Operating Systems

Review

ENCE 360

High-level Concepts

- What are three conceptual pieces fundamental to operating systems?

High-level Concepts

- What are three conceptual pieces fundamental to operating systems?

1. Virtualization – sharing computer hardware in time and space
2. Concurrency – simultaneous access to shared resources
3. Persistence – making information exist across power outages, crashes, etc.

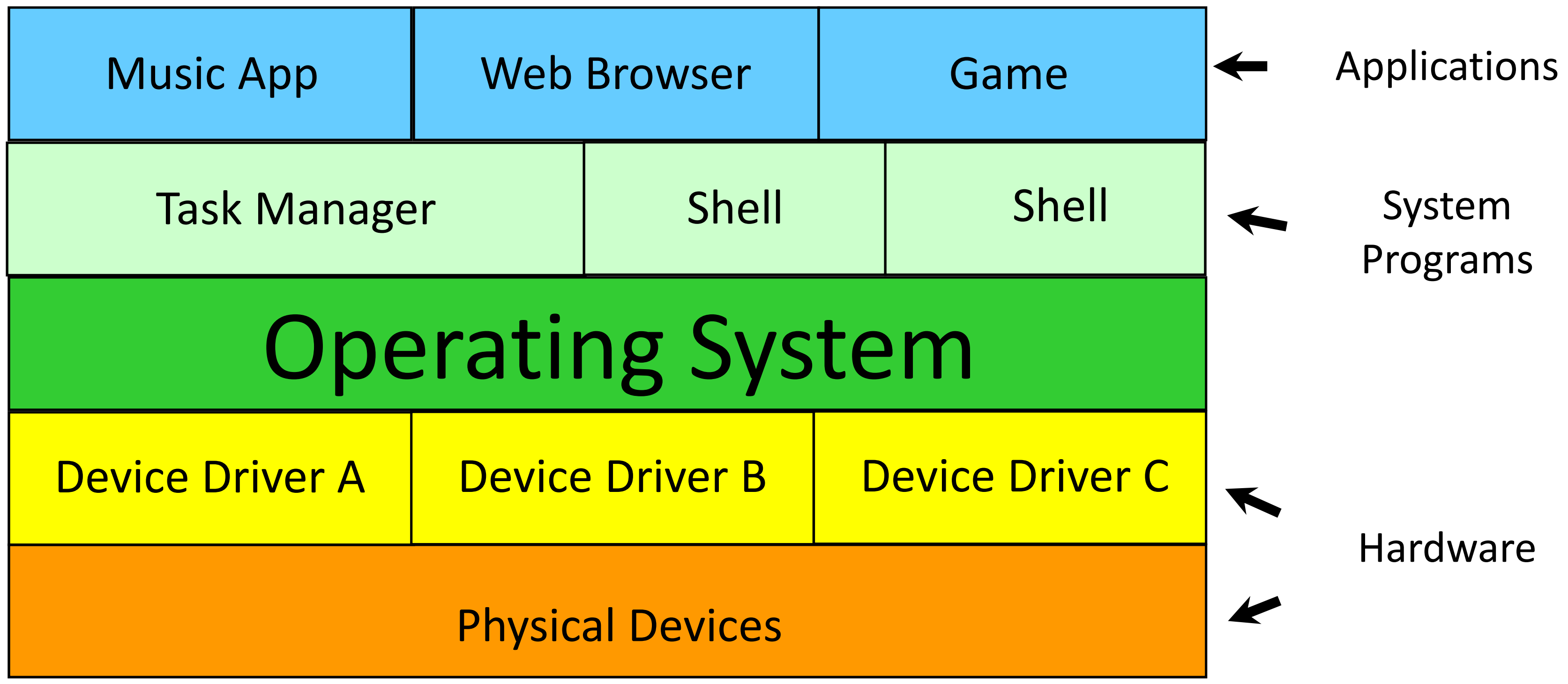
Operating System Model

- Arrange layers in order, top (user) to bottom

A. Device driver (e.g., mouse)
B. Computer game (e.g., FIFA 2018)
C. Shell (e.g., Bash)
D. Physical devices (e.g., Hard disk)
E. Operating System (e.g., Linux)
F. Program control (e.g., Task Manager)

Operating System Model

- Arrange layers in order, top (user) to bottom



The Process

- What is a process?

The Process

- What is a process?

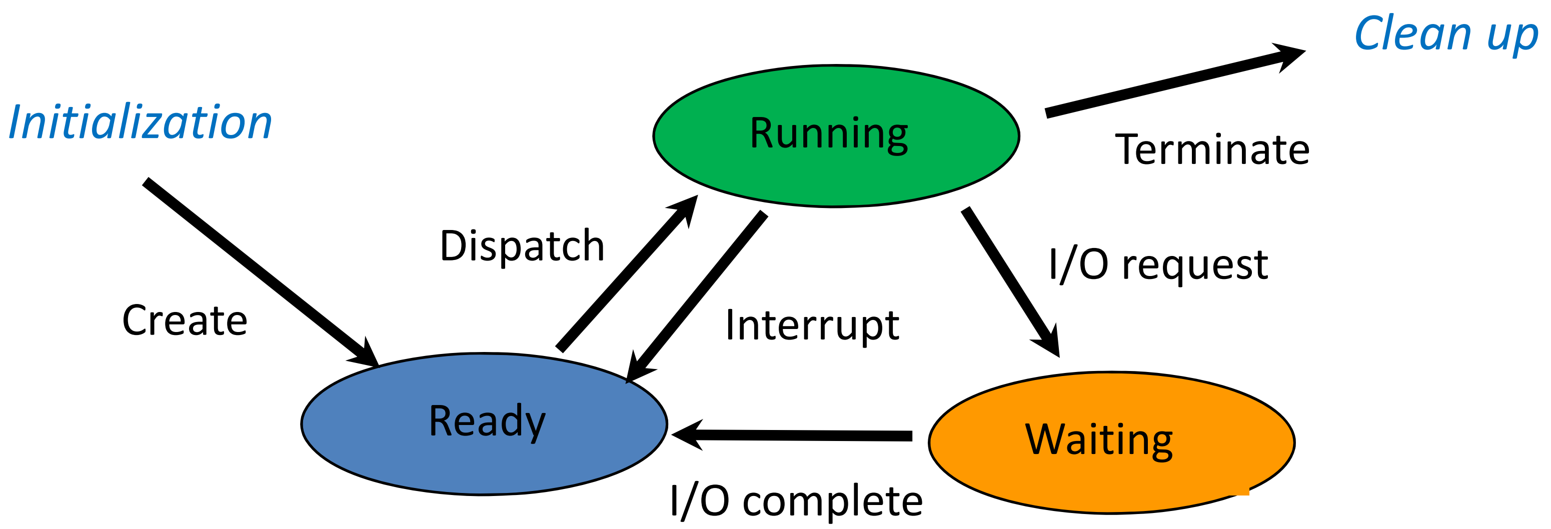
- “A program in execution”

Process States

- What are the 3 main process states?
- What are the transitions between them?

Process States

- What are the 3 main process states?
- What are the transitions between them?



Advice, help and support on campus

Medical care, counselling, travel advice, or physiotherapy.

UC Health Centre
healthcentre@canterbury.ac.nz

Upskill your academic writing and study skills.

Academic Skills Centre
academicskills@canterbury.ac.nz

Are you Māori and need advice, cultural or academic support?

Māori Student Development Team
maoridevelopment@canterbury.ac.nz

A disability or medical condition affecting your study?

Disability Resource Service
disabilities@canterbury.ac.nz

Feel more energised. Lift. Move. Play. Compete. Excel.

UC RecCentre
@UC RecCentre

UC Sport
@UC Sport

Have issues? Need help?

Students' Association (UCSA)
help@ucsa.org.nz

Need to talk things over? Practical guidance, advice and support for our domestic and international students.

Student Care
studentcare@canterbury.ac.nz

Are you Pasifika and need advice, cultural or academic support?

Pacific Development Team
pasifika@canterbury.ac.nz

Feeling unsafe or need emergency help? UC Security 0800 823 637

Check your uclive emails regularly.
You can have them forwarded to another email account—see the IT Services webpage for more info.

Process Control Block

- What is a process control block?
- What are the main components?

Process Control Block

- What is a process control block?
  - A data structure the OS uses to manage a running program (a process)

- What are the main components?
  - Running current code stuff – PC, registers, state, …
  - Memory stuff – stack, heap, code, …
  - I/O stuff – file descriptors, working directory, …

Process Creation in Unix

```
main() {
  fork();
  puts(“hello”);
}
```

- What does the code to the left do when run?
- How can we change it to only have child process print “hello”?

Process Creation in Unix

```
main() {
  fork();
  puts(“hello”);
}
```

- What does the code to the left do when run?

```
hello
hello
```

- How can we change it to only have child process print “hello”?
  - Change `fork()` line to be:

```
if (fork() == 0)
```

Processes and Threads

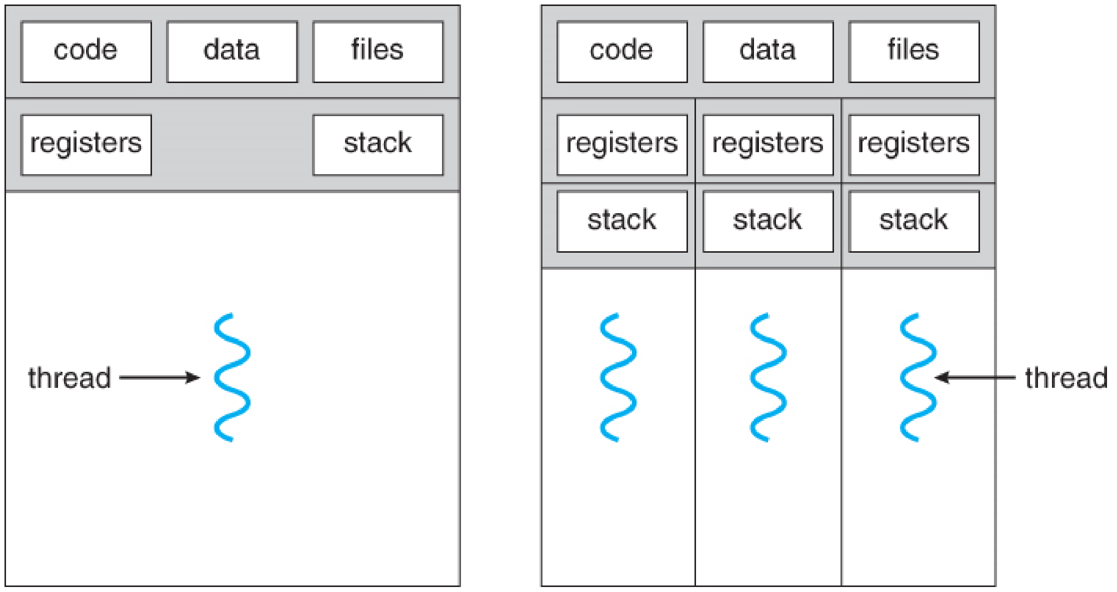
- What is a process?
- What is a thread?
- For two processes, what is private?
- For two threads in the same process, what is private?

Processes and Threads

- What is a process?
  - A program in execution / a running program
- What is a thread?
  - A single sequence of execution within a process
- For two processes, what is private?
  - Code, memory (global variables, stack), hardware state (program counter, registers), OS resources (file descriptors+)
- For two threads in the same process, what is private?
  - Memory (stack), Hardware state (program counter, registers)

Processes and Threads

- For two processes, what is private?
- For two threads in the same process, what is private?



single-threaded process

multithreaded process

(Helpful picture)

Thread Creation with Pthreads

```
void A() {
  puts(“hello”);
}

void main() {
  pthread_create(&t,A);
  puts(“goodbye”);
}
```

- What does the code to the left do when run?

Thread Creation with Pthreads

```
void A() {
  puts(“hello”);
}

void main() {
  pthread_create(&t,A);
  puts(“goodbye”);
}
```

- What does the code to the left do when run?

```
goodbye  or  hello
hello        goodbye
```

- What code to *add* to always have “hello” before “goodbye”?

Thread Creation with Pthreads

```
void A() {
  puts(“hello”);
}

void main() {
  pthread_create(&t,A);
  pthread_join(t);
  puts(“goodbye”);
}
```

- What does the code to the left do when run?

  ```
  goodbye or hello   or goodbye
  hello      goodbye
  ```

- What code to *add* to always have “hello” before “goodbye”?
  - `pthread_join(t)` before `puts(“goodbye”)`

IPC Paradigms

- What are two main paradigms for Interprocess Communication (IPC)?
- What are some advantages/disadvantages for each?

IPC Paradigms

- What are two main paradigms for Interprocess Communication (IPC)?
- What are some advantages/disadvantages for each?

  1. Message passing

  Good: explicit, less chance for programmer error

  Bad: overhead

  2. Shared memory

  Good: performance, flexibility for programmer

  Bad: changes without process knowing (side effects), programmer needs to handle sync

IPC Mechanisms

- What are some IPC mechanisms?

IPC Mechanisms

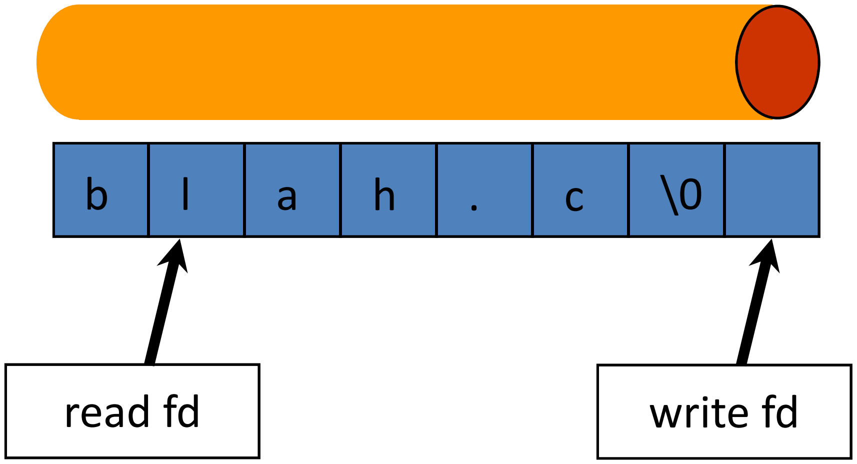
- What are some IPC mechanisms?
  - Pipe
  - Files
  - Shared memory
  - Signals
  - Sockets
  - …

Pipe

- What is a pipe? What operations does it support?

Pipe

- What is a pipe? What operations does it support?
  - IPC mechanism provided by OS
  - Gives bounded-buffer, FIFO/queue access
  - Write to one end, Read from other
  - Block on full write, Block on empty read



System Exploration

- File-descriptors and `exec()`
- Signal-signal
- Thread-signal
- Challenge: once you use `dup2()` to change STDOUT, can you restore it?
  - Hint, see: https://stackoverflow.com/questions/11042218/c-restore-stdout-to-terminal

Dup2

- From the user's perspective, what does this code do?

```
fd = open("dup.txt", O_WRONLY)
dup2(fd, STDOUT_FILENO)
```

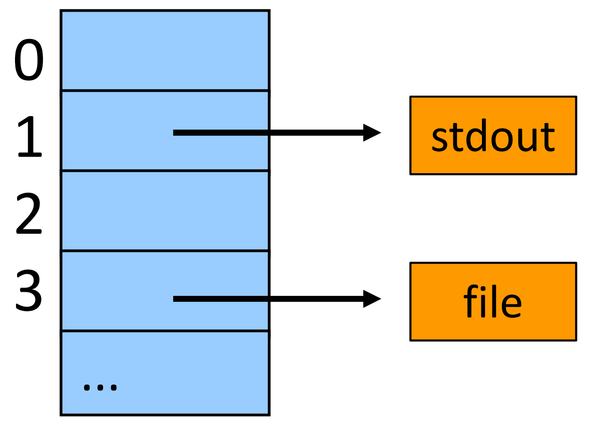
- What does it do from the system's perspective?

Dup2

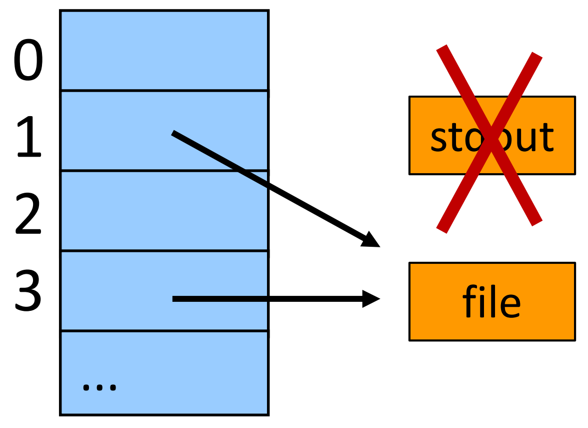
- From the user's perspective, what does this code do?

```
fd = open("dup.txt", O_WRONLY)
dup2(fd, STDOUT_FILENO)
```

  – Opens a file and changes standard output to go to the file instead of the screen
- What does it do from the system's perspective?
  – Closes STDOUT_FILENO, copies the file descriptor to the new file descriptor



Before dup2()



After dup2()

Signals

- What is an operating system signal?
- Broadly describe how to write code to use one
- Provide an example of how signals might be used

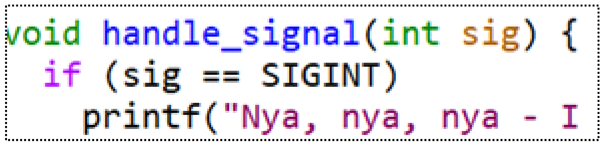
Signals

- What is an operating system signal?
  - A message to a process corresponding to an event, sent by either another process or the OS
- Broadly describe how to write code to use one

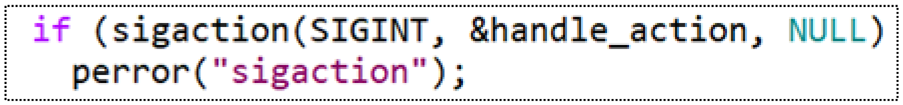
- Provide an example of how signals might be used

Signals

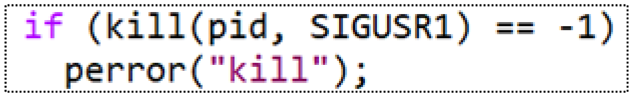
- What is an operating system signal?
  - A message to a process corresponding to an event, sent by either another process or the OS
- Broadly describe how to write code to use one
  - Write handler function
  - Make system call
  - Send signals



```
void handle_signal(int sig) {
  if (sig == SIGINT)
    printf("Nya, nya, nya - I
```



```
 if (sigaction(SIGINT, &handle_action, NULL)
   perror("sigaction");
```

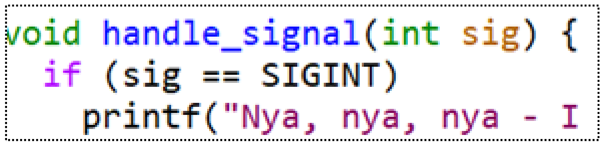


```
 if (kill(pid, SIGUSR1) == -1)
   perror("kill");
```

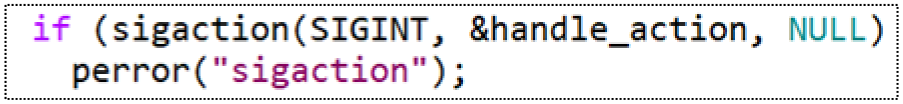
- Provide an example of how signals might be used

Signals

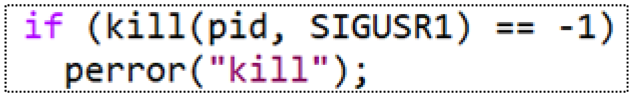
- What is an operating system signal?
  – A message to a process corresponding to an event, sent by either another process or the OS
- Broadly describe how to write code to use one
  – Write handler function
  – Make system call
  – Send signals



```
void handle_signal(int sig) {
  if (sig == SIGINT)
    printf("Nya, nya, nya - I
```



```
 if (sigaction(SIGINT, &handle_action, NULL)
   perror("sigaction");
```



```
 if (kill(pid, SIGUSR1) == -1)
   perror("kill");
```

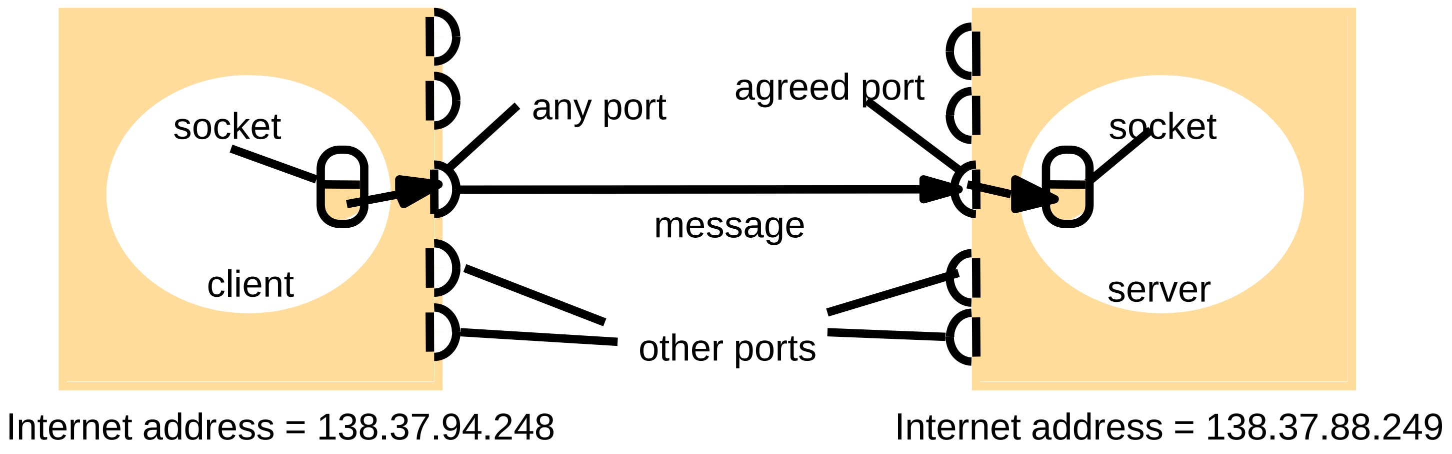
- Provide an example of how signals might be used
  – Gracefully shut down upon ctrl-c
  – Indicate to parent child has finished work
  – Wake up and do some action upon an alarm
  – …

Sockets

- What two (maybe three) pieces of information does a client need to know to connect to a server?

Sockets

- What two (maybe three) pieces of information does a client need to know to connect to a server?
  - IP Address – gets data to right computer
  - Port – gets data to right process
  - (Network protocol – TCP or UDP)



socket
client
any port
agreed port
message
other ports
socket
server
Internet address = 138.37.94.248
Internet address = 138.37.88.249

Sockets –
Put in Order for Client & Server

```
send()
connect()
bind()
close()
recv()
accept()
listen()
socket()
```

Client

Server

Sockets – Put in Order for Client & Server

`send()`
`connect()`
`bind()`
`close()`
`recv()`
`accept()`
`listen()`
`socket()`

Client	Server
`socket()`	`socket()`
`connect()`	`bind()`
`send()`	`listen()`
`recv()`	`accept()`
`close()`	`recv()`
	`close()`

Sockets - Describe

- What is the difference between
  - `send()` and `recv()`

    `vs.`
  - `read()` and `write()`

Sockets - Describe

- What is the difference between
  - `send()` and `recv()`

    `vs.`
  - `read()` and `write()`

- Answer: `send()` and `recv()` have flags that may be useful
    - `E.g., MSG_PEEK, MSG_DONTWAIT`

Sockets - Describe

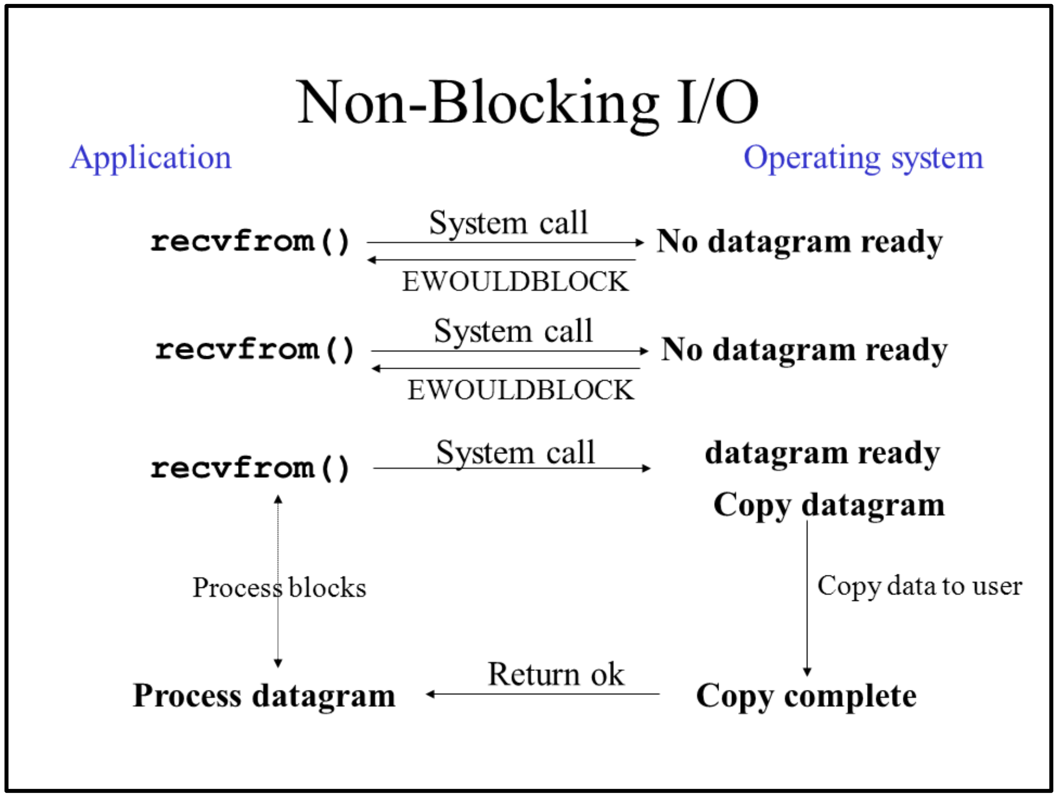
- What does “non-blocking” mean for a socket?

Sockets - Describe

- What does "non-blocking" mean for a socket?

Answer:
`recv()`/`send()`, do not sleep if no data.

Note: can be done for `accept()` too on server



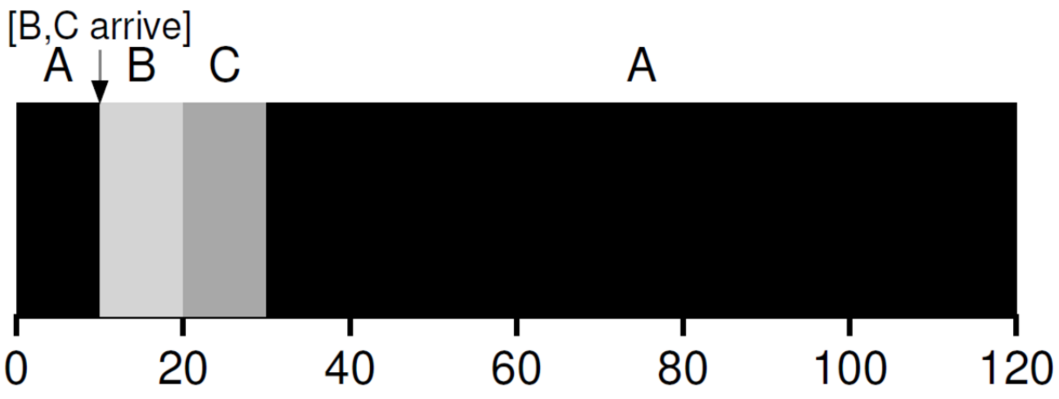
CPU Scheduling

- Briefly describe the *shortest time to completion first* (STCF) algorithm

- Is STCF pre-emptive?

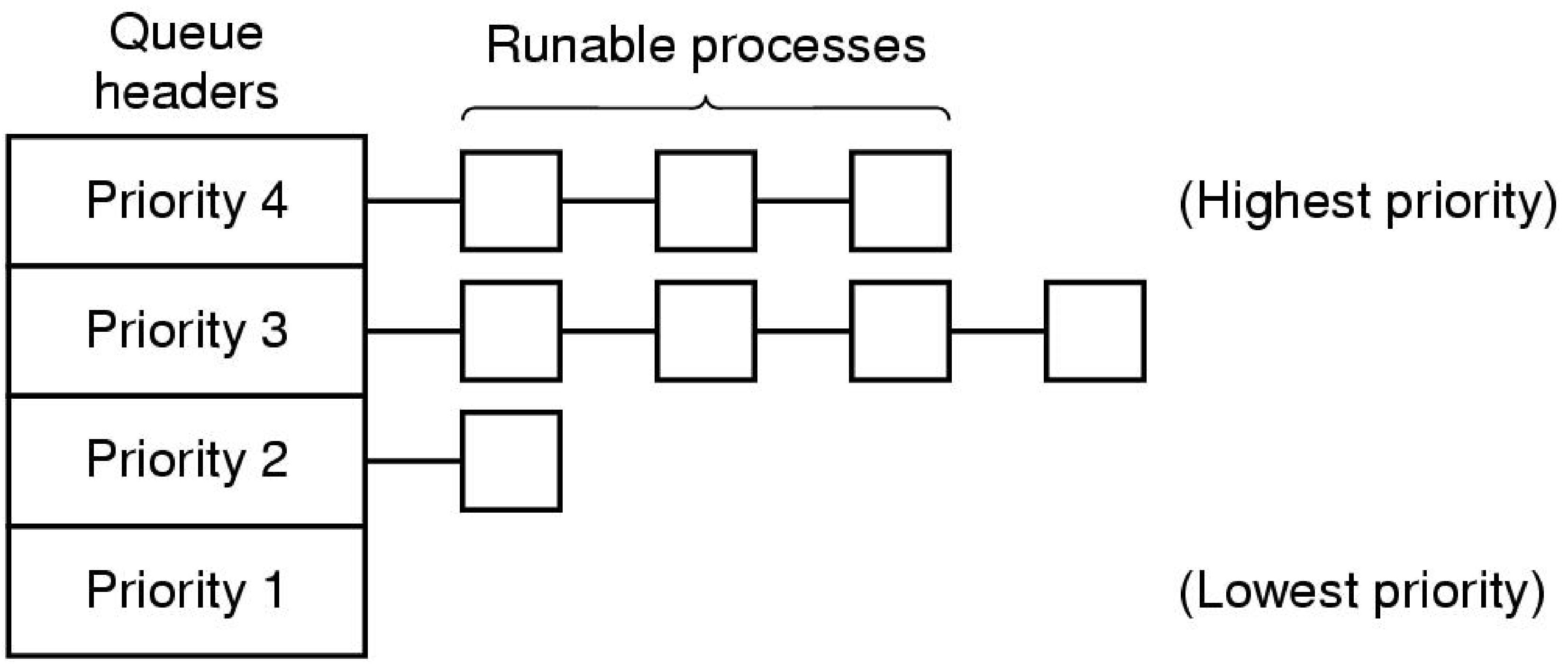
CPU Scheduling

- Briefly describe the *shortest time to completion first* (STCF) algorithm
  - From ready to run processes, select process with shortest time to finish it's CPU burst



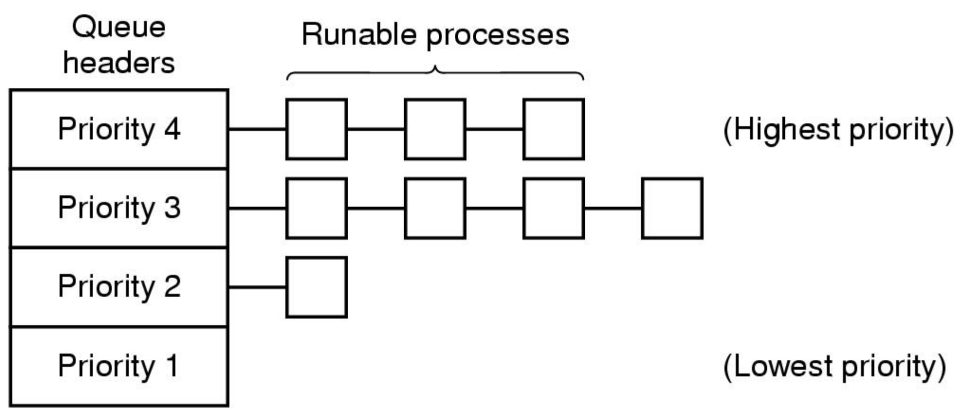
- Is SCTF pre-emptive?
  - Yes – if a process arrives that has a shorter completion time than the one currently running, it is chosen instead

CPU Scheduling



- Describe some rules that will make the MLFQ adaptive

CPU Scheduling



- Describe some rules that will make the MLFQ adaptive

1. New processes at highest priority
2. If process uses all of timeslice, reduce priority
3. If process voluntarily blocks before timeslice expires, increase priority