Threads

CSE 410, Spring 2006

Computer Systems

http://www.cs.washington.edu/education/courses/410/06sp/

Reading and References

- Reading
  - » Chapter 5, *Operating System Concepts*, Silberschatz, Galvin, and Gagne
- Other References
  - » *Inside Microsoft Windows 2000*, Third Edition, Solomon and Russinovich
  - » *Pthreads Programming*, Nichols, Buttlar and Farrell

A Process

- A complete process includes numerous things
  - » address space (all the code and data pages)
  - » OS resources and accounting information
  - » a "thread of control", which defines where the process is currently executing
    - the Program Counter
    - CPU registers

Processes are heavyweight objects

- Creating a new process is costly
  - » lots of data must be allocated and initialized
  - » operating system control data structures
  - » memory allocation for the process
- Communicating between processes is costly
  - » most communication goes through the OS
  - » need a context switch for each process

Parallelism

- Why build a parallel program?
  - » responsiveness to user
  - » web server handling simultaneous web requests
  - » execute faster on a multiprocessor
- One approach using heavyweight processes
  - » create several processes to execute in parallel
  - » map each process to same address space
  - » specify starting address and initial parameters

Parallelism

- With multiple paths of execution, we can implement (or simulate) simultaneous actions
- Why build a parallel program?
  - » responsiveness to user
    - user interface always responds quickly
  - » web server handling simultaneous web requests
    - each request is handled independently
  - » execute faster on a multiprocessor
    - two CPUs can run two programs at once

Parallel processes are expensive

- There's a lot of performance cost
  - » creating these processes
  - » coordinating them through the OS
- There's a lot of duplication
  - » same program code, protection, etc…
- It may be time for a little refinement and complexity ...

Process definition

- What is fundamental in a process?
  - » Code and data
  - » Access and control privileges
  - » Operating system management
    - scheduling, memory map, ...
- What else is there?
  - » Program Counter, registers, and stack
- Separate the idea of "process" from the idea of a "thread of control" (PC, SP, registers)

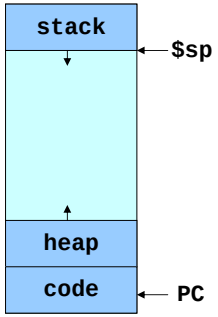
Threads are "Lightweight Processes"

- Most operating systems now support two entities
  - » the process, which defines the address space and general process attributes
  - » the thread, which defines one or more execution paths within a process
- Threads are the unit of scheduling
- Processes are the "containers" in which threads execute

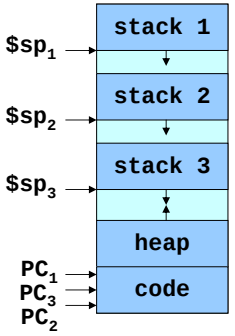
Multi-threaded design benefits

- Separating execution path from address space simplifies design of parallel applications
- Some benefits of threaded designs
  - » improved responsiveness to user actions
  - » handling concurrent events (e.g., web requests)
  - » simplified program structure (code, data)
  - » more efficient and so less impact on system
  - » map easily to multi-processor systems

One thread



Three threads



Cookbook Analogy

- Think of a busy kitchen
  - » 3 cooks and 1 cookbook
- Each cook maintains a pointer to where they are in the cookbook (the Program Counter)
- Two cooks could both be making the same thing (threads running the same procedure)
- The cooks must coordinate access to the kitchen appliances (resource access control)

Implementation

- A thread is bound to the process that provides its address space
- Each process has one or more threads
- How are threads actually implemented?
  - » Kernel threads
    - In the kernel (OS) and user mode libraries combined
  - » User threads
    - In user mode libraries alone

Kernel Threads

- The operating system knows about and manages the threads in every program
- Thread operations (create, yield, ...) all require kernel involvement
- Major benefit is that threads in a process are scheduled independently
  - » one blocked thread does not block the others
  - » threads in a process can run on different CPUs

Kernel Thread Performance

- Kernel threads have performance issues
- Even though threads avoid process overhead, operations on kernel threads are still slow
  - » a thread operation requires a kernel call
  - » kernel threads may be overly general, in order to support needs of different users, languages, etc.
  - » the kernel can't trust the user, so there must be lots of checking on kernel calls

User Threads

- To make thread operations faster, they can be implemented at the user level
  - » Each thread is managed by the run-time system
  - » user-mode libraries are linked with your program
- Each thread is represented simply by a PC, registers, stack and a control block, managed in the user's address space

User Thread Performance

- All activities happen in user address space so thread operations can be faster
- But OS scheduling takes place at process level
  - » block entire process if a single thread is I/O blocked
  - » may run a process that is just running an idle thread
- Win2K provides “fibers” as user mode threads
  - » application can schedule its own “lightweight threads” in user mode code

Simplified Thread Interface

- t = thread_create(), thread_start(t)
  - » create a new thread of control and start it
- thread_yield()
  - » voluntarily give up the processor for awhile
- thread_exit()
  - » terminate the calling thread