CSCI 2410 Systems Programming

Fall 2026

Instructor: Jie Zhou

Department of Computer Science

George Washington University



Slides materials are mostly credited to Gabriel Parmer and Sibin Mohan at GWU.

Process

- A running instance of a program, together with the state and resources needed for its execution.
  - has its own virtual address space
    - code, global/stack/heap data are exclusively owned by this process
  - has execution state such as registers and program counter
  - owns/uses resources such as open files, identified by integer descriptors

Process

- A running instance of a program, together with the state and **resources** needed for its execution.
  - has its own virtual address space
    - code, global/stack/heap data are exclusively owned by this process
  - has execution state such as registers and program counter
  - owns/uses resources such as open files, identified by integer descriptors

Process Resources



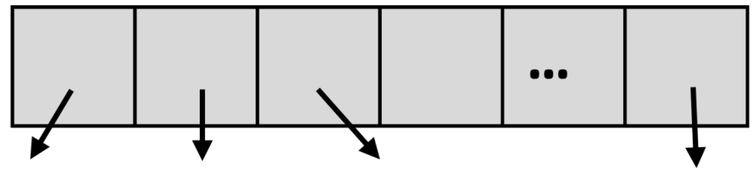
What are resources?

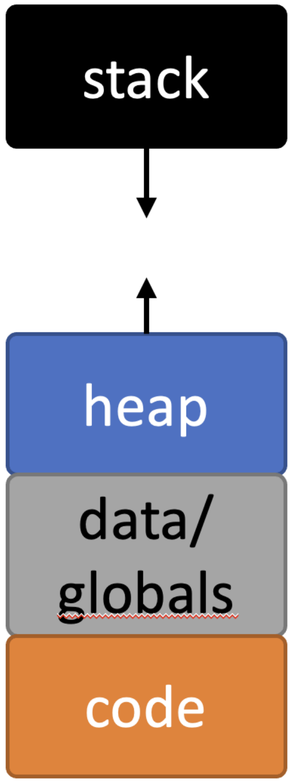
Process Resources

pid
ppid
user: sibin
cwd: /home/sibin

command arguments
environment variables

descriptors





Process Resources



What are resources?

- Three main types of I/O resources:

Resources	Purpose
Regular files	Store and retrieve
Pipes	Transfer data between processes on the same machine
Sockets	Communication between processes locally or across network

Pipes in Terminal

```
ls | wc -l
```

- Count the number of entries listed by `ls`

Unix Pipes



A one-way communication channel that carries a stream of bytes between processes.

- The kernel temporarily stores unread bytes in a buffer with limited capacity.
- Bytes are read in the same order they were written (FIFO).



- A pipe is accessed using two file descriptors.



Pipes API

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

- Including from <unistd.h>
- Creates a kernel pipe and fills the array with two file descriptors
- `fildes[2]`: `fildes[0]` for read and `fildes[1]` for write
- Return 0 on success, 1 on failure.
- After `fork()`, both parent and child inherit both descriptors.
- Use `write()` & `read()` syscalls to send & receive data to/from a pipe
  - `read()` blocks is the pipe is empty but some write exists.
  - `write()` blocks if the buffer is full.

```
write(fd[1], buf, n);   // put bytes into pipe
read(fd[0], buf, n);    // take bytes out
```