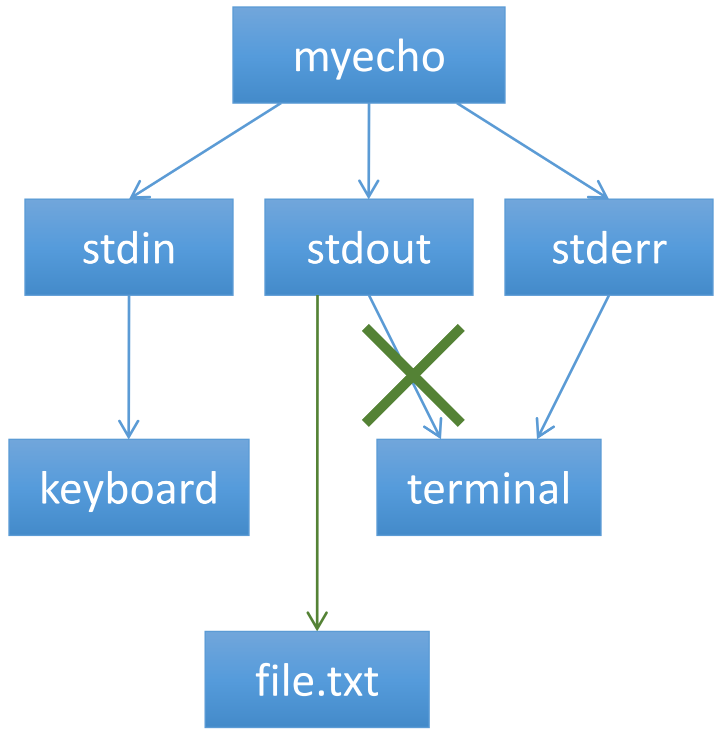
- Lots of programs read `stdin` & write `stdout` by default
- Assume `myecho` duplicates input to output

```
> ./myecho
hello
hello
There once was a
There once was a
CTRL-D
>
```



Redirect Output

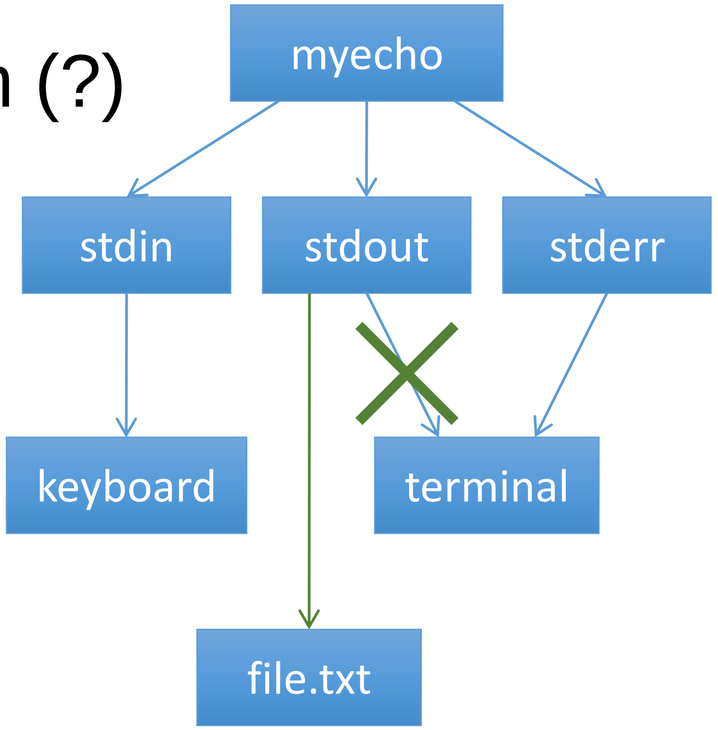
```
> ./myecho >file.txt
One fish, two fish
Red fish, blue fish
Old fish, new fish
Black fish, blue fish
> cat file.txt
One fish, two fish
Red fish, blue fish
Old fish, new fish
Black fish, blue fish
>
```



freopen(3)

- Associates a file with a stream (?)
- File is the output file
- Stream is `stdout`

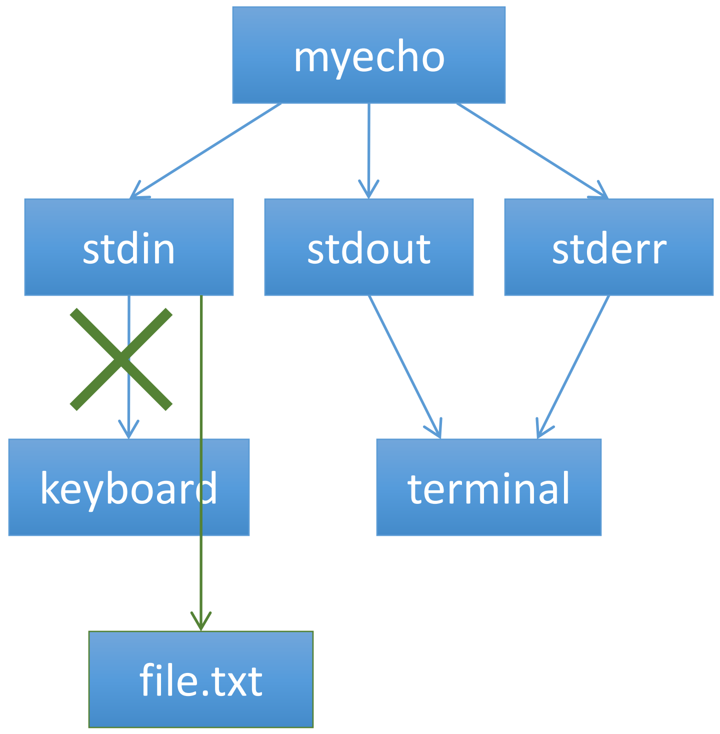
```
FILE *fp;
…
fp=freopen(filename,"w", stdout);
…
printf(“Hello, world\n”);
fprintf(stderr, “Error!\n”);
```



Redirect Input

- Use the < to redirect input

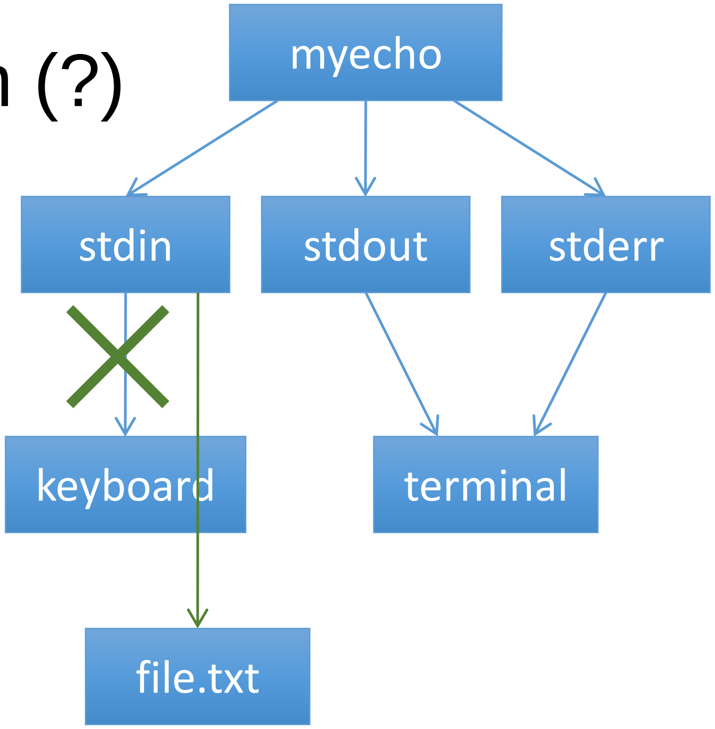
```
> ./myecho <file.txt
One fish, two fish
Red fish, blue fish
Old fish, new fish
Black fish, blue fish
>
```



freopen(3)

- Associates a file with a stream (?)
- File is the input file
- Stream is `stdin`

```
FILE *fp;
…
fp=freopen(filename, "r", stdin);
…
gets(buffer);
```



Shell Revisited

1. Prompt and parse input (previous project).
2. If input is exit, do termination stuff.
3. Call `fork()` as many times as the user requested to create child processes.
4. If executing in child:
   a. If (redirect input)

      `freopen(…, "r", stdin);`

   b. If (redirect output)

      `freopen(…, "w", stdout);`

   c. Call `exec*()` to run other program
5. Child should never return from `exec*()`.
6. Parent waits for all children to finish.
7. Parent returns to 1.

Parent Termination

- Any number of children may still be active.
- Need to keep track of them.
- Some may terminate, others become daemons.
- How many are there?
- Read the man page for `waitpid()`.
- Pay close attention to the parameters.
  - Any special values that can be used?
- Read about the return values.
  - What if we have no children?

Programming Assignment

Write

`myshell!`

Summary

- A process may not terminate before its children have terminated unless it is a daemon process.
- wait() and waitpid() may be used for process to wait for another process, to prevent zombies.
- freopen() may be used to redirect IO to/from a file.