Frequently asked questions from the previous class survey

- Pipes:
  - Why do parent and child close one-end of the pipe?
  - With `ls | more` is the parent "ls"?
- Microkernels vs regular (monolithic) kernels
  - Pros of microkernels?
- IPC: Message passing
  - What if a sender is generating messages at too high a rate?
  - Why not use shared memory for small and large messages?
- Endian representations: Is one better?

Topics covered in this lecture

- Background
- Rationale for threads
- Thread model
- Benefits of multithreaded programming
- User- and kernel-level threads

Some background on threading

- Exploited to make programs easier to write
  - Split programs into separate tasks
- Took off when GUIs became standard
  - User perceives better performance
    - Programs did not run faster; this was an illusion
    - Dedicated thread to service input or display output
- Growing trend to exploit available processors on a machine

What are threads?

- Miniprocesses or lightweight processes
- Why would anyone want to have a kind of process within a process?
The main reason for using threads
- In many applications, multiple activities are going on at once.
- Some of these may block from time to time.
- Decompose application into multiple sequential threads.
  - Running in quasi-parallel.

Isn't this precisely the argument for processes?
- Yes, but there is a new dimension...
- Threads have the ability to share the address space (and all of its data) among themselves.
- For several applications:
  - Processes (with their separate address spaces) don't work.

Threads are also lighter weight than processes
- Faster to create and destroy than processes.
- In many systems, thread creation is 10-100 times faster.
- When the number of threads that are needed changes dynamically and rapidly?
  - Lightweight property is very useful.

Threads: The performance argument
- When all threads are CPU bound all the time?
  - Threads yield no performance gain.
- But when there is substantial computing and substantial I/O:
  - Having threads allows activities to overlap.
  - Speeds up the application.

Our Word Processor
- Displays document being created on the screen.
- Document formatted exactly as it will appear on a printed page.
Let's take a look at someone editing a 800-page document

- User deletes one sentence from Page-1 of an 800-page document
- Now user wants to make a change on page 600
  - Either go to that page or search for term that only appears there

Page 600 after the edit on Page 1

- Word processor does not know what's the first line on page 600
- Word processor has to reformat entire book up to page 600
- Threads could help here ...

Suppose the word processor is written as a 2-threaded program

- One thread interacts with the user
- The second thread handles formatting in the background
- As soon as the sentence is deleted
  - Interactive thread tells formatter thread to format the book

While we are at it, why not add a third thread?

- Automatically save file every few minutes
- Handle disk backups without interfering with the other 2 threads

What if the program were single threaded?

- Whenever disk backup started
  - Commands from keyboard/mouse would be ignored till backup was finished
  - User perceives sluggish performance
- Alternatively, keyboard/mouse events could interrupt the disk backup
  - Good performance
  - Complex, interrupt-driven programming

With 3 threads the programming model is simpler

- First thread interacts with the user
- Second thread reformats when told to
- Third thread writes contents of RAM on to disk periodically
Three separate processes WOULD NOT work here

- All three threads need to operate on document
- By having 3 threads instead of 3 processes
  1. The threads share a common memory
  2. Have access to document being edited

Applications are typically implemented as a process with multiple threads of control

- Perform different tasks in the application
  - Web browser
    - Thread A: Render images and text
    - Thread B: Fetch network data
  - Assist in the performance of several similar tasks
    - Web Server: Manages requests for web content
      - Single threaded model: One client at a time
      - Poor response times
      - Multithreaded model: Multiple clients served concurrently

The process model is based on two independent concepts

- Resource grouping
- Execution

A process can be thought of as a way to group related resources together

- Address space containing program text and data
- Other resources
  - Open files, child processes, signal handlers, etc.

A process also has a thread-of-execution

- Usually shortened to just thread

  - The thread has
    1. Program counter
    2. Registers: Current working variables
    3. Stack: Contains execution history
      - One frame for each procedure called, but not returned from
Although a thread must execute in some process

- The process and thread are different concepts
- Can be treated separately
- Processes are used to group resources together
- Threads are entities scheduled for execution on the CPU

Threads & Processes

- Threads extend the process model by allowing multiple executions in the same process
- Multiple threads in parallel in one process?
  - Analogous to multiple processes running in parallel on one computer

Threads and Processes

- Threads extend the process model by allowing multiple executions in the same process
- Multiple threads in parallel in one process?
  - Analogous to multiple processes running in parallel on one computer

Different threads in a process are NOT as independent as different processes

- All threads within a process have the same address space
  - Share the same global variables
- Every thread can access every memory address within the process’ address space
  - Read
  - Write
  - Wipe out another thread’s stack

There is no protection between threads, because …

1. It is impossible
2. It should not be necessary

Unlike processes which may be from different users

- A process is always owned by a single user
- The user created threads so that they can cooperate … not fight
Contrasting items unique & shared across threads

<table>
<thead>
<tr>
<th>Per process items (Shared by threads within a process)</th>
<th>Per thread items (Items unique to a thread)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Address space</td>
<td>Program Counter</td>
</tr>
<tr>
<td>Global variables</td>
<td>Registers</td>
</tr>
<tr>
<td>Open files</td>
<td>Stack</td>
</tr>
<tr>
<td>Child Processes</td>
<td>State</td>
</tr>
<tr>
<td>Pending alarms</td>
<td></td>
</tr>
<tr>
<td>Signals and signal handlers</td>
<td></td>
</tr>
<tr>
<td>Accounting Information</td>
<td></td>
</tr>
</tbody>
</table>

A thread is a basic unit of CPU utilization

- Thread ID
- Program Counter
- Register Set
- Stack
- State

Sharing among threads belonging to a given process

- Code section
- Data section
- OS resources
- Open files
- Signals

A process with multiple threads of control can perform more than 1 task at a time

<table>
<thead>
<tr>
<th>CODE</th>
<th>DATA</th>
<th>FILES</th>
</tr>
</thead>
<tbody>
<tr>
<td>Registers</td>
<td>Stack</td>
<td></td>
</tr>
<tr>
<td>CODE</td>
<td>DATA</td>
<td>FILES</td>
</tr>
<tr>
<td>Registers</td>
<td>Stack</td>
<td>Registers</td>
</tr>
</tbody>
</table>

Traditional Heavy weight process

- Process with multiple threads

Why each thread needs its own stack (1)

- Stack contains one frame for each procedure called but not returned from
- Frame contains
  - Local variables
  - Procedure’s return address

Why each thread needs its own stack (2)

- Procedure X calls procedure Y, Y then calls Z
- When Z is executing?
  - Frames for X, Y and Z will be on the stack
- Each thread calls different procedures
- So has a different execution history
Each thread has its own stack

Thread states are similar to processes
- Running
- Blocked
- Ready
- Terminated

The rationale for threads
- Process creation is time consuming, resource intensive
- If new process performs same tasks as existing process, why incur this overhead?
- Much more efficient to use multiple threads in the process

Benefits of multithreaded programming
- Responsiveness
- Resource Sharing
- Economy
- Scalability
Multithreaded programming: Benefit #1
Responsiveness
- Shifting work to run in the background
- Interactive multithreaded application
  - Parts of program may be blocked or slow
  - Remainder of program may still chug along
- E.g., Web browser
  - You may read text, while high-resolution image is being downloaded

Multithreaded programming: Benefit #2
Resource Sharing
- Programmer arranges sharing between processes
  - Shared memory & message passing
- Threads within a process share its resources
  - Memory, code, and data
  - Allows several different threads of activity within the same process

Multithreaded programming: Benefit #3
Economy
- Process creation is memory and resource intensive
- Threads share process’ resources
  - Economical to create and context-switch threads

Multithreaded programming: Benefit #4
Scalability
- A single threaded process can ONLY run on 1 processor
  - Regardless of how many are available
  - Underutilization of compute resource
- Programs can use threads on a multiprocessor to do work in parallel
  - Do the same work in less time OR
  - Do more work in the same elapsed time

Comparing thread executions on single core and dual core systems

<table>
<thead>
<tr>
<th></th>
<th>T1</th>
<th>T2</th>
<th>T3</th>
<th>T4</th>
<th>T2</th>
<th>T1</th>
<th>T4</th>
<th>T3</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Time</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Core 1</strong></td>
<td>T1</td>
<td>T2</td>
<td>T1</td>
<td>T2</td>
<td></td>
<td>T1</td>
<td>T2</td>
<td></td>
</tr>
<tr>
<td><strong>Core 2</strong></td>
<td>T3</td>
<td>T4</td>
<td>T4</td>
<td>T3</td>
<td></td>
<td>T4</td>
<td>T3</td>
<td></td>
</tr>
</tbody>
</table>

True concurrency: Threads execute in parallel on different cores

Demand pulls of multicore systems
- OS designers
  - Scheduling algorithms to harness multiple cores
- Application Programmers
  - Modify existing non-threaded programs
    - Daunting!
  - Design multithreaded programs
Going about writing multithreaded programs (1)

- **Subdivide** functionality into multiple separate & concurrent tasks
- Ensure tasks perform equal work of equal value

Semantics of `fork()` and `exec()` with a multithreaded program

- If one thread calls `fork()`
  - Does new thread duplicate all threads?
  - Is the new process single-threaded?
- Depends on when/if `exec()` is called
  - If immediate: Duplicating all threads unnecessary
  - If NOT: Separate process should duplicate all threads

Complications introduced by threads

If the child process gets as many threads as the parent

- What happens if a thread in the parent was blocked on a read system call?
  - Say from the keyboard
- Are there two threads blocked on the keyboard?
  - When a line is typed, do both threads get a copy?
- Same issue with open network connections

Problems relating to sharing data structures

- What if one thread closes a file ...
  - When another thread is reading from it?
- A thread notices there is little memory
  - Starts allocating more memory
  - Midway in the allocation, a thread-switch occurs
  - New thread notices there is too little memory
    - Starts allocating more memory
    - Memory gets allocated twice!
Support for threads

- Kernel threads
  - Supported & managed by the OS

- User threads
  - User level
  - Above the kernel

- A relationship must exist between user threads and kernel threads

Summarizing threading models

- Many-to-One
- One-to-one

- Many-to-Many
- Two-level

The contents of this slide-set are based on the following references