CIS 452 - Operating Systems Concepts

Nathan Bowman

Images taken from Silberschatz book

Pipes

One specific type of message passing you should all be familiar with is pipes

```
cat foo.txt | sort | uniq
```

Output from one process is sent (as a message) to another process

Types of pipes

- Bidirectional or unidirectional?
- Half-duplex or full-duplex? (applies only if bidirectional)
- Special relationship (e.g., parent-child) required between processes?

Ordinary (Unnamed) Pipes

```
cat foo.txt | sort
```

- Unidirectional (write end and read end)
- Can use two pipes if bidirectional communication required
- Processes must have parent-child relationship (see why later)

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

On success, OS overwrites entires of `pipefd` with two file descriptors

- `pipefd[0]` is read end
- `pipefd[1]` is write end

Can be used with `read` and `write` syscalls just like any other file descriptors

Only process with same file descriptors can access the pipe...

`fork()`!

```
#define BUFFER_SIZE 25
#define READ_END 0
#define WRITE_END 1

int main(void)
{
 char write_msg[BUFFER_SIZE] = "Greetings";
 char read_msg[BUFFER_SIZE];
 pid_t pid;
 int fd[2];

 /* create the pipe */
 if (pipe(fd) == -1) {
  fprintf(stderr,"Pipe failed");
  return 1;
 }

 /* now fork a child process */
 pid = fork();

 if (pid < 0) {
  fprintf(stderr, "Fork failed");
  return 1;
 }
```

```
 if (pid > 0) {  /* parent process */
  /* close the unused end of the pipe */
  close(fd[READ_END]);

  /* write to the pipe */
  write(fd[WRITE_END], write_msg, strlen(write_msg)+1);

  /* close the write end of the pipe */
  close(fd[WRITE_END]);
 }
 else { /* child process */
  /* close the unused end of the pipe */
  close(fd[WRITE_END]);

  /* read from the pipe */
  read(fd[READ_END], read_msg, BUFFER_SIZE);
  printf("child read %s\n",read_msg);

  /* close the write end of the pipe */
  close(fd[READ_END]);
 }

 return 0;
}
```

Once processes finish communicating, pipe ceases to exist

Named Pipes

Unlike ordinary pipes, *named pipes*

- are persistent in the filesystem
- can be used by many processes (not just parent and child)
- are bi-directional. However, communication is half-duplex, so second pipe is often opened if bi-directional communication is desired

Referred to as "FIFO"s in Linux

In C code, created by the `mkfifo` system call and used with standard (`read`, `write`, etc.) system calls for files

Exist in filesystem (though not stored on disk), so can be accessed by any process with appropriate permissions

Must be explicitly deleted

Default to blocking sends and receives

Can also be created in shell with `mkfifo` command

With `ls -l`:

```
prw-r--r-- 1 bowmnath bowmnath     0 Sep 14 15:44 myfifo|
```

You can experiment with FIFOs in the shell using `cat`

```
cat > myfifo
test message
<CTRL-D>
```

The `cat` instruction will block and `ls -l` will still show `myfifo` has size 0

To read (and unblock the `cat`), open another shell and try

```
cat myfifo
```