Inter-Process Communications. Pipes

Operating Systems part II

Viktor Iakovlev (Victor Yacovlev)

UNIX Process

- Isolated virtual memory
- Implemented by CPU

- Side Effect: it is required to communicate via Kernel to make processes to interact

Process Interaction

- Signals
  - UNIX Signals
  - POSIX Real-Time Extensions
- Shared Memory Mapping:
  mmap(..., MAP_SHARED, ...)
- Regular Files

UNIX Pipelines

```
          ls          -l          |          wc
#   command     argument     vert    command
```

- Two programs starts at a time
- Out of first one redirects to input of second one
- Implemented using **unnamed pipe**

UNIX Pipe

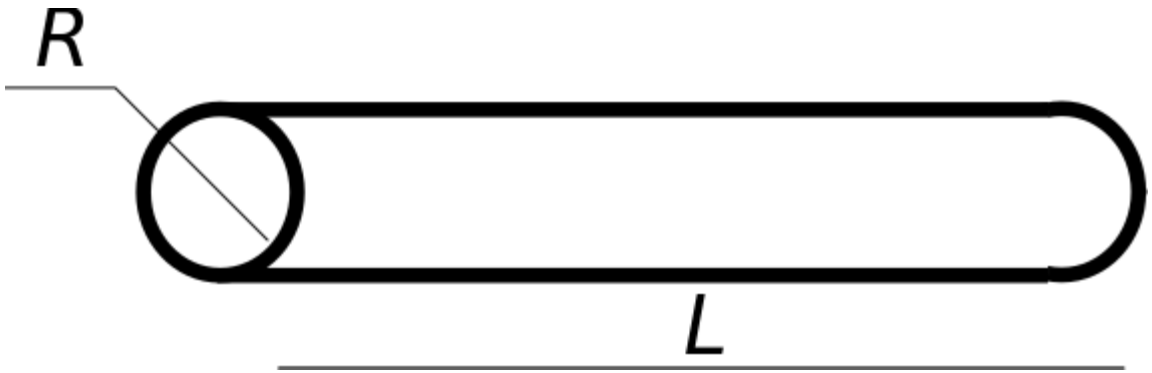
- System call `pipe(int fds[2])`
  - out-parameter - where to store fds
  - fds[0] - read-only, fds[1] - write-only

Pipes usage

- Interprocess Communications:
  - file descriptors are inherited by child processes
  - fork+exec keeps created pipe
- Within the same process:
  - one-threaded case: like a buffer
  - might be used for multithreading

Demo: pipe-one-process.c

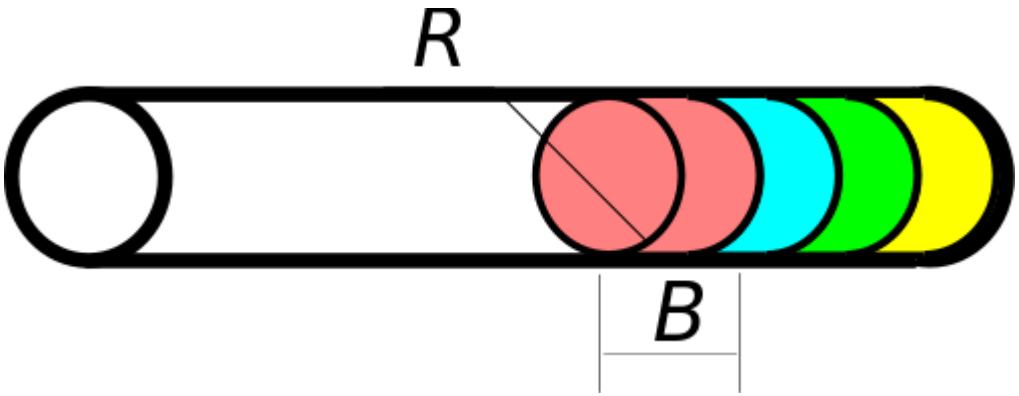
UNIX Pipe



$$V = \pi R^2 \cdot L = 65356 \text{ bytes}$$

(old Linux Kernels: only 4096 bytes)

UNIX Pipe



$$Q = \pi R^2 \cdot B = \text{PIPE\_BUF bytes}$$

- Atomicity is warranted for blocks of size `PIPE_BUF`
- For Linux (x86) `PIPE_BUF value is 4096`
- POSIX.1-2001 requires `PIPE_BUF >= 512`

Pipe write

- Its **required** than reading part of pipe is opened by some process
- If no more space in pipe buffer:
  - blocks until free space; or
  - special mode O_NONBLOCK fails write system call with EAGAIN
- «Broken pipe» error occurs in case of no one readers exist

Demo: pipe-one-process.c

Non-Blocking Read/Write Operations

- `O_NONBLOCK` flag for open system call
- Reading and writing operations do not the process, but fails with `EAGAIN error`
- Might be controled by `fcntl`

Pipe read

- If at least one byte in the buffer: no block, just read
- Waits until data ready (except O_NONBLOCK mode)
- If there is no writing side (e.g. opposite side of pipe was closed) - works like end-of-file

Dead Lock

- Trying to read a pipe, which has will be never written
- Usually due to forgotten `close` in some parent process

Demo: pipe-dead-lock.c

UNIX Pipelines

```
          ls          -l          |          wc
#   command     argument     vert    command
```

- 0 and 1 - stdin and stdout
- pipe(int fds[2]) created two **random** descriptor numbers

dup

- **dup2 - create a copy of file descriptor**
- The copy is just a reference for file-like object

dup v.s. dup2

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

1. Close newfd in case if in use
2. newfd = dup(oldfd)

Those two operations processed at once!

FIFO: Named Channels

- Unnamed channels might be used only by **relative** processes
  Example:
  `ls -l | wc`
  bash process creates a pipe, then creates two childs and both of them inherits the pipe
- Named channels (FIFO) allows to interact non-relatice processes

FIFO: Named Channels

- FIFO - special file type
- mkfifo - the command and function

- The special file is opened like a regular file
- But behaviour like a pipe

Demo: 1) mkfifo channel && strace cat channel; 2) echo hello >channel

Pipes in Real Life

- File descriptors 0, 1 and 2
- Might communicate several programs via pipeline

- Other cases: launch program in Terminal or IDE

- Usually unnamed pipes are in use, but not FIFO

Pipe’s Limitations

- Unidirectional: requires two pipes for bidirectional communications
- Synchronization is not easy when dealing with multiple readers/writers

Sockets v.s. Pipes

- Bidirectional
- Allows to detect events, but pipes - not
- Unified API for local and network communications

- Much more complex then pipes

The simpliest socket: socket pair

- FreeBSD and Linux only, but not any UNIX
- Bidirectional channel for link

```
socketpair(AF_LOCAL,
           SOCK_STREAM,
           0,
           out: int fds[2]
           ) → err;
```

Sockets

- The topic of next week
- Many types of sockets are declared, but two types in use:
  - local to computer (AF_LOCAL)
  - network (AF_INET or AF_INET6)