ECS 251

Sam King

Processes and threads: general concepts

Administrative

- HW 1 is going out today
- Quiz 1 is going to be on Thursday
  - Will cover everything in class today
- Cancelling office hours today

Project ideas

- Put the required reading up for this class (subject to change)
- Suggestion: look for other grad OS classes that are on the web that list project ideas!

More on the project

- Expect to do a lot of programming
- Pick a project specific to this class
  - Not your research
  - Not a project that you're working on in a separate class
- I'm ok with limited novelty as long as you know that it's not novel

- Start sharing ideas with Art and I now for feedback!

Administrative

- Last time – history of OS
- This time – concepts behind processes and threads
- Next time – cooperating threads

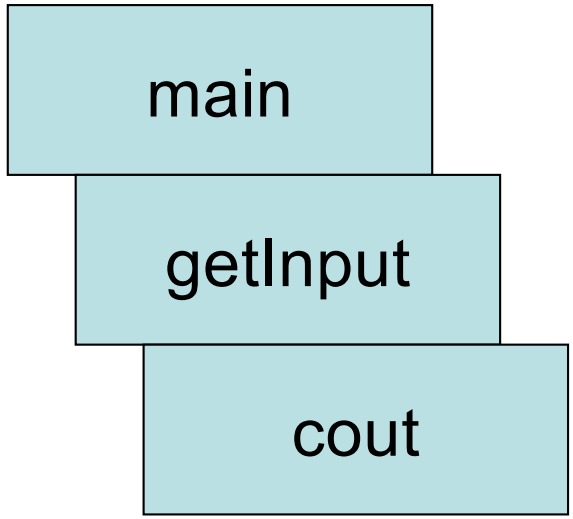
Threads and concurrency

- Motivation
  - OSes getting complex
  - Multiple users, programs, I/O devices, etc.
  - How to mange this complexity?

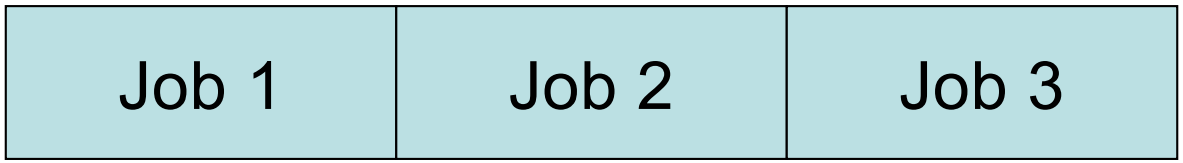
- Decompose or separate hard problems into simpler ones

Programs decompose into several rows

```
main() {
    getInput();
    computeResult();
    printOutput();
}
getInput() {
    cout();
    cin();
}
computeResult() {
    sqrt();
    pow();
}
printOutput() {
    cout();
}
```



- Processes decompose mix of activities running on a processor into several parallel tasks (columns)



- Each job can work independently of the others
- Remember, for any area of OS, ask:
  - What interface does the hardware provide?
  - What interface does the OS provide?

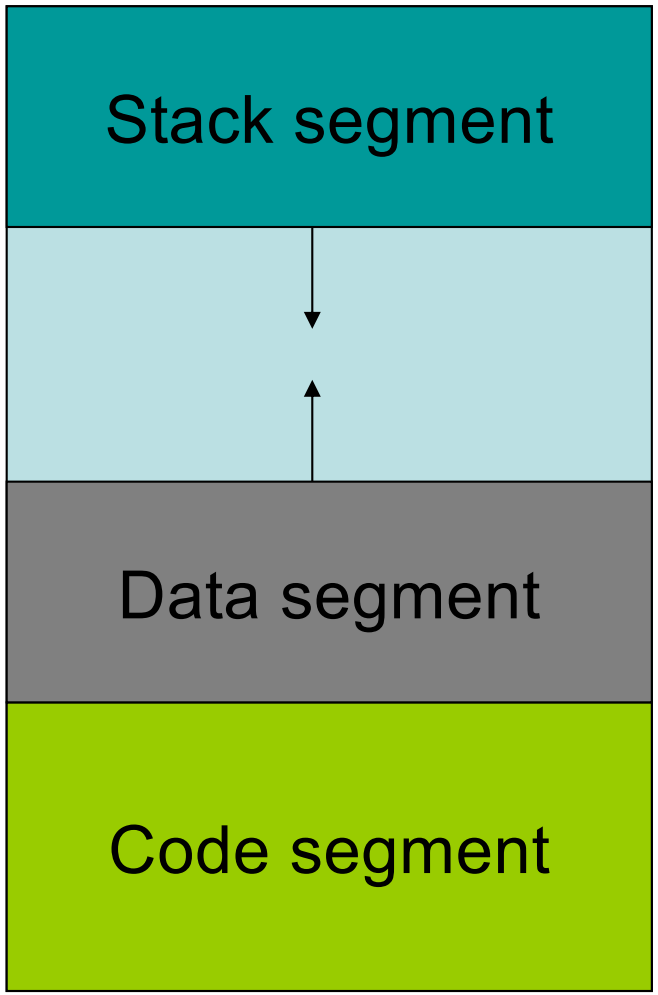
What's in a process?

- Definition of a process
  - (informal) a program in execution. A running piece of code along with all the things the program ca read/write
    - Note: process != program
  - (formal) one of more threads in their own address space
- Play analogy

- Thread
  - Sequence of executing instructions from a program (i.e., the running computation)
  - Active
  - Play analogy

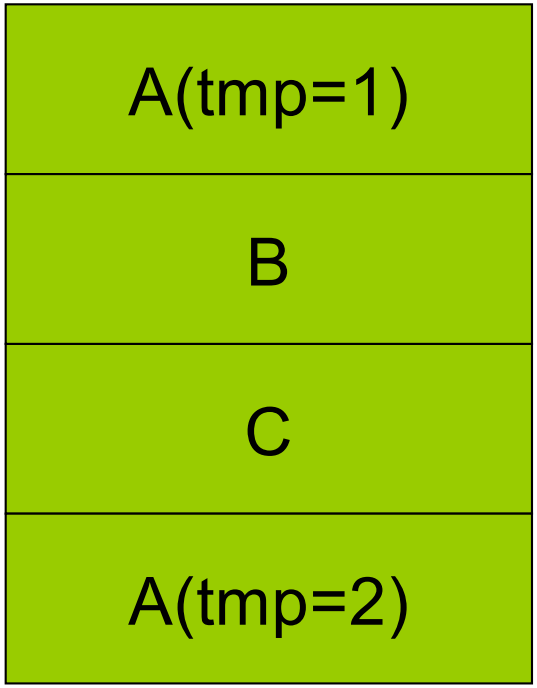
- Address space
  - All the data in the process uses as it runs
  - Passive (acted upon by the thread)
  - Play analogy: all the object on the stage in a play

Types of data in the address spaces



Stacks

```
A(int tmp) {
     B();
}
B() {
     C();
}
C() {
     A(2);
}
```



Start by calling A(1)

Multiple threads

- Can have several threads in a single address space
  - Play analogy: several actors on a single set. Sometimes interact (e.g., dance together), sometimes do independent tasks
- Private state for a thread vs. global state shared between threads

- What private state must a thread have?
  - <WRITE IN>

- Other state is shared between all threads in a process

Can threads be independent?

- Is it possible to have multiple threads on a computer system that don't cooperate or interact at all?
  - Mail program reads PDF attachment and starts acrobat to display attachment?
  - Running Halo and compiling kernel on a computer at the same time?

- Two possible sources of sharing

- Correct example of non-interacting threads

A little bit of history

- Computer systems circa 2000 were uniprocessor, I/O bound
- Web servers were the research problem of the day

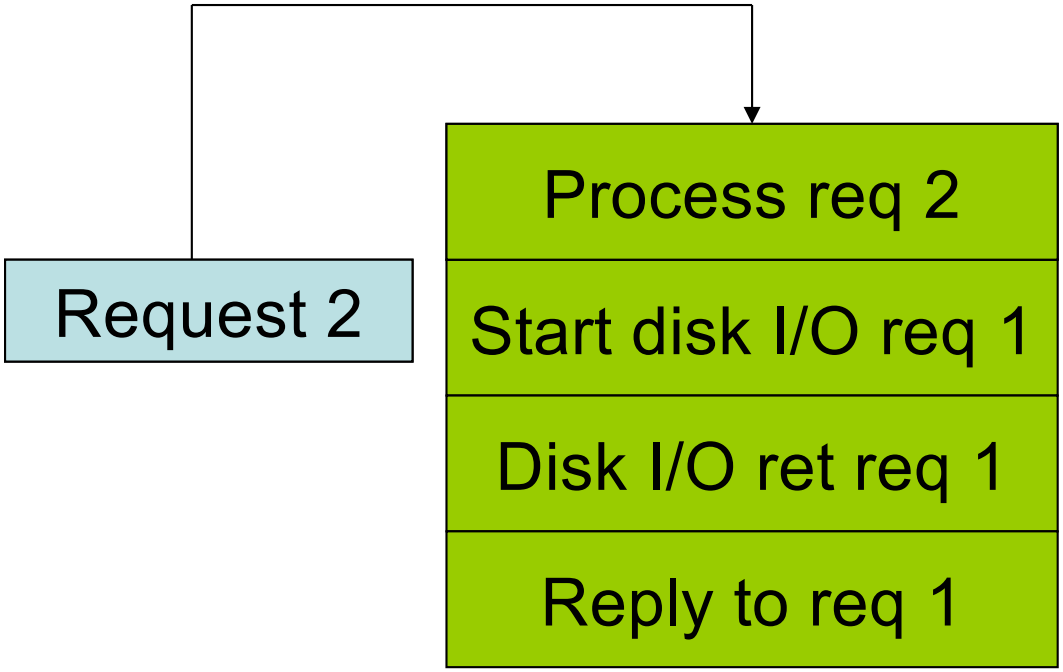
General flow for handling a web server request

```
handleWebRequest() {
  socket = newClientConnection(serverSock)
  request = readHTTPRequest(socket)
  object = accessDatabase(request)
  response = accessFilesystem(request)
  sendResponse(socket, response)
}
```

Web server example

- Web server
  - Receives multiple simultaneous requests
  - Read file from disk to satisfy request

- Handle one request at a time
  - Easy to program, slow
    - No overlapping disk requests with computation or with network receive

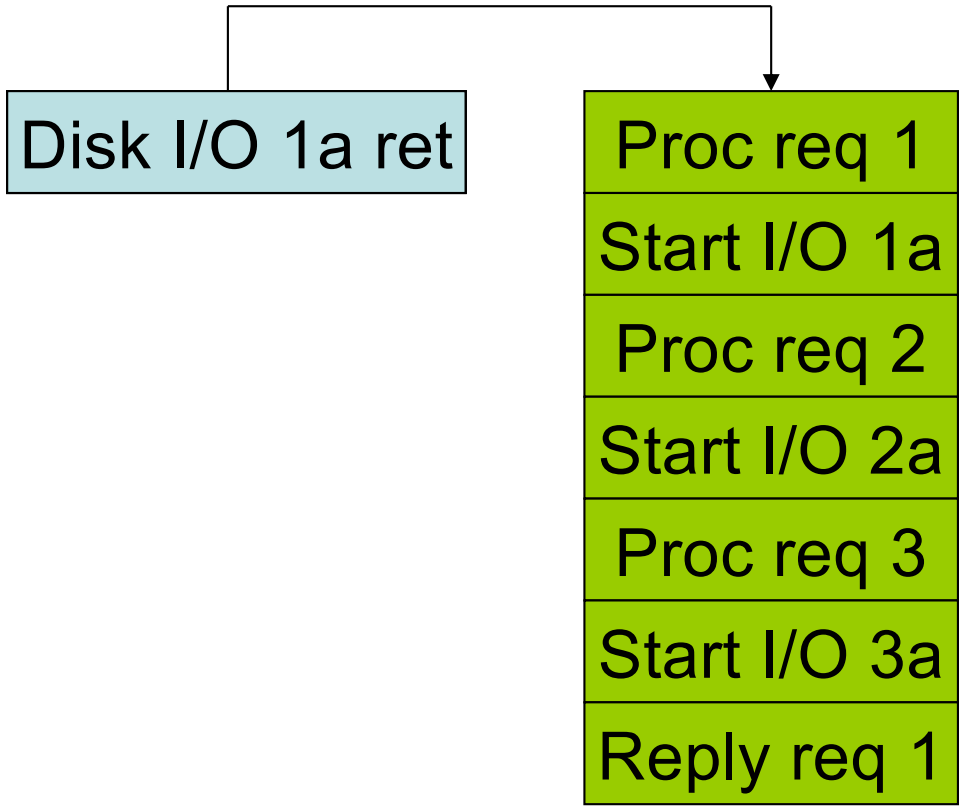


Event driven with async I/O

- Need to keep track of pending requests

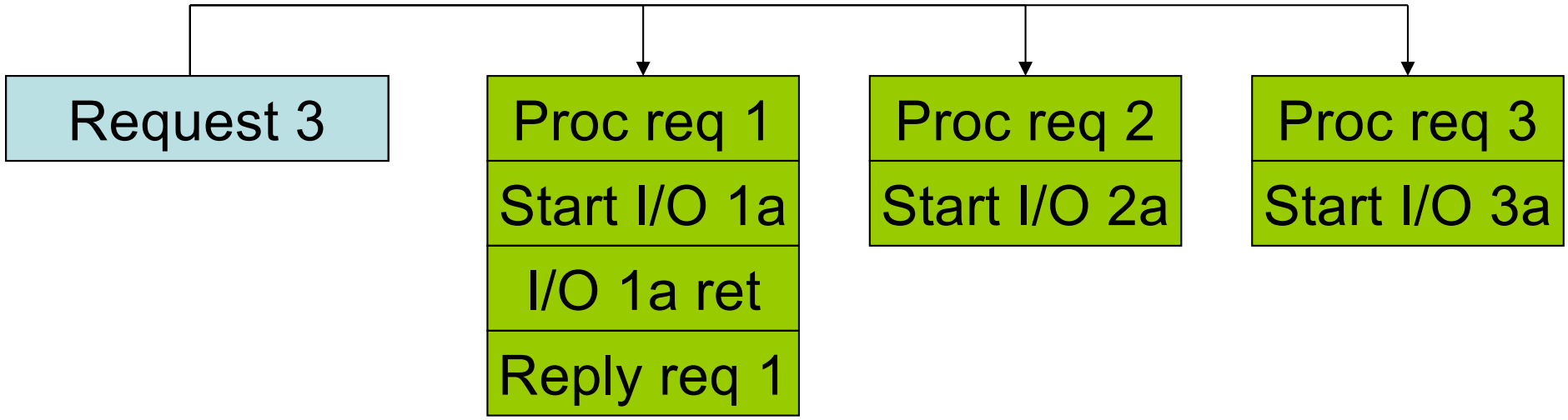
```
activeFds.add(serverSock)
handleWebRequest() {
  activeFd = select(activeFds)
  // giant state machine to track pending
  // requests (yuck)
  switch (activeFd) {
    case serverSock:
      // new request
    case isHttpRequest(activeFd):
      // read request, store state when done
    case isDatabaseFd(activeFd):
      // handle response from database
    …
  }
```

- Event-driven with async I/O
  - Need to keep track of outstanding requests



Web server using threads

- Each thread handles one request



```
handleWebRequest() {
  while (true) {
    socket = newClientConnection(serverSock)
    createNewThread(webRequestThread, socket)
  }
}

webRequestThread(socket) {
  request = readHTTPRequest(socket)
  object = accessDatabase(request)
  response = accessFilesystem(request)
  sendResponse(socket, response)
}
```

Web server

- Advantages of thread example?

- Advantages of event-driven example?

Benefits and uses of threads

- Thread system in operating system manages the sharing of the single CPU among several threads
  - Applications get a simpler programming interface

- Typical domains that use multiple threads
  - Physical control
    - Slow component?
  - Window system (1 thread per window)
  - Network server
  - Parallel programming (for using multiple CPUs)

A new baseline circa 2010

- All computers are multicore
- Many computers are power and CPU constrained (mobile phones)

- Threads are easier than old style event driven programming, still have issues
  - Race conditions on shared state
  - Atomicity violations on shared state
  - Not enough or too many threads

Too many threads: combine threads and event driven programming?

- Build a runtime layer on top of the OS
- What is the underlying interface?

- What abstraction do we want to provide to applications running above?