Lecture 4 - Threads

Instructor : Bibhas Ghoshal (bibhas.ghoshal@iiita.ac.in)

Autumn Semester, 2015

Lecture Outline

- Thread Concept
- Thread Usage
- Multi-threaded Process
- Thread Types
- Threading Issues
- Signal Handling
- Windows/Linux Threads

References and Illustrations have been used from:

- lecture slides of the book - Operating System Concepts by Silberschatz, Galvin and Gagne, 2005
- Modern Operating System by Andrew S. Tanenbaum
- lecture slides of CSE 30341: Operating Systems (Instructor : Surendar Chandra),

Thread Concept

Use of concurrency within an application incurs high process switching overhead

Process switching overhead has two components:

- Execution related overhead : While switching between process, the CPU state of the running process has to be saved and state of the new process has to be loaded (unavoidable)
- Resource related overhead : Process environment contains information related to resources allocated to a process and its interaction with other processes. It leads to large size of process state information, which adds to process swtching information. (avoidable)

Thread Concept

Switching overhead can be reduced by eliminating resource related overhead in some situations.
Example: Occurence of event may result in switching of execution of process P1 to execution of P2. If both P1 and P2 belong to the same application, they share the same data, code and resources; their state information differs only in values contained in CPU, registers and stack (much of it is redundant). This feature is exploited in notion of Thread.

Thread Usage : Need for Threads

- Simpler programming model
  - Decompose applications into multiple sequential threads that run in quasi-parallely
  - Ability for parallel entities to share an address space and all of its data among themselves
- Since **lightweight**, threads are easier to create and destroy
- Threads speed up application involving substantial computing and I/O by overlapping operations

Thread Usage Example

*Example*1 :

User uses word processor for writing a document. Deletes a line from page 1 and then wants to change a phrase in line 1 of page 600. The word processor has to re-format the book upto line 600 beacuse it doesn't know what will be line 1 until it has processed all other pages. Threads help here. If the word-processor is written as a three threaded program :

- Interactive thread - listens to the keyboard and mouse to accept commands
- computing madly in background
- saving document automatically

Thread Concept

Thread : A program execution that uses resources of the process

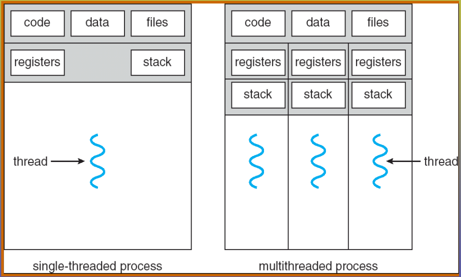
- A thread has its own stack and CPU state
- Thread of same process share code,data and resources with one another
- Kernel allocates stack and Thread Control Block to each thread
- OS saves only CPU state and stack while switching between threads

Thread States

Threads and Process are analogous barring no allocation of resources to threads. Thus, process and thread states are anologous

- When a thread is created, it is put in *ready* state as its parent process already has the resources allocated to it.
- It enters *running* state when scheduled
- It does not enter *blocked* state since it does not request resource; however it can enter bloked state due to process synchronization requirement

Single and Multithreaded Processes



Advantages of threads

- Low overhead - contains only state of computation
- Responsiveness - Interactive applications can be performing two tasks at the same time (rendering, spell checking)
- Resource Sharing - Sharing resources between threads is easy (too easy?)
- Economy - Resource allocation between threads is fast (no protection issues)
- Speed-up - concurrency is realized by creating many threads within a process
- Efficient communication - threads of a processs communicate through shared data space, avoiding system calls for communication
- Utilization of MP Architectures - seamlessly assign multiple threads to multiple processors (if available). Future appears to be multi-core anyway

Thread types

- User threads: thread management done by user-level threads library. Kernel does not know about these threads
    - Three primary thread libraries:
        - POSIX Pthreads
        - Win32 threads
        - Java threads
- Kernel threads: Supported by the Kernel and so more overhead than user threads
    - Examples: Windows XP/2000, Solaris, Linux, Mac OS X
- User threads map into kernel threads

Thread types

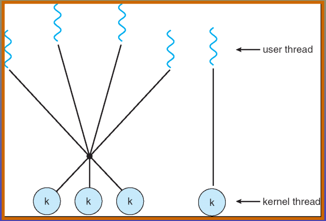
- User threads: thread management done by user-level threads library. Kernel does not know about these threads
- Three primary thread libraries:
    - POSIX Pthreads
    - Win32 threads
    - Java threads
- Kernel threads: Supported by the Kernel and so more overhead than user threads
    - Examples: Windows XP/2000, Solaris, Linux, Mac OS X
- User threads map into kernel threads

Multi-threading models

- Many-to-One: Many user-level threads mapped to single kernel thread
  - If a thread blocks inside kernel, all the other threads cannot run
  - Examples: Solaris Green Threads, GNU Pthreads
- One-to-One: Each user-level thread maps to kernel thread
- Many-to-Many: Allows many user level threads to mapped to many kernel level threads
  - Allows the operating system to create a sufficient number of kernel threads

Two-level Model

Similar to M:M, except that it allows a user thread to be bound to kernel thread - IRIX, HP-UX, Tru64 UNIX, Solaris 8 and earlier



Threading Issues

- Properly functioning OS ensures that protection between processes are not breached. OS makes no such promise for threads within a single process.
    - Two processes cannot share memory unless explicitly allowed. Two threads can trample on the local memory. Sometimes (not always) you get segmentation violation. Regardless, you cannot rely on the OS giving you segmentation fault all the time.
- Semantics of fork() and exec() system calls
    - Does fork() duplicate only the calling thread or all threads?
- Thread cancellation
    - Asynchronous cancellation terminates the target thread immediately
    - Deferred cancellation allows the target thread to periodically check if it should be cancelled

Signal Handling

- Signals are used in UNIX systems to notify a process that a particular event has occurred
  - Ctrl-C sends SIGINT (Interrupt)
  - float x=1/0; sends SIGFPE
- A signal handler is used to process signals
  - Signal is generated by particular event
  - Signal is delivered to a process
  - Signal is handled
- Options:
  - Deliver the signal to the thread to which the signal applies
  - Deliver the signal to every thread in the process
  - Deliver the signal to certain threads in the process
  - Assign a specific thread to receive all signals for the process

Thread Pools and Thread Specific Data

- Create a number of threads in a pool where they await work
- Advantages:
  - Usually slightly faster to service a request with an existing thread than create a new thread
  - Allows the number of threads in the application(s) to be bound to the size of the pool

Thread Specific Data :

  - Allows each thread to have its own copy of data
  - Useful when you do not have control over the thread creation process (i.e., when using a thread pool)

Scheduler Activations

- Both M:M and Two-level models require communication to maintain the appropriate number of kernel threads allocated to the application
- Scheduler activations provide upcalls - a communication mechanism from the kernel to the thread library
- This communication allows an application to maintain the correct number kernel threads

Windows XP Threads

- Implements the one-to-one mapping
- Each thread contains
  - A thread id
  - Register set
  - Separate user and kernel stacks
  - Private data storage area
- The register set, stacks, and private storage area are known as the context of the threads
- The primary data structures of a thread include:
  - ETHREAD (executive thread block)
  - KTHREAD (kernel thread block)
  - TEB (thread environment block)

Linux Threads

- Linux refers to them as tasks rather than threads
- Thread creation is done through clone() system call
- clone() allows a child task to share the address space of the parent task (process)