Recitation 3: File Descriptors and Pipes

Department of Computer Science and Engineering
University of Minnesota

June 23, 2025

Overview

- File Descriptors.
- File Tables.
- File Redirection.
- Pipes.
- Exercise 1 - Redirection
- Exercise 2 - Pipes

File Descriptors.

- File Descriptors: Unique IDs used by OS when files are opened.
- Rec 2: System File I/Os use file descriptors for file manipulation. Standard File I/Os are wrappers over System File I/O.
- Everything is a file in OS: disk, devices, communication interfaces, ...
- Standard FDs: `STDIN_FILENO(1)`, `STDOUT_FILENO(2)`, `STDERR_FILENO(3)`: Always opened in a process

FDs and Processes

- File descriptors created in Parent processes are shared across children.
- Changes made by one process using an FD is reflected in all other processes.
- The FD needs to be closed in all processes.
- sample code: sample/p1.c

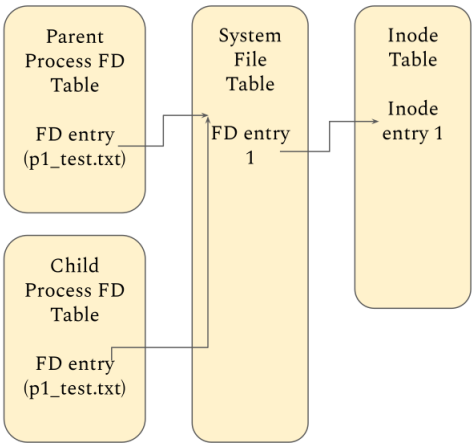
Inode

- Data structure to represent file system objects like files/directories.
- Unique to each file.
- Check man page of `stat` for more information
- Sample code: samples/p2.c

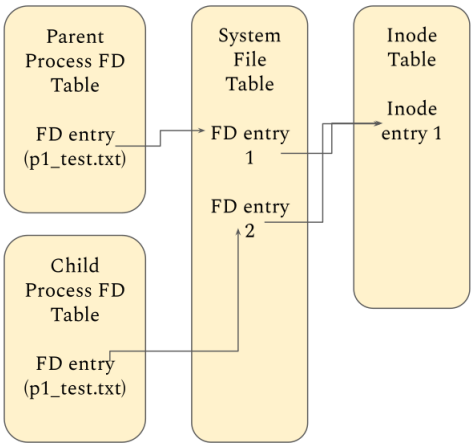
Inode(cont.)

```
#include <sys/type.h>
#include <sys/stat.h>
#include <unistd.h>
int stat(const char *pathname, struct stat *statbuf);
// pathname: file/directory name/path
// statbuf: on successful execution of stat(), statbuf is
// populated with information on file
// returns -1 on failure, otherwise 0
struct stat{
  ino_t st_ino; /* Inode number */
  mode_t st_mode; /* File type and mode */
};
```

File Tables



Parent opened file and fd inherited by child



Parent and child opened file separately

File Redirection

- On terminal, try the following:
  - `ls -l > tmp.txt`
  - The listed files/directories are written to `tmp.txt`
  - `STDOUT_FILENO` is redirected to `tmp.txt`
- Redirection creates/reuses an entry in process file descriptor table to point to an existing entry in system file table
- Programmatically redirection could be achieved using `dup()` and `dup2()`

File Redirection(cont.)(sample code: samples/p3.c and p4.c)

```
#include <unistd.h>
int dup(int oldfd);
// dup() creates a copy of the file descriptor oldfd,
// using the lowest-numbered unused file descriptor
// for the new descriptor.

int dup2(int oldfd, int newfd);
// dup2() uses the newfd to create a copy of oldfd.
// If newfd corresponds to an open file, it is silently
// closed, before reusing.
```

Pipes

- On terminal, try the following:
  - `ls -l | sort`
  - `ps -e | grep systemd`
- Virtual file/buffer for inter process communication, no filename.
- One a process and its descendants could use a pipe.
- At any point of time, data flow is unidirectional.
- One end of the pipe is using for writing and the other end for reading.
- Always close the unused end of the pipe in a process.

Pipes(samples/p5.c)

```
#include <unistd.h>
int pipe(int fd[2]);
// fd[0]: read end
// fd[1]: write end
// returns 0 on success, otherwise -1.
```

Exercises

1. Redirection: Complete **exercises/redirect.c**. First create a new file `tmp.txt` and write the inode number of `tmp.txt` to the file itself. Then duplicate `stdout` and redirect `stdout` to `tmp.txt`. Print the fd of `tmp.txt` and duplicated `stdout` (redirected to `tmp.txt`). Finally restore stdout using the duplicated stdout.
2. Pipes: Complete **exercises/server.c** and **exercises/child.c**. You are given code similar to samples/p5.c. This time, however, you will be passing the read end of the pipe to ./child.o as an argument using exec*. There are a few things to think about: how could you pass the read end as an argument, how will you close the pipe ends? You would require something similar for PA1 Phase 2 for communication across parent-child processes.

Deliverables

- Submit the completed exercises folder as a **tar.gz** to **GradeScope** by June 27th, 11:59 pm
- The sample codes should have all the necessary information for completing all the above exercises. If you have any questions, please post it to Piazza.
- Autograder will not be used as the output from each student will be different depending on the reporting style. Also, the order of output is not predictable as the OS schedules the processes spawned. Manual grading will be employed.