COMP 310

Ass 1 Tutorial

letao.chen@mail.mcgill.ca

09/14/29

Outline

1. Tiny Shell
2. wait(), waitpid()
3. fork(), execvp(), wait()
4. Debugging child processes
5. Internal commands
    a. chdir, history, limit
6. FIFOs
7. Signal Handling
8. pipe(), dup2()
9. Questions + Networking

Tiny Shell

`tshell < input.txt`

`ltrace tshell < input.txt`

- strace: show syscalls made
- ltrace : shows dynamic lib calls (higher level)

Useful flags:

1.	-o <file>	- output file
2.	-f	- follow child

```
while (1) {
    cmd = get_a_line();
    if (len(cmd) > 1)
        my_system(line);
}
```

Tiny Shell

`my_system()` spawns child process

- Example cmd: 'ls'
- If cmd fails, do not crash the entire shell

Do not put command in background

- Use `wait()` or `waitpid()` syscall

```
while (1) {
      cmd = get_a_line();
      if (len(cmd) > 1)
            my_system(line);
}
```

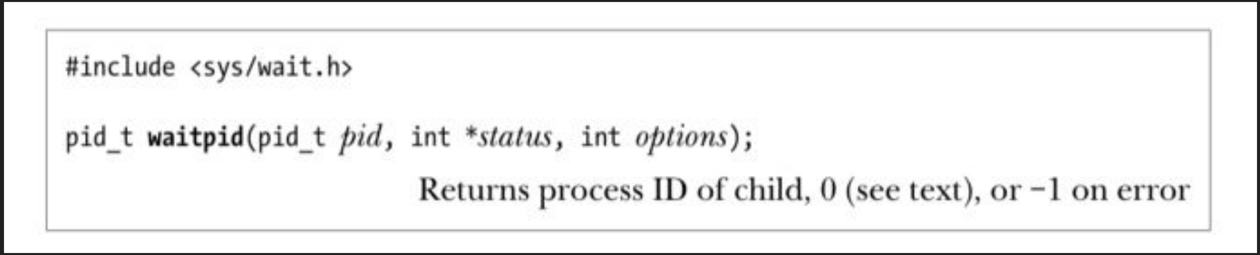
wait(), waitpid()

`wait(NULL)`
same as
`waitpid(-1, NULL, 0)`

Arguments:

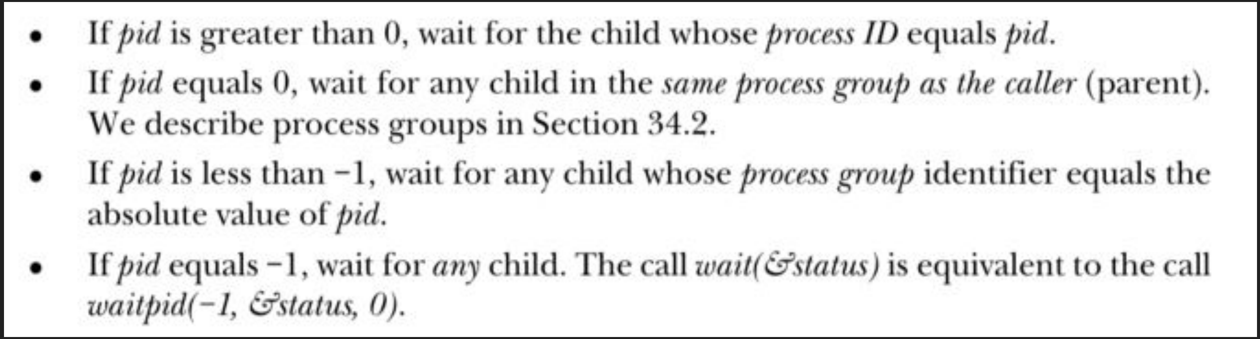
- NULL status
  - Ignore child status
- 0 options
  - No flags, just finish child

See man pages for more information



```
#include <sys/wait.h>

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

Returns process ID of child, 0 (see text), or −1 on error

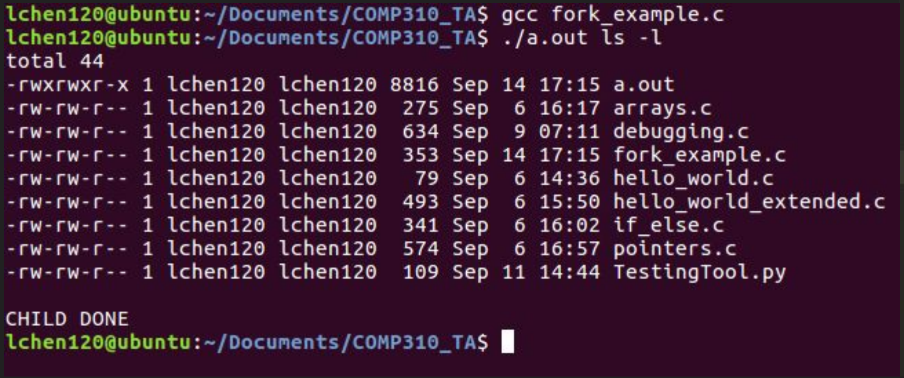


- If *pid* is greater than 0, wait for the child whose *process ID* equals *pid*.
- If *pid* equals 0, wait for any child in the *same process group as the caller* (parent). We describe process groups in Section 34.2.
- If *pid* is less than −1, wait for any child whose *process group* identifier equals the absolute value of *pid*.
- If *pid* equals −1, wait for *any* child. The call *wait(&status)* is equivalent to the call *waitpid(−1, &status, 0)*.

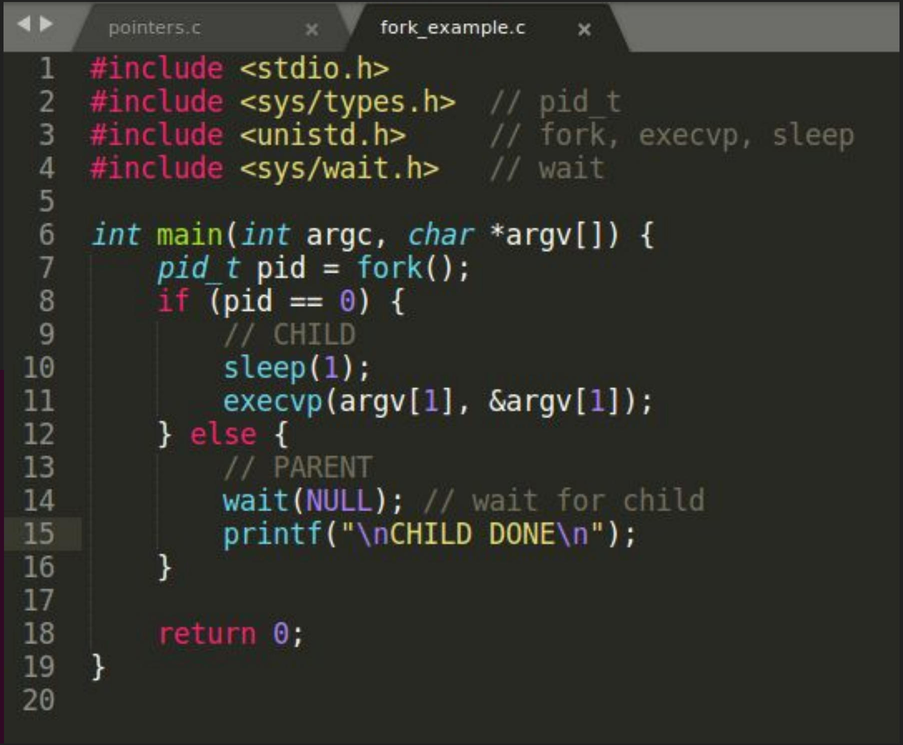
fork(), execvp(), wait()

Read man pages for extra info

If `execvp()` returns - something went wrong



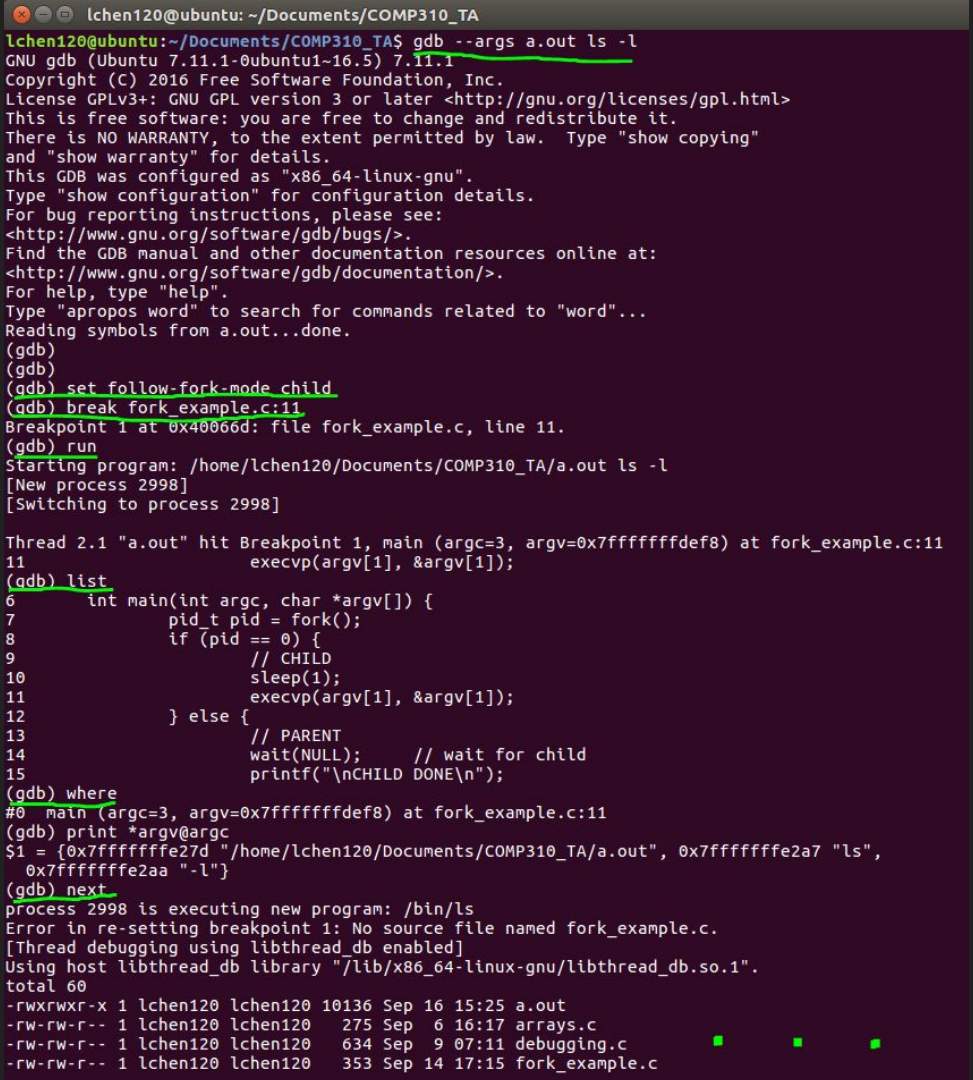
```
lchen120@ubuntu:~/Documents/COMP310_TA$ gcc fork_example.c
lchen120@ubuntu:~/Documents/COMP310_TA$ ./a.out ls -l
total 44
-rwxrwxr-x 1 lchen120 lchen120 8816 Sep 14 17:15 a.out
-rw-rw-r-- 1 lchen120 lchen120  275 Sep  6 16:17 arrays.c
-rw-rw-r-- 1 lchen120 lchen120  634 Sep  9 07:11 debugging.c
-rw-rw-r-- 1 lchen120 lchen120  353 Sep 14 17:15 fork_example.c
-rw-rw-r-- 1 lchen120 lchen120   79 Sep  6 14:36 hello_world.c
-rw-rw-r-- 1 lchen120 lchen120  493 Sep  6 15:50 hello_world_extended.c
-rw-rw-r-- 1 lchen120 lchen120  341 Sep  6 16:02 if_else.c
-rw-rw-r-- 1 lchen120 lchen120  574 Sep  6 16:57 pointers.c
-rw-rw-r-- 1 lchen120 lchen120  109 Sep 11 14:44 TestingTool.py

CHILD DONE
lchen120@ubuntu:~/Documents/COMP310_TA$
```



```c
#include <stdio.h>
#include <sys/types.h>  // pid_t
#include <unistd.h>     // fork, execvp, sleep
#include <sys/wait.h>   // wait

int main(int argc, char *argv[]) {
    pid_t pid = fork();
    if (pid == 0) {
        // CHILD
        sleep(1);
        execvp(argv[1], &argv[1]);
    } else {
        // PARENT
        wait(NULL); // wait for child
        printf("\nCHILD DONE\n");
    }

    return 0;
}
```

fork() and gdb

`set follow-fork-mode child`



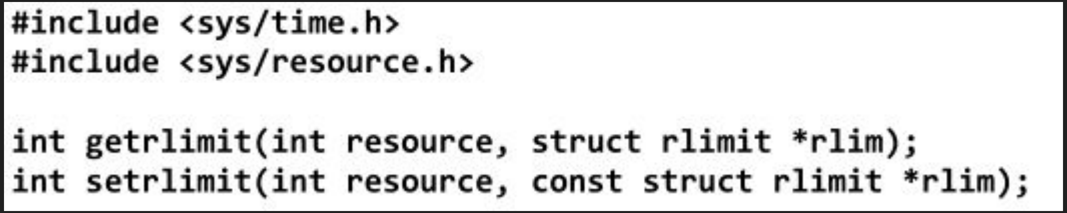
lchen120@ubuntu: ~/Documents/COMP310_TA

```
lchen120@ubuntu:~/Documents/COMP310_TA$ gdb --args a.out ls -l
GNU gdb (Ubuntu 7.11.1-0ubuntu1~16.5) 7.11.1
Copyright (C) 2016 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from a.out...done.
(gdb)
(gdb)
(gdb) set follow-fork-mode child
(gdb) break fork_example.c:11
Breakpoint 1 at 0x40066d: file fork_example.c, line 11.
(gdb) run
Starting program: /home/lchen120/Documents/COMP310_TA/a.out ls -l
[New process 2998]
[Switching to process 2998]

Thread 2.1 "a.out" hit Breakpoint 1, main (argc=3, argv=0x7fffffffdef8) at fork_example.c:11
11                      execvp(argv[1], &argv[1]);
(gdb) list
6       int main(int argc, char *argv[]) {
7               pid_t pid = fork();
8               if (pid == 0) {
9                       // CHILD
10                      sleep(1);
11                      execvp(argv[1], &argv[1]);
12              } else {
13                      // PARENT
14                      wait(NULL);     // wait for child
15                      printf("\nCHILD DONE\n");
(gdb) where
#0  main (argc=3, argv=0x7fffffffdef8) at fork_example.c:11
(gdb) print *argv@argc
$1 = {0x7fffffffe27d "/home/lchen120/Documents/COMP310_TA/a.out", 0x7fffffffe2a7 "ls",
  0x7fffffffe2aa "-l"}
(gdb) next
process 2998 is executing new program: /bin/ls
Error in re-setting breakpoint 1: No source file named fork_example.c.
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
total 60
-rwxrwxr-x 1 lchen120 lchen120 10136 Sep 16 15:25 a.out
-rw-rw-r-- 1 lchen120 lchen120   275 Sep  6 16:17 arrays.c
-rw-rw-r-- 1 lchen120 lchen120   634 Sep  9 07:11 debugging.c
-rw-rw-r-- 1 lchen120 lchen120   353 Sep 14 17:15 fork_example.c
```

chdir, history, limit

```
struct rlimit {  rlim_t rlim_cur;
                 rlim_t rlim_max;  }
```

`rlim_cur` is the soft limit, `rlim_max` is the hard limit (ceiling for `rlim_cur`)



```
#include <sys/time.h>
#include <sys/resource.h>

int getrlimit(int resource, struct rlimit *rlim);
int setrlimit(int resource, const struct rlimit *rlim);
```

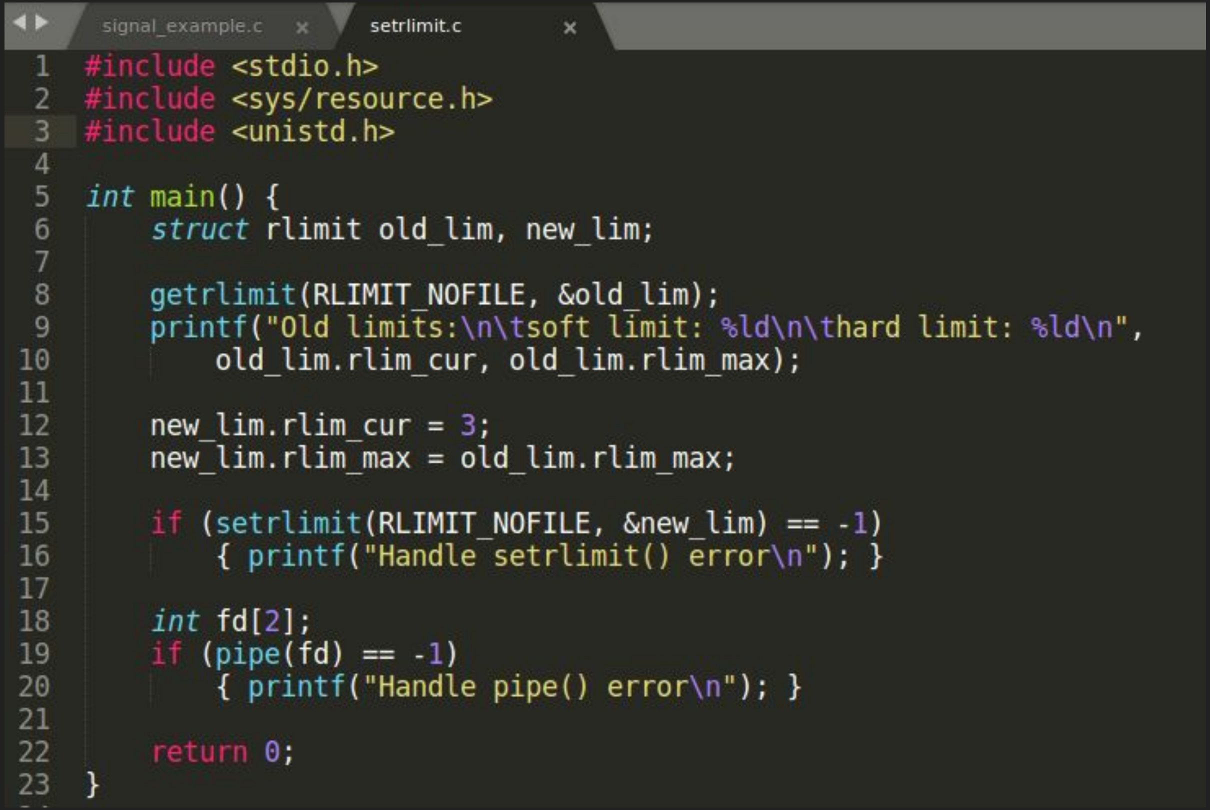
See man pages for `resource` options

chdir, history, limit

Example: limit number of files opened

Set new limit to 3

`pipe()` to simulate opening new files



```c
#include <stdio.h>
#include <sys/resource.h>
#include <unistd.h>

int main() {
    struct rlimit old_lim, new_lim;

    getrlimit(RLIMIT_NOFILE, &old_lim);
    printf("Old limits:\n\tsoft limit: %ld\n\thard limit: %ld\n",
        old_lim.rlim_cur, old_lim.rlim_max);

    new_lim.rlim_cur = 3;
    new_lim.rlim_max = old_lim.rlim_max;

    if (setrlimit(RLIMIT_NOFILE, &new_lim) == -1)
        { printf("Handle setrlimit() error\n"); }

    int fd[2];
    if (pipe(fd) == -1)
        { printf("Handle pipe() error\n"); }

    return 0;
}
```

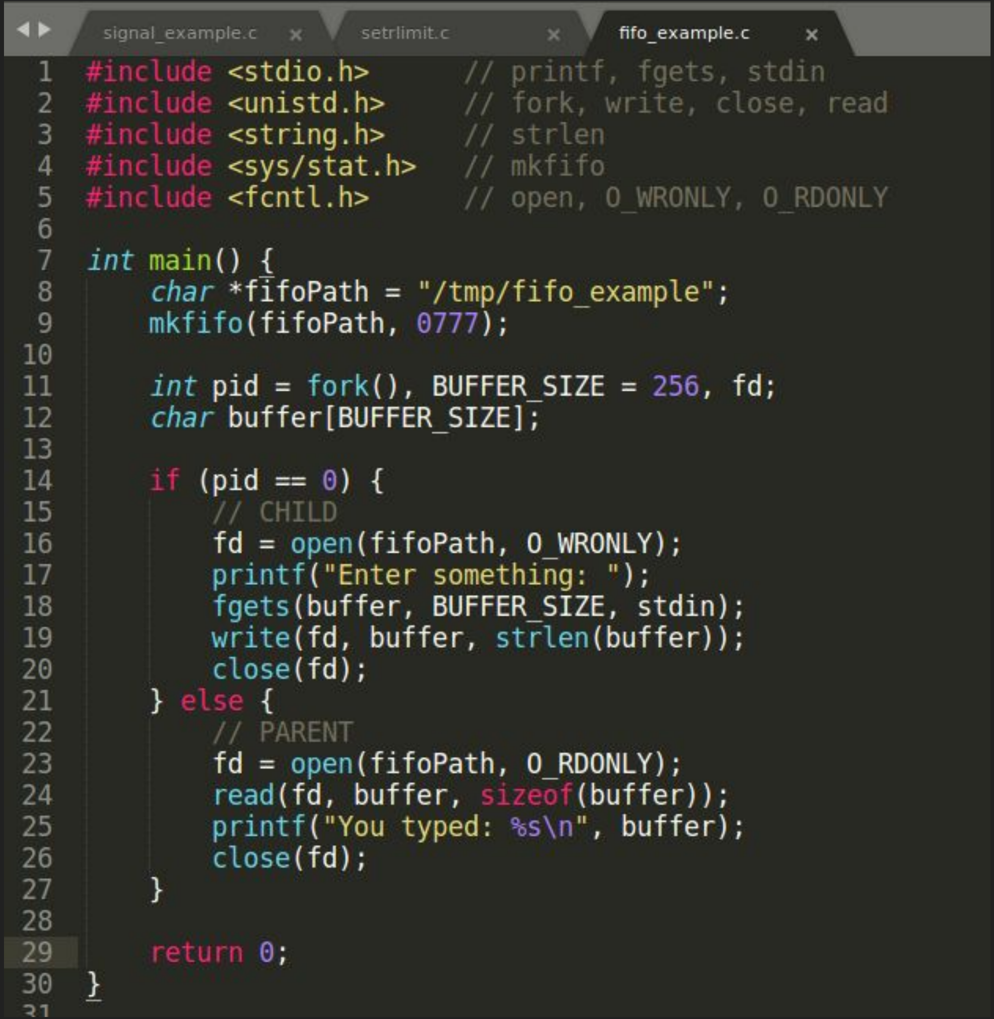
FIFOs

Example program:

1. Child gets user input and writes to the FIFO
2. Parent reads from the FIFO and prints

Example testing procedure for Graders:

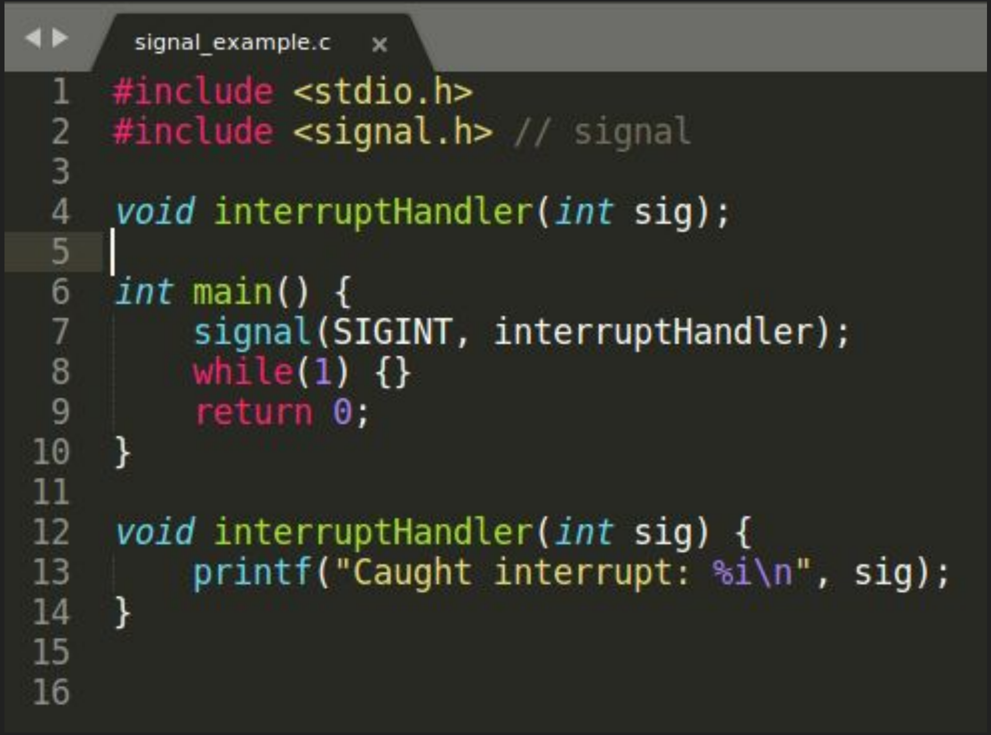
1. $ mkfifo <mode> <pathname>
2. Create 2 tiny shell instances
   a. $ tshell <pathname> < input.txt
   b. $ tshell <pathname> < input2.txt



```c
#include <stdio.h>      // printf, fgets, stdin
#include <unistd.h>     // fork, write, close, read
#include <string.h>     // strlen
#include <sys/stat.h>   // mkfifo
#include <fcntl.h>      // open, O_WRONLY, O_RDONLY

int main() {
    char *fifoPath = "/tmp/fifo_example";
    mkfifo(fifoPath, 0777);

    int pid = fork(), BUFFER_SIZE = 256, fd;
    char buffer[BUFFER_SIZE];

    if (pid == 0) {
        // CHILD
        fd = open(fifoPath, O_WRONLY);
        printf("Enter something: ");
        fgets(buffer, BUFFER_SIZE, stdin);
        write(fd, buffer, strlen(buffer));
        close(fd);
    } else {
        // PARENT
        fd = open(fifoPath, O_RDONLY);
        read(fd, buffer, sizeof(buffer));
        printf("You typed: %s\n", buffer);
        close(fd);
    }

    return 0;
}
```

Signal Handling

CTRL - C	- raises SIGINT
CTRL - Z	- raises SIGTSTP



signal_example.c

```c
#include <stdio.h>
#include <signal.h> // signal

void interruptHandler(int sig);

int main() {
    signal(SIGINT, interruptHandler);
    while(1) {}
    return 0;
}

void interruptHandler(int sig) {
    printf("Caught interrupt: %i\n", sig);
}
```

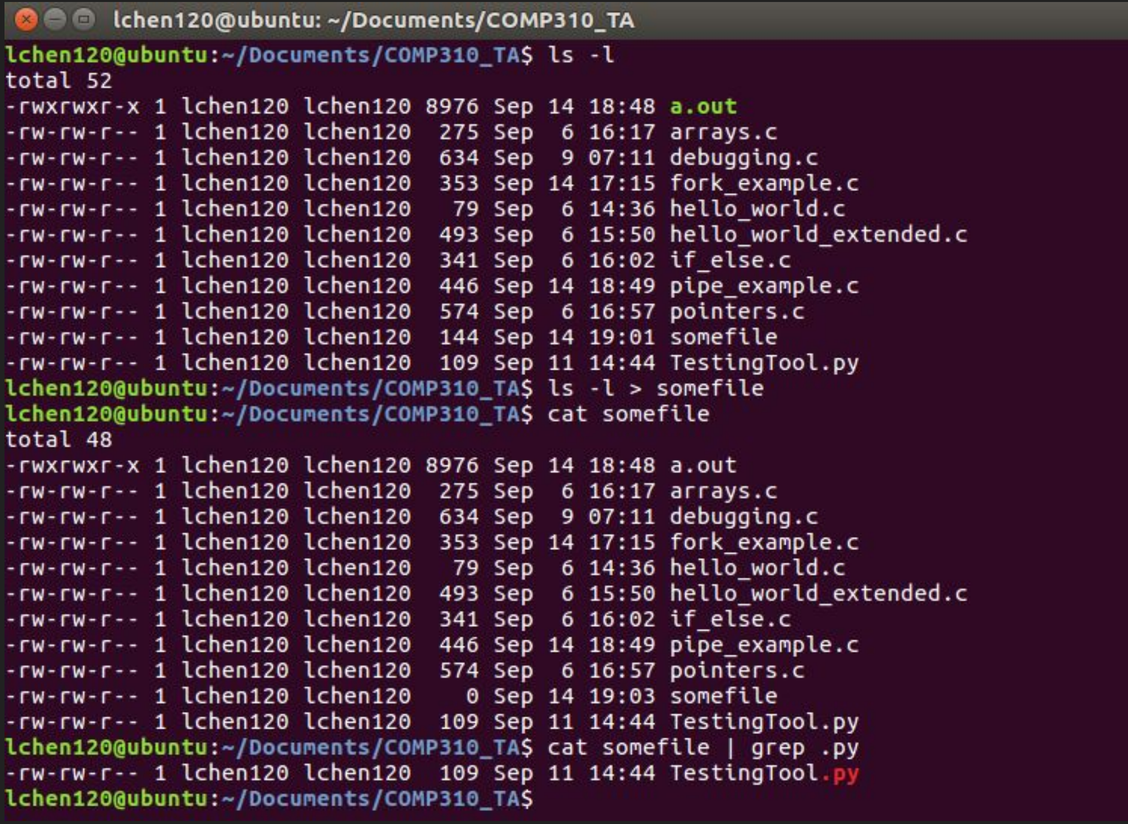
pipe()

stdin (0), stdout (1), stderr (2)

Redirect stdout to somefile

Pipe stdout to stdin for grep

Very powerful for `tail -f <log_file> | grep ...`



lchen120@ubuntu: ~/Documents/COMP310_TA

```
lchen120@ubuntu:~/Documents/COMP310_TA$ ls -l
total 52
-rwxrwxr-x 1 lchen120 lchen120 8976 Sep 14 18:48 a.out
-rw-rw-r-- 1 lchen120 lchen120  275 Sep  6 16:17 arrays.c
-rw-rw-r-- 1 lchen120 lchen120  634 Sep  9 07:11 debugging.c
-rw-rw-r-- 1 lchen120 lchen120  353 Sep 14 17:15 fork_example.c
-rw-rw-r-- 1 lchen120 lchen120   79 Sep  6 14:36 hello_world.c
-rw-rw-r-- 1 lchen120 lchen120  493 Sep  6 15:50 hello_world_extended.c
-rw-rw-r-- 1 lchen120 lchen120  341 Sep  6 16:02 if_else.c
-rw-rw-r-- 1 lchen120 lchen120  446 Sep 14 18:49 pipe_example.c
-rw-rw-r-- 1 lchen120 lchen120  574 Sep  6 16:57 pointers.c
-rw-rw-r-- 1 lchen120 lchen120  144 Sep 14 19:01 somefile
-rw-rw-r-- 1 lchen120 lchen120  109 Sep 11 14:44 TestingTool.py
lchen120@ubuntu:~/Documents/COMP310_TA$ ls -l > somefile
lchen120@ubuntu:~/Documents/COMP310_TA$ cat somefile
total 48
-rwxrwxr-x 1 lchen120 lchen120 8976 Sep 14 18:48 a.out
-rw-rw-r-- 1 lchen120 lchen120  275 Sep  6 16:17 arrays.c
-rw-rw-r-- 1 lchen120 lchen120  634 Sep  9 07:11 debugging.c
-rw-rw-r-- 1 lchen120 lchen120  353 Sep 14 17:15 fork_example.c
-rw-rw-r-- 1 lchen120 lchen120   79 Sep  6 14:36 hello_world.c
-rw-rw-r-- 1 lchen120 lchen120  493 Sep  6 15:50 hello_world_extended.c
-rw-rw-r-- 1 lchen120 lchen120  341 Sep  6 16:02 if_else.c
-rw-rw-r-- 1 lchen120 lchen120  446 Sep 14 18:49 pipe_example.c
-rw-rw-r-- 1 lchen120 lchen120  574 Sep  6 16:57 pointers.c
-rw-rw-r-- 1 lchen120 lchen120    0 Sep 14 19:03 somefile
-rw-rw-r-- 1 lchen120 lchen120  109 Sep 11 14:44 TestingTool.py
lchen120@ubuntu:~/Documents/COMP310_TA$ cat somefile | grep .py
-rw-rw-r-- 1 lchen120 lchen120  109 Sep 11 14:44 TestingTool.py
lchen120@ubuntu:~/Documents/COMP310_TA$
```

pipe()

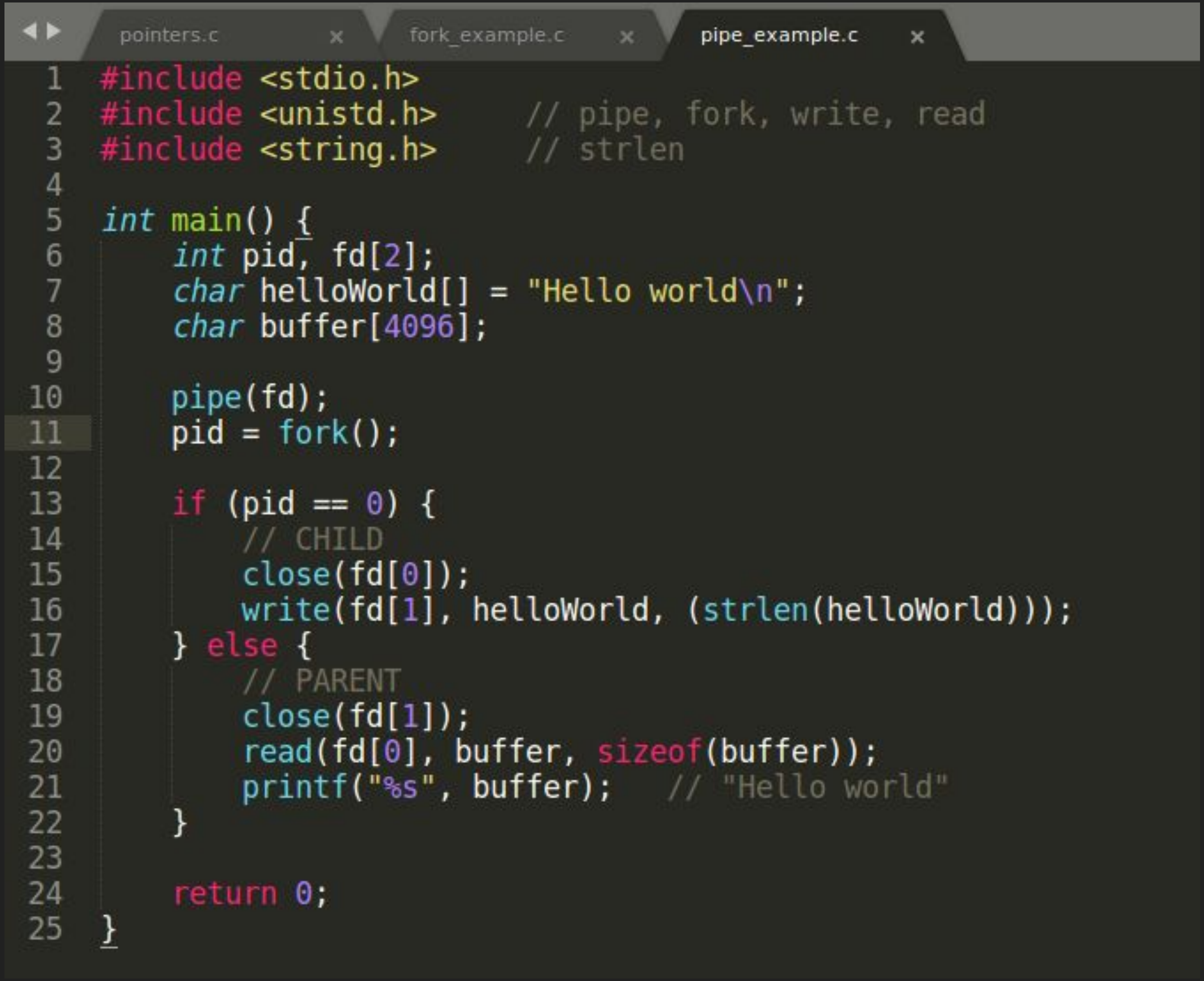
Child inherits new fds

- fd[0] - reading
- fd[1] - writing

Output of fd[1] becomes input for fd[0]

Example where parent receives data from the child

1. Child closes unused fd[0]
2. Child writes to fd[1] - not stdout
3. Parent closes unused fd[1]
4. Parent reads from fd[0] - not stdin



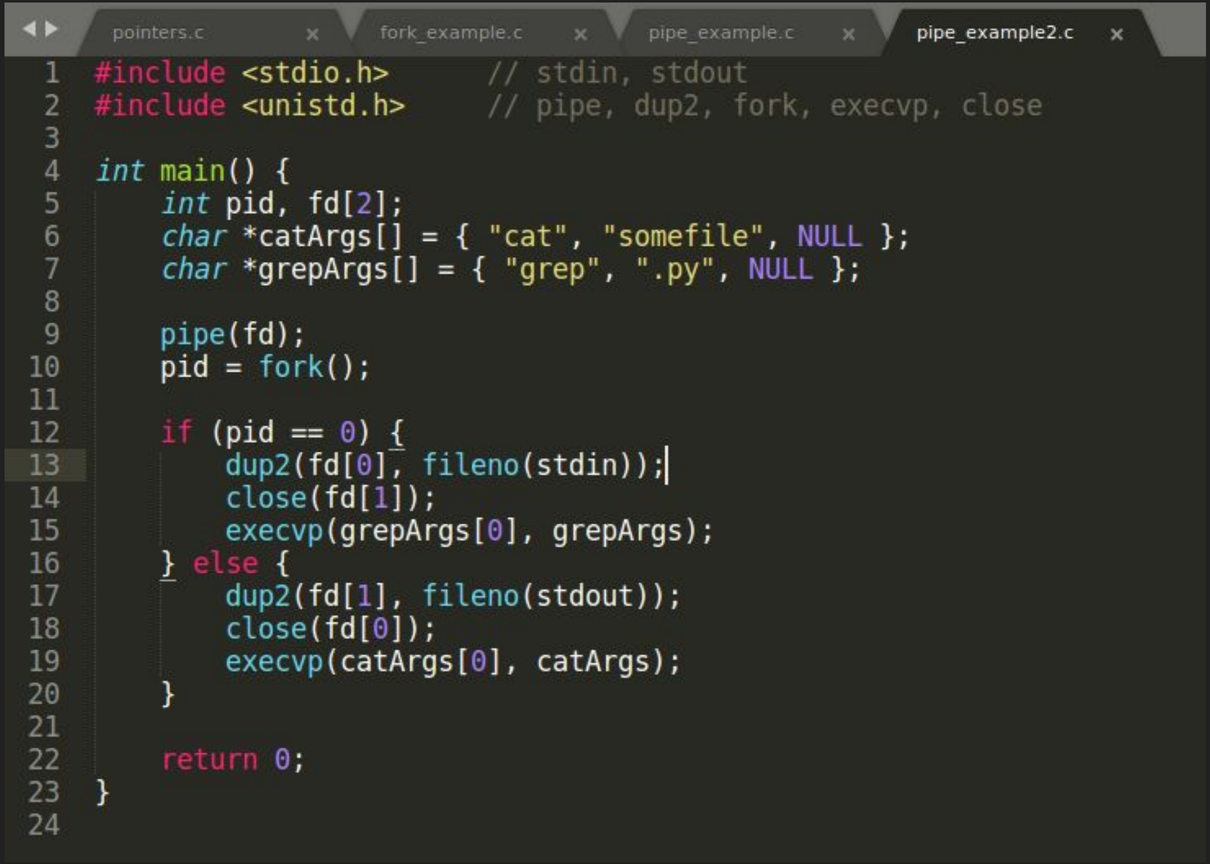
```c
#include <stdio.h>
#include <unistd.h>     // pipe, fork, write, read
#include <string.h>     // strlen

int main() {
    int pid, fd[2];
    char helloWorld[] = "Hello world\n";
    char buffer[4096];

    pipe(fd);
    pid = fork();

    if (pid == 0) {
        // CHILD
        close(fd[0]);
        write(fd[1], helloWorld, (strlen(helloWorld)));
    } else {
        // PARENT
        close(fd[1]);
        read(fd[0], buffer, sizeof(buffer));
        printf("%s", buffer);   // "Hello world"
    }

    return 0;
}
```

pipe(), dup2()

Same as `cat somefile | grep .py`

In this example, the child will receive data from the parent

1. Parent redirects stdout to fd[1]
2. Parent closes unused fd[0]
3. Parent executes `cat`
4. Child redirects stdin to fd[0]
5. Child closes unused fd[1]
6. Child executes `grep`



```c
#include <stdio.h>      // stdin, stdout
#include <unistd.h>     // pipe, dup2, fork, execvp, close

int main() {
    int pid, fd[2];
    char *catArgs[] = { "cat", "somefile", NULL };
    char *grepArgs[] = { "grep", ".py", NULL };

    pipe(fd);
    pid = fork();

    if (pid == 0) {
        dup2(fd[0], fileno(stdin));
        close(fd[1]);
        execvp(grepArgs[0], grepArgs);
    } else {
        dup2(fd[1], fileno(stdout));
        close(fd[0]);
        execvp(catArgs[0], catArgs);
    }

    return 0;
}
```

Questions + Networking

Discussion is encouraged. Absolutely no plagiarism, however.