### Features/Issues

- **Handling ctrl-c from keyboard**
  - Should kill all processes in a foreground pipeline
    - `ls -l | sort -n +4 | more`
  - What about background pipelines???
- **Multiple signal interruptions**
- **Pipeline in foreground**
- **Pipeline in background**
- **Job control**
- **Miscellaneous**
  - Parsing pipeline
  - Check #args to built-in commands
    - `mytime` and `mylimit` commands

### Signals

- **xssh should go back to prompt if user enters ctrl-c**
  - Example: Abort long-running foreground command
    - Ctrl-c causes SIGINT to be delivered to process
      - Default: Abnormal termination
  - Use `sigsetjmp` and `siglongjmp`
    - May require unraveling the program stack if buried deep into many function calls
- **Multiple signal interrupts**
  - Probably not a major issue in Project B
  - See simple guidelines at end of signals lecture

### Web Resources

- **test40.C**
  - Shows effect of handling or ignoring SIGINT (ctrl-c)
- **doio.c**
  - Shows: read stdin/write stdout using unbuffered I/O
- **Homework 7, Problem solution (To Come)**
  - Shows how to separate plumbing and I/O
    - But different structure than Project B
  - `do_plumbing(); do_io();`
    - `do_plumbing:` for (int i=0; i<npipes; i++) {Pipe(fd); ...}
    - `do_io: while (Read(STDIN_FILENO, ...) > 0) Write(STDOUT_FILENO, ...);`
Siglongjmp and Sigsetjmp

- **sigsetjmp** is analogous to a statement label
  - Like placing a marker at the current location
  - int sigsetjmp(sigjmp_buf env, int savemask);
    - env is initialized with information needed to jump back to the current location
    - Save the current signal mask in the env buffer if savemask is nonzero
    - Return value is 0 when directly called
    - Return value is val argument of siglongjmp(env, val) otherwise

- **siglongjmp** is analogous to a goto statement
  - void siglongjmp(sigjmp_buf env, int val);

Siglongjmp and Sigsetjmp Example

```
static sigjmp_buf jmpbuf;
static volatile sig_atomic_t jumpok = 0;
static void handler(int signo) {
    if (jumpok == 0) return;
    siglongjmp(jmpbuf, 1);
}
int main(void) {
    struct sigaction act;
    act.sa_flags = 0;
    act.sa_handler = handler;
    if ( (sigemptyset(&act.sa_mask) == -1) ||
        (sigaction(SIGINT, &act, NULL) == -1) ) … Error/Exit …
    sigsetjmp(jmpbuf, 1); // return here from handler
    jumpok = 1;
    for ( ; ; ) { … Main Loop … }
}
```

Pipeline (Review)

- Example (sort by non-decreasing file size)
  - ls -l | sort -n +4

```
File Descriptor Tables

Was stdin: [0]
standard input [0]
pipe (read) [1]

Was stdout: [2]
pipe (write) [1]
standard output [2]
standard error [2]
```

- Needed:
  - Create pipe
  - Replace STDIN/STDOUT with end of a pipe

```
Pipeline Movie (1)
```

```
xssh child: [0] stdin
[1] stdout
[2] stderr

[0] stdin
[1] stdout
[2] stderr
```

```
xssh child: fork
```

```
[0] stdin
[1] stdout
[2] stderr
```

```
xssh child: pipe
```

```
[0] stdin
[1] stdout
[2] stderr
```

```
fd[0] = [3] pipe.I
```

```
xssh child: pipe
```

```
[0] stdin
[1] stdout
[2] stderr
```

```
```

```
xssh child: pipe
```

```
[0] stdin
[1] stdout
[2] stderr
```

```
```

```
xssh child: pipe
```

```
[0] stdin
[1] stdout
[2] stderr
```

```
```

```
xssh child: pipe
```

```
[0] stdin
[1] stdout
[2] stderr
```

```
```
Pipeline Movie (2)

```
xssh child:  dup2(fd[1], X);
close(fd[0]);
close(fd[1]);
child:  dup2(fd[0], Y);
close(fd[0]);
close(fd[1]);
```

where X is STDOUT_FILENO
Y is STDIN_FILENO

```
[0] stdin
[1] pipe
[2] stdout
[3] stderr
[4] 0
```

Very Important:
Close both ends of all pipes

Aside:
If xssh creates more than one pipe, close both ends of ALL pipes before using them

Foreground Pipeline

- Parsing
  "ls -l /usr/bin | sort -n +4 | more" becomes
  "ls -l /usr/bin", "sort -n +4", "more"
  - i.e., 3 strings, each representing a command
  - We know how to handle individual commands

- Approach
  - xssh sets up pipes for 3 children and fork-exec's the 3 processes for 'ls', 'sort' and 'more'
  - Creates 4 processes in the same process group
    - xssh is the process group leader; i.e., process group = pid of xssh
  - ctrl-c: SIGINT sent to and terminates all 4 processes !!!

- Need To:
  - Put the 3 children in a new process group
  - xssh should catch SIGINT; send SIGINT to processes

Process Group Example

```
#!/bin/sh
echo "pid of this shell is $$"
ls -l | sort -n +4 | mysleep $1 &
ps -o pid,ppid,pgrp,cmd
```

```
sh script that calls sleep command
```

```
pid of this shell is 27129
PID PPID PGRP CMD
17233 17232 17233 -tcsh
27129 17233 27129 /bin/sh ./script0 60
27131 27129 27129 /bin/sh ./mysleep 60
27134 27132 27129 sleep 60
```

```
"kill -9 -27129" kills all processes in group 27129
```

Process Group

- Def. A collection of processes that can receive signals from the same terminal
- Every process belongs to some process group
  - A process group leader is a process whose PID is the same as its PGRP
    - The leader can die or join another group
  - A child created by fork() inherits the PGRP of the parent
- Some functions
  - pid_t getpgrp(void)
    - Return PGRP of caller
  - int setpgid(pid_t pid, pid_t pgid)
    - Change PGRP of child process 'pid' (current process if 0)

- Beware of race conditions !!!
xssh

- Is in same process group as the shell that exec'd it
- Should ignore SIGINT
  - ctrl-c will terminate it's foreground children but not xssh
  - Won't terminate xssh's parent (login shell) either
    - Shell's ignore SIGINT
- Change process group of background pipelines
  - ctrl-c will have no effect on these processes

Miscellaneous

- mytime
  - Need to get usage stats of terminated child
  - See getrusage
- mylimit
  - A limited form of the ulimit Bash shell command
    - "ulimit -S 3" sets the CPU usage limit to 3 seconds
    - -S means it is a soft limit: i.e., can be modified later so long as it
      doesn't violate the hard limit
  - See setrlimit and getrlimit
- Design Guide (like A.txt)
  - See documentation template
  - Like design and implementation notes
  - Write a rough version before writing code and update it
    as you go