ECE 344: Operating Systems
Lecture 16

Lab 4 Primer

1.0.0

Jon Eyolfson
October 17, 2022

We Want to Send and Recieve Data From a Process

1. Create a new process that launches the command line argument
2. Send the string `Testing\n` to that process
3. Receive any data it writes to standard output

Our First New API — `pipe`

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

Returns `0` on success, and `-1` on failure (and sets `errno`)

`pipe` forms a one-way communication channel using two file descriptors

- `pipefd[0]` is the `read` end of the pipe
- `pipefd[1]` is the `write` end of the pipe

You can think of it as a kernel managed buffer

- Any data written to one end can be read on the other end

A More Convenient API – `execlp`

```
int execlp(const char *file, const char *arg /*..., (char *) NULL */);
```

Does not return on success, and `-1` on failure (and sets `errno`)

`execlp` will let you skip using string arrays (using C varargs),
and it will also search for executables using the PATH environment variable

Our Next API — `close`

```
int close(int fd);
```

Returns `0` on success, and `-1` on failure (and sets `errno`)

Closes the file descriptor for the process, no longer usable

This frees up the file desciptor (recall, it's just a number) to be reused

Our Final APIs — dup and dup2

```
int dup(int oldfd);
int dup2(int oldfd, int newfd);
```

Returns a new file descriptor on success, and `-1` on failure (and sets `errno`)

Copies the file descriptor so `oldfd` and `newfd` refer to the same thing

For `dup` it'll return the lowest file descriptor

For `dup2` it'll atomically close the `newfd` argument (if open),
and then make `newfd` refer to the same thing

Coding Example

Done live!

You can find the template in `lecture-16` in the `examples` repository

To compile it, run the following commands:

```
cd lecture-16 # if not already there
mkdir build
cd build
cmake ..
cmake --build . # or make
```

Run the program using: `./subprocess <program>`

Running with `cat` May Cause Problems

Run the program with the following arguments:

```
./subprocess uname
./subprocess cat
```

You have to be careful with the file descriptors!

Why might `cat` not exit when using pipes?