CSCI-3753: Operating Systems Fall 2019

Abigail Fernandes

Department of Computer Science

University of Colorado Boulder

PA 1 Learnings

- Prepare well for your interview
- Know what every command does
- Write correct code
- Brownie points for writing clean readable code
- Don’t miss your appointment
- No (TA) Office Hours during the week of interview grading

Week 5
Inter-Process Communication (IPC)

Visual Metaphor

IPC is like working together in a toy shop

- Workers share work area
  - Parts and tools on the table
- Workers call each other
  - Explicit requests and responses
- Requires synchronization
  - I'll start when you finish

Visual Metaphor

IPC is like working together in a toy shop

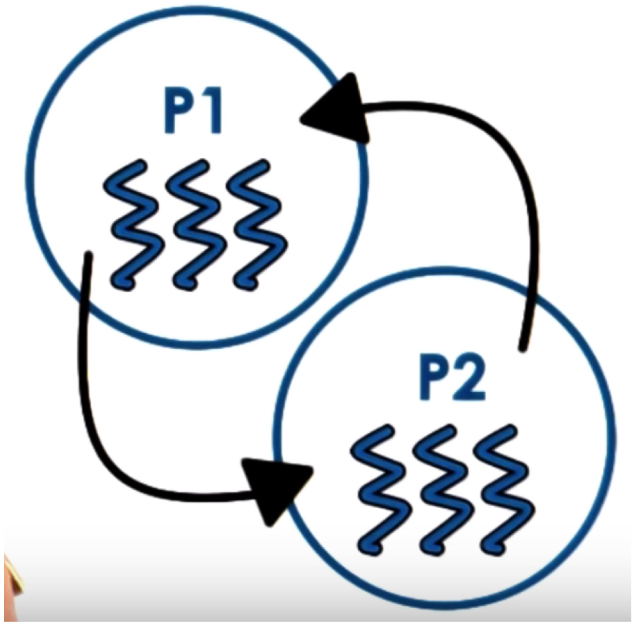
- Processes share memory
  - Data in shared memory
- Processes exchange messages
  - Message passing via sockets
- Requires synchronization
  - Mutexes, waiting

- Workers share work area
  - Parts and tools on the table
- Workers call each other
  - Explicit requests and responses
- Requires synchronization
  - I'll start when you finish

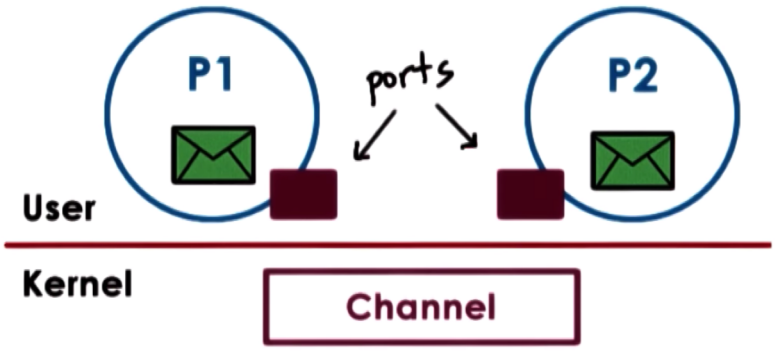
Inter Process Communication

IPC = **OS supported mechanisms for interaction amongst processes (co-ordination and communication)**

- Message Passing
  - Sockets, pipes, message queues
- Memory Based IPC
  - Shared memory, memory mapped files
- Higher level semantics
  - Files, RPC
- Synchronization primitives
  - Signals, interrupts

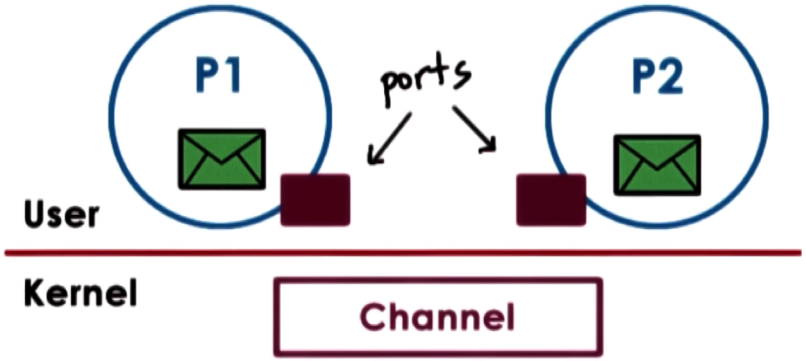


Message Passing



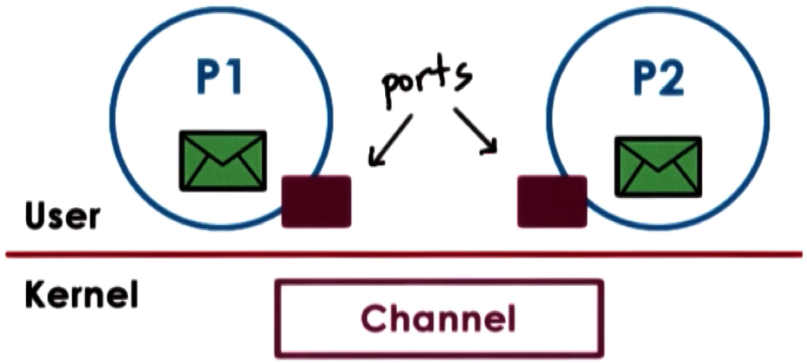
- Send / Receive messages
- OS creates and maintains a channel
  - Buffer, FIFO queue
- OS provides an interface to the process
  - A port
- Processes send/write messages to a port
- Processes receive/read messages from a port

Message Passing



- Kernel required to
  - Establish communication
  - Perform each IPC operation
- Send: System call+ data copy
- Receive: System call + data copy
- Request – Response Interaction
  - 4x user – kernel crossings + 4 data copies.

Message Passing

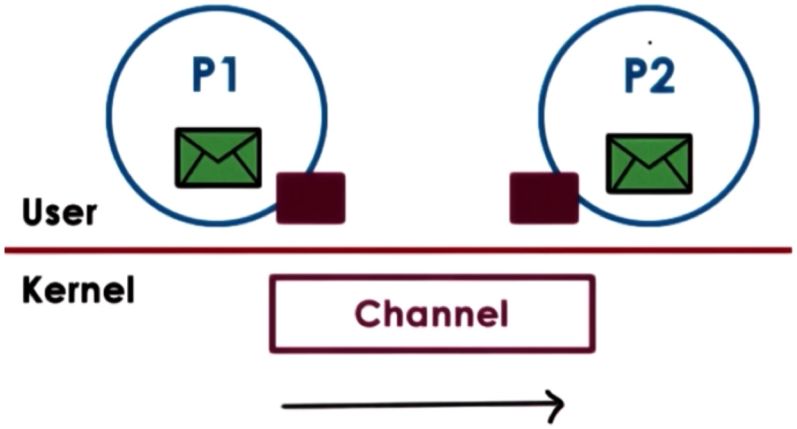


Overheads
Time Consuming

Simplicity: Kernel does channel management and synchronization

Message Passing (Pipes)

- Pipes
- Carry byte stream between 2 processes.
  - Eg. Connect output from one process to input of another.
- Unidirectional in nature
- Data written to the write end of the pipe is buffered by the kernel until it is read from the read end of the pipe.
- Read/Write system calls, used for I/O are implemented using pipes



Linux Pipes

- A pipe is created using the **`pipe()`** system call:

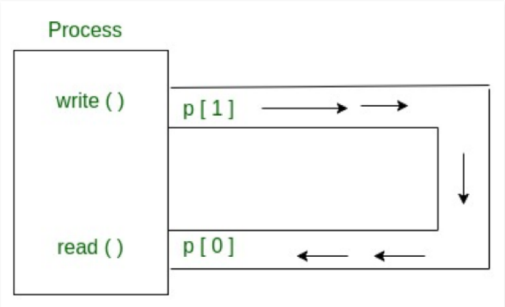
```
int pipe(int fildes[2]);
```

Parameters :

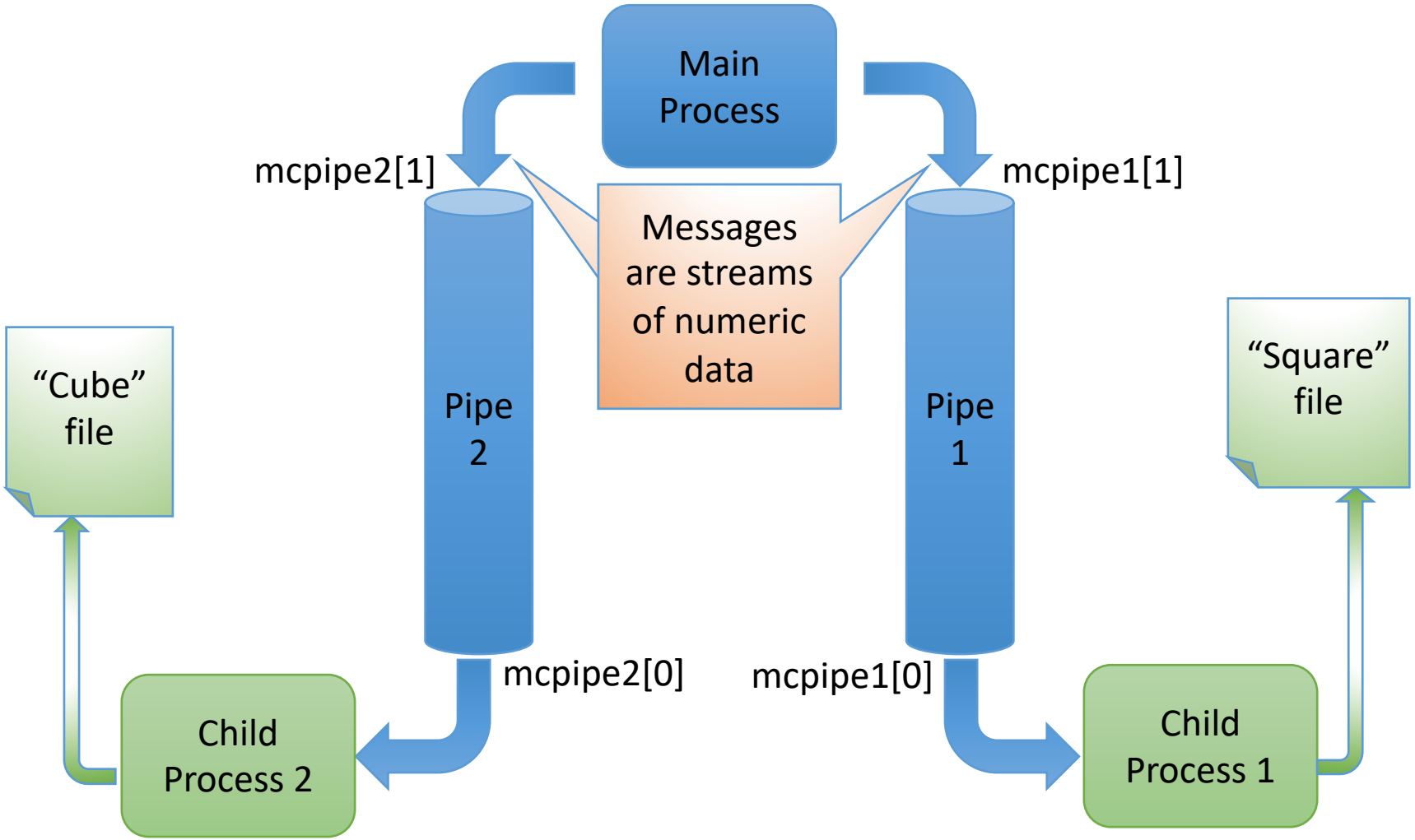
- `fd[0]` will be the fd (file descriptor) for the read end of pipe.
- `fd[1]` will be the fd for the write end of pipe.
- Returns :

  0 on Success.

  - 1 on Error.

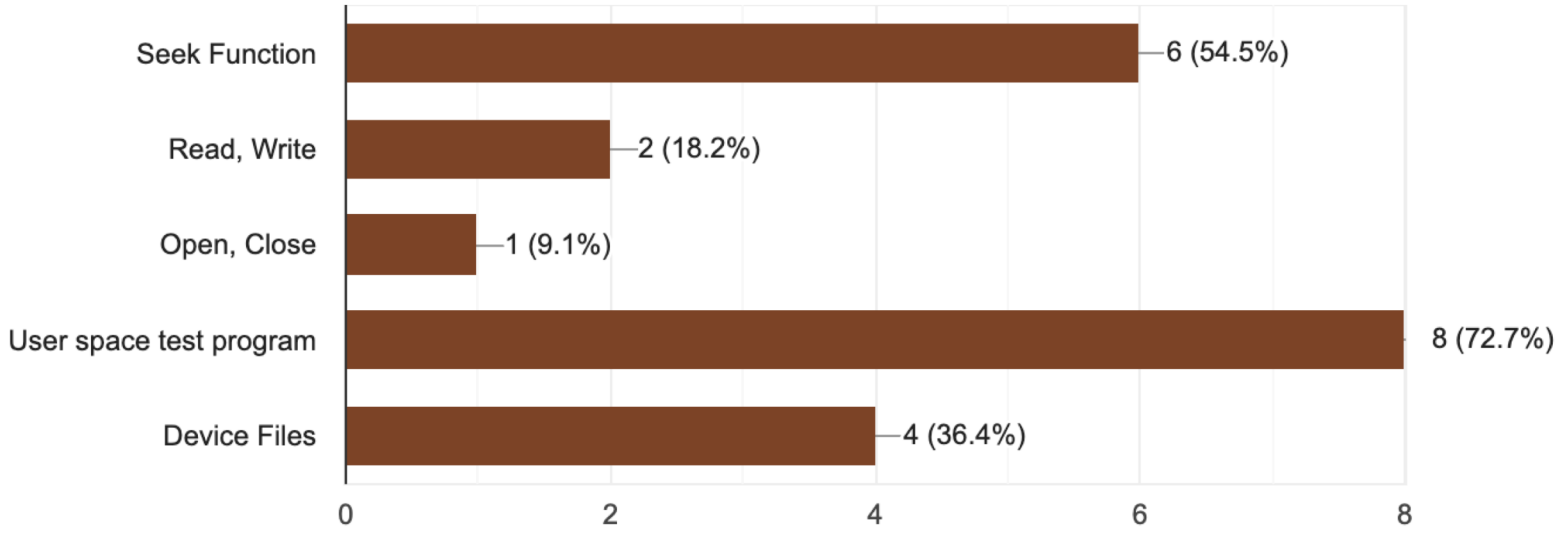


Pipe Example (Code Demo)



Programming Assignment 2

Feedback form



Test Program

1. Use the standard open, close, write, read functions, not the ones specified in the device driver. Defined in **<unistd.h>.**
2. Open the device file. All operations to be performed on it. Return will be a file descriptor (fd).
3. Infinite loop until exit is encountered.
    1. Should be passing in fd to each of your functions
    2. Seek -> Offset, whence
    3. Read -> Number of bytes to read
    4. Write -> Enter the data you want to write
4. Be sure to close your device file before exiting

File Operations

```
struct file_operations simple_char_driver_file_o
perations = {
   .owner = THIS_MODULE,
    .write = *your_write*
};
```

```
struct file_operations {
        struct module *owner;
        loff_t (*llseek) (struct file *, loff_t, int);
        ssize_t (*read) (struct file *, char __user *, size_t, loff_t *);
        ssize_t (*write) (struct file *, const char __user *, size_t, loff_t *);
        ssize_t (*read_iter) (struct kiocb *, struct iov_iter *);
        ssize_t (*write_iter) (struct kiocb *, struct iov_iter *);
        int (*iterate) (struct file *, struct dir_context *);
        int (*iterate_shared) (struct file *, struct dir_context *);
        unsigned int (*poll) (struct file *, struct poll_table_struct *);
        long (*unlocked_ioctl) (struct file *, unsigned int, unsigned long);
        long (*compat_ioctl) (struct file *, unsigned int, unsigned long);
        int (*mmap) (struct file *, struct vm_area_struct *);
        unsigned long mmap_supported_flags;
        int (*open) (struct inode *, struct file *);
        int (*flush) (struct file *, fl_owner_t id);
        int (*release) (struct inode *, struct file *);
        int (*fsync) (struct file *, loff_t, loff_t, int datasync);
        int (*fasync) (int, struct file *, int);
```

Seek function

- Your job is to compute a new position in the file and set it, let's call it `new_position`
- What will the type be?

```
loff_t simple_char_driver_seek(struct file
*pfile, loff_t offset, int whence)
```

- What will `new_position` be
  - Whence = SEEK_SET => `new_position = offset`
  - Whence = SEEK_CUR => `new_position = curr_position + offset`
  - Whence = SEEK_END => `new_position = file_size – offset`
- Set file position to equal new_position
- Check how to access current pointer position in the file.

Week 5 – Checklist

- ❑ Discuss IPC
- ❑ Read more about IPC
- ❑ PA2 Help Session