CISC 434: Processes - I

- Processes - a program in execution
  - Threads
  - Virtualization
  - Further Discussions on Clients and Servers
  - Code Migration
  - Agents
Processes

Process

- A program in execution - a program that is currently being executed on one of the OS’s virtual processors
- The OS creates an independent address space for each created virtual process for running a different program
- The OS maintains a process table to store CPU register values, memory maps, open files, accounting information, privileges, etc.
Threads

- Very similar to a process - can be considered as the execution of a (or part of) program on a virtual processor
- A finer granularity than processes (multiple threads of control in a single process)
- A thread system generally maintains only the minimum information so that several threads can share a single CPU
- Access by threads in a single process are left with the users (For processes, the OS makes sure that any process does not affect the correctness of other processes)
- The performance of a multithreaded application hardly ever be worse than of its single threaded counterpart
Thread Implementation

- Thread implementation – create, destroy, and synchronization of variables
  - User level threads
  - Kernel level threads
  - Hybrid threads
Thread Implementation – contd.

- **User level threads**
  - Executed exactly in user mode
  - Cheaper to create and destroy threads
  - Switching a thread context is easier – can often be done in just a few instructions
  - Disadvantage – invocation of a blocking system call will immediately block the entire process of the thread (including the other threads in that process)
Thread Implementation – contd.

Kernel level threads
- Implemented at the OS’s kernel
- Creation, deletion, synchronization, etc. require system call
- Switching is also as expensive as process switching
Thread Implementation – contd.

- Hybrid Threads – Combination of kernel-level (light weight process, i.e., LWP) and user-level threads
  - An LWP runs in the context of a single (heavy weight) process and there can be several LWPs per process.
  - A user-level thread package can be shared by multiple LWPs – the package has a scheduling routine which is executed by the LWPs to run a thread.
  - A thread table maintains the current set of threads – simultaneous access to the table by multiple LWPs are avoided by using mutexes implemented in the user space.
Thread Implementation – contd.

- **Hybrid Threads - Advantages**
  - Creating, destroying, and synchronization threads is relatively cheap and involves no kernel intervention.
  - Provided that a process has enough LWPs, a blocking system call will not suspend the entire process.
  - There is no need for an application to know about the LWPs. It just needs to know about user-level threads.
  - LWPs can be used in multiprocessing environments (hidden from the application) by executing different LWPs on different CPUs.
Disadvantage

- Still needs to create and destroy LWPs which may be as expensive as kernel level threads,
- However, creating and destroying are not that frequent and are completely controlled by the OS
Multithreaded Clients and Servers

- **Multithreaded clients**
  - High degree of distribution transparency – conceal long inter-process messages
  - Reduce communication latency – establish the communication and do something else
  - Example – operations of Web browsers (fetching different elements separately in parallel, sending requests to multiple replicas at a time)
Multithreaded Clients and Servers

- **Multithreaded servers**
  - Primary use of multithreading is on server side
  - Simplifies server code
  - Easier to develop servers that exploit parallelism for high performance even on uniprocessor systems
  - In multiprocessor system, multithreading for parallelism is even more useful
Multithreaded Servers – An Example

A multithreaded server organized in a dispatcher/worker model

Diagram:
- Dispatcher thread
- Request coming in from the network
- Request dispatched to a worker thread
- Worker thread
- Server
- Operating system
Virtualization

- Introduced in 1970s on the IBM 370 mainframes to use legacy software to run on mainframe hardware

- Major Objectives
  - To maintain legacy software in its platform
  - To reduce the diversity of platforms and machines by letting each application run on its own virtual m/c in a networked environment
  - To provide high degree of portability and flexibility
General Organization of Virtualization Systems

(a) Organization between a program, interface, and system
(b) Organization of virtualizing system A on top of system B
Various Levels of Interfaces in a Computer System

- Between the hardware and software - machine instructions can be invoked by any program (general instructions)

- Between the hardware and software - machine instructions can be invoked only by privileged programs, such as an OS

- An interface of system calls is offered by an operating system

- An interface of library calls is called application programming interface (API) - system calls are hidden by an API
Various Levels of Interfaces in a Computer System—contd.
Two Different Ways of Virtualization

- (a) A process virtual machine, with multiple instances of (application, runtime) combinations – e.g., JVM
- (b) A virtual machine monitor, with multiple instances of (applications, operating system) combinations – e.g., VMware
Virtualization – a process virtual machine

- With multiple instances of (application, runtime) combinations – e.g., JVM
Virtualization – a virtual machine monitor

(b) A virtual machine monitor, with multiple instances of (applications, operating system) combinations – e.g., VMware
Summary

 Threads
  Discussions on processes and threads
  Implementation of threads
    User level threads
    Kernel level threads
    Hybrid threads
  Multithreaded clients and servers

 Virtualization
  Architecture