CS 134:
Operating Systems
Processes
Overview

Processes

Processes in Unix
Implementation
States

Threads

Concepts
Uses
Models
Design
What is a process?

What is concurrency?
A fundamental OS abstraction

- But details vary from OS to OS:
  - Batch system—Jobs
  - Time-shared systems—User programs or tasks

- Common idea: Process = “A program in execution”

- Processes have a degree of independence from each other
  - Possibly only allowed communicate through designated mechanisms
  - One errant processes should not affect other unrelated ones
What makes up a process? (“A process has...”)

- In general
- On a typical POSIX system
Components of a Process (Unix)

- Execution state
  - Registers
  - Program counter
  - Program status word
  - Stack pointer
- Scheduling information
  - Process state
  - Priority
  - Class, etc.
- Memory
  - Text area
  - Data area
  - Stack area
- Security/Authentication Info
  - User ID
  - Group ID
- I/O State
  - File descriptors
  - Working directory
  - Root directory
- Event Notifications
  - Signals waiting
  - Signal mask
  - Time of next alarm
- Other
  - Process ID
  - Parent process
  - Process group
  - Controlling terminal
  - Start time
  - CPU time
  - Children’s CPU time
Processes:

- Create with `fork`
- Exit with `exit`
- Replace “process image” with `execve`

Multiple processes may be active at any one time (compare with uniprogrammed system)
If there's fork, should we have join?
Processes under UNIX

The environment you interact with is made up of processes
How does the OS implement the process abstraction?
The OS needs to maintain a process image for each process:

- Process’s address space, containing:
  - Program code
  - Program data
  - Processor stack

- Housekeeping information (PCB)
  - One of most important is process state
A Two-State Process Model

Simplest model for processes:
More useful model for processes:

A Four-State Process Model

- Ready
- Running
- Finished
- Blocked
A Five-State Process Model

Five states can model additional needs of batch systems:

- **New**
- **Ready**
- **Running**
- **Blocked**
- **Finished**

Scheduler queues:
- **Ready queue**: Processes ready and waiting to execute.
- **New queue**: Processes waiting to be created
Generalizing Processes

Simple view of process is
  Address space
  + Thread of execution

Does the mapping need to be one-to-one?
Motivation:

- Traditional processes: Virtual uniprocessor machine
- Multithreaded processes: Virtual multiprocessor machine
Various reasons why people use threads

- Performing foreground and background work
- Supporting asynchronous processing
- Speeding execution
- Organizing programs
/* Dispatcher Thread */
for ( ; ; ) {
    url = get_next_request();
    handoff_work(url);
}

/* Worker Thread */
for ( ; ; ) {
    url = wait_for_work();
    page = look_in_cache(url);
    if (page == NULL)
        page = generate_page(url);
    send_page(page);
}
Can an application implement threads without built-in thread support in the OS?

If so, what *does* it need from the OS to support threads?
Model for User Threads

Class Exercise
What are the pros and cons of this approach?

So, maybe we should put the threads in the kernel?
Model for User Threads

Key:

- User-level thread
- Kernel-level thread
- Process

+ No kernel overhead for thread library calls
+ Own scheduler = Application-specific scheduling policy?
- I/O issues
- Can’t (easily) take advantage of multiprocessing

So, maybe we should put the threads in the kernel?
Model for Kernel-Level Threads

Class Exercise
What are the pros and cons of this approach?
Now we have kernel overheads:

- Kernel data structures
- Mode switch to kernel
Hybrid Thread Schemes

Class Exercise
What are the pros and cons of this approach?
Traditional vs. Multithreaded Processes

Single-Threaded Process Model

- Process Control Block
- User Stack
- User Address Space
- Kernel Stack

Multithreaded Process Model

- Process Control Block
- User Stack
- Kernel Stack

Class Question: But what's per-process and what's per-thread?
Traditional vs. Multithreaded Processes

Class Question
But what's per-process and what's per-thread?
Per-Process vs. Per-Thread—You Decide...

- Execution state
  - Registers
  - Program counter
  - Program status word
  - Stack pointer
- Scheduling information
  - Process state
  - Priority
  - Class, etc.
- Memory
  - Text area
  - Data area
  - Stack area
- Security/Authentication Info
  - User ID
  - Group ID

- I/O State
  - File descriptors
  - Working directory
  - Root directory
- Event Notifications
  - Signals waiting
  - Signal mask
  - Time of next alarm
- Other
  - Process ID
  - Parent process
  - Process group
  - Controlling terminal
  - Start time
  - CPU time
  - Children's CPU time