Operating Systems Design (CS 423)

Elsa L Gunter
2112 SC, UIUC
http://www.cs.illinois.edu/class/cs423/

Based on slides by Roy Campbell, Sam King, and Andrew S Tanenbaum

What is a process?

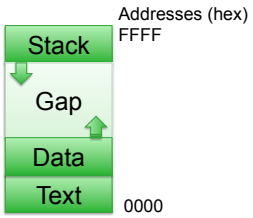
- Process definition (old view): instance of execution of a program
  - A running piece of code with all resources it can read / write
  - Note! Program =/= process
- Process definition (modern view): Full collection of threads sharing same address space
- Play analogy: Process is a performance of a play on a stage in a stage house

Process Components

- Thread
  - Sequence of executing instructions from a program (i.e., the running computation)
  - Active
  - Play analogy: actors on stage
- Address space
- All data used by process as it runs
- Passive (acted upon by the thread)
- Play analogy: stage, possibly stage props

UNIX Process Memory Layout

- Process has three segments
  - Text – lowest addresses, fixed size
  - Data – immediately after text, grows only by explicitly system call
  - Stack – top of region, grows downward automatically
  - Gap in middle for dynamic allocation



Stack

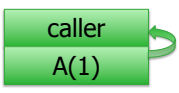
```
void A(int tmp){
if (tmp == 2) {return}
else    {B()};
}
void B(){
   C();
}
void C(){
   A(2);
}
Start: A(1);
```

- Stack

Stack

```
void A(int tmp){
if (tmp == 2) {return}
else    {B()};
}
void B(){
   C();
}
void C(){
   A(2);
}
Start: A(1);
```

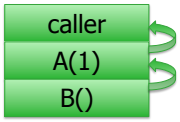
- Stack



Stack

```
void A(int tmp){
if (tmp == 2) {return}
else    {B()};
}
void B(){
   C();
}
void C(){
   A(2);
}
Start: A(1);
```

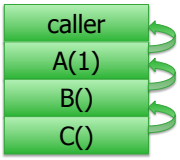
- Stack



Stack

```
void A(int tmp){
if (tmp == 2) {return}
else    {B()};
}
void B(){
   C();
}
void C(){
   A(2);
}
Start: A(1);
```

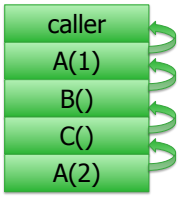
- Stack



Stack

```
void A(int tmp){
if (tmp == 2) {return}
else    {B()};
}
void B(){
   C();
}
void C(){
   A(2);
}
Start: A(1);
```

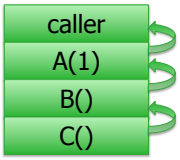
- Stack



Stack

```
void A(int tmp){
if (tmp == 2) {return}
else    {B()};
}
void B(){
   C();
}
void C(){
   A(2);
}
Start: A(1);
```

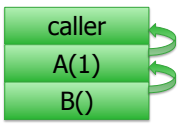
- Stack



Stack

```
void A(int tmp){
if (tmp == 2) {return}
else    {B()};
}
void B(){
   C();
}
void C(){
   A(2);
}
Start: A(1);
```

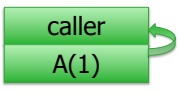
- Stack



Stack

```
void A(int tmp){
if (tmp == 2) {return}
else    {B()};
}
void B(){
   C();
}
void C(){
   A(2);
}
Start: A(1);
```

- Stack



Stack

```
void A(int tmp){
if (tmp == 2) {return}
else    {B()};
}
void B(){
   C();
}
void C(){
   A(2);
}
Start: A(1);
```

- Stack

caller

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

Thread State

- What private state must a thread have?
- Other state is shared between all threads in a process

Thread State

- What private state must a thread have?
  - Thread ID
  - Program counter, registers
  - Local data
- 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?

Types of thread interactions

- Two possible sources of sharing
  - Application dependent sharing (shared memory)
  - Resource dependent sharing (hardware, time,…)
- Correct example of non-interacting threads?
  - Two threads on two different computers not connected directly or to the web …

SVN server example

- If threads cooperating, is it still helpful to think of multiple threads? Or is it simpler to think of a single thread doing multiple things?
- SVN server
  - Receives multiple simultaneous requests
  - Read file from disk to satisfy request

SVN server example

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

SVN server example

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

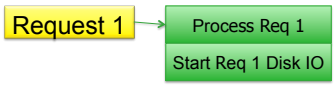
Request 1

SVN server example

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

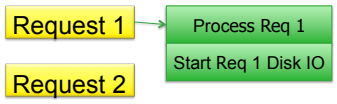
SVN server example

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



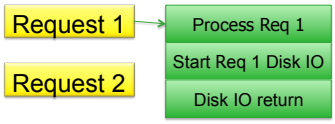
SVN server example

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



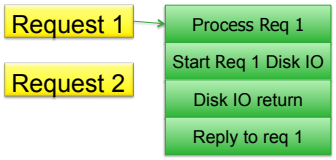
SVN server example

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



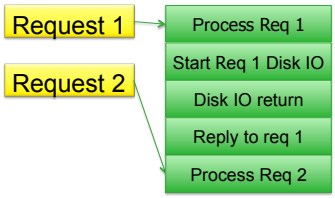
SVN server example

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



SVN server example

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



SVN server example

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

SVN server example

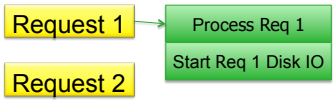
- Event-driven with async I/O
  - Need to keep track of outstanding requests

SVN server example

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

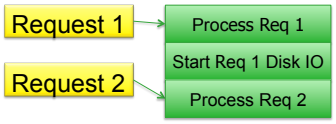
SVN server example

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



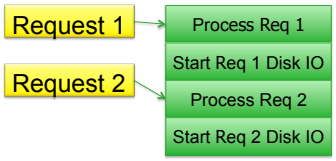
SVN server example

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



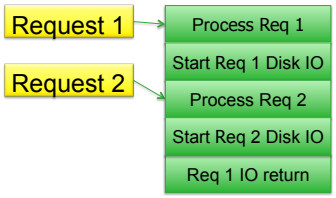
SVN server example

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



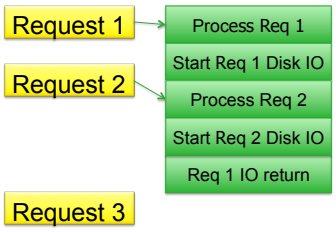
SVN server example

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



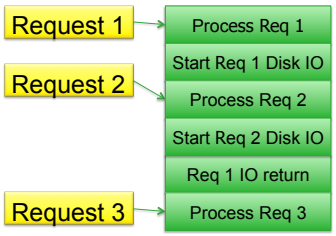
SVN server example

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



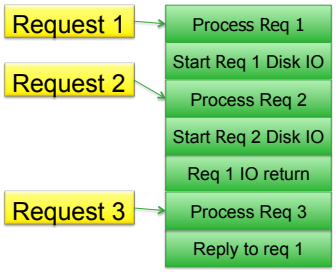
SVN server example

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



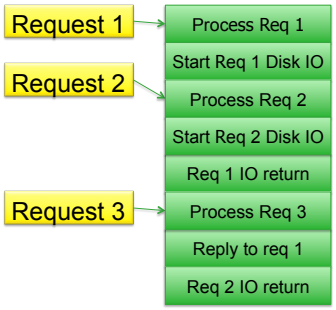
SVN server example

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



SVN server example

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



SVN server example

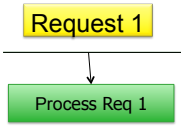
- SVN server using threads
  - Each thread handles one request

SVN server example

- SVN server using threads
  - Each thread handles one request

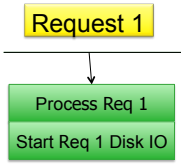
SVN server example

- SVN server using threads
  - Each thread handles one request



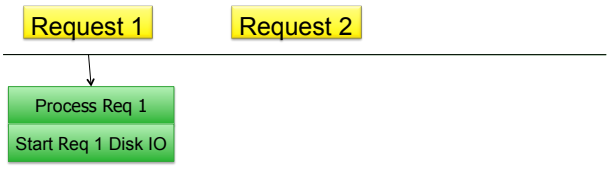
SVN server example

- SVN server using threads
  - Each thread handles one request



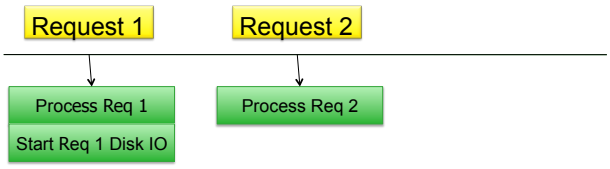
SVN server example

- SVN server using threads
  - Each thread handles one request



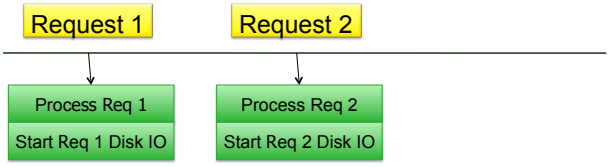
SVN server example

- SVN server using threads
  - Each thread handles one request



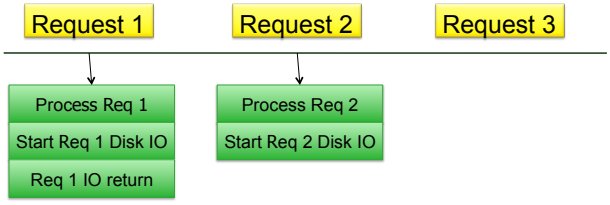
SVN server example

- SVN server using threads
  - Each thread handles one request



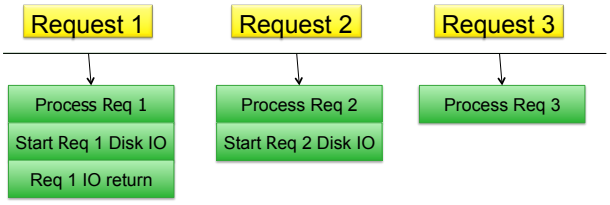
SVN server example

- SVN server using threads
  - Each thread handles one request



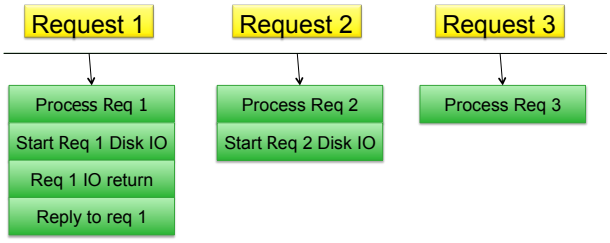
SVN server example

- SVN server using threads
  - Each thread handles one request



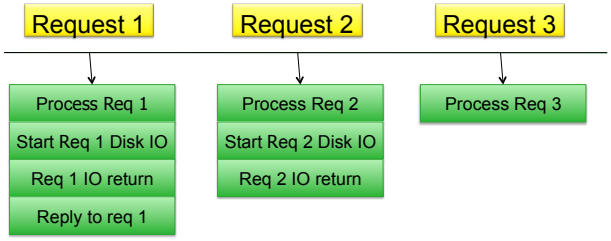
SVN server example

- SVN server using threads
  - Each thread handles one request



SVN server example

- SVN server using threads
  - Each thread handles one request



SVN server example

- Where is the state of each request stored?
  - Event-driven:
  - Thread-based: In the operating system
- Advantages of event-driven example?

SVN server example

- Where is the state of each request stored?
  - Event-driven: In the user process state, must be managed by user routine
  - Thread-based: In the operating system
- Advantages of event-driven example?

SVN server example

- Where is the state of each request stored?
  - Event-driven: In the user process state, must be managed by user routine
  - Thread-based: In the operating system
- Advantages of event-driven example?
  - More efficient

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)

Cooperating threads

- How multiple threads can cooperate on a single task
  - Assume for now that we have enough physical processors for each thread
  - Later we'll discuss how to give illusion of infinite processors on single processor

Ordering of events

- Ordering of events from different threads is non-deterministic
  - Processor speeds may vary
  - E.g., after 10 seconds, different thread have different amounts of work done

Thread A ---------->
Thread B - - - ->
Thread C - - - - - - - - - - - - - - - ->

Nondeterminisim

- Non deterministic ordering produces non deterministic results
- Printing example
  - Thread A: print ABC
  - Thread B: print 123
  - Possible outputs?
  - Impossible outputs?
  - Why or why not?
  - What is being shared?

Arithmetic example

- Initially y=10
- Thread A: x = y+1;
- Thread B: y = y*2;
- Possible results?

Atomic operations

- Example:
  - Thread A: x=1;
  - Thread B: x=2;
  - Possible results?
  - Is 3 a possible output?

Non-interference and Atomic operations

- Non-interference assumption:
  - Before we can reason **at all** about cooperating threads, we must know that some operations will execute to completion without interference from any other source
  - Non-interference: operation will always return result as if it were the only operation running
- Operation that in all circumstances will execute to completion is **atomic**