Threads

Multithreading Models

Threading Issues

Threads

• Threads
  – Motivation
  – Benefits
  – User and Kernel Threads

• Multithreading Models
  – Many to 1 Model
  – 1 to 1 Model
  – Many to Many Model

• Threading Issues

Threads - Motivation

• Single Threaded Process
  – Only services one client at a time
  – Clients may have to wait prohibitive time for request to be completed

• Alternative Solutions?
  – Let server run as single process accepting requests
  – Use process-creation method
  • Let server process generate processes to service requests, one process for each request

Threads - Motivation

• Process-Creation Method
  – Heavyweight approach with high overheads, see GSS Chapter 4

• Multithread the web-server Process
  – If new process performs same tasks as existing process then generally more efficient to employ one process with multithreads instead of above approach
Threads - Motivation

- Multithread Process
  - Process creates thread to listen for client requests
  - Upon receipt of request, process creates another thread to service request

- Benefits
  - can break into 4 major categories
    - Responsiveness
    - Resource sharing
    - Economy
    - Utilisation of multiprocessor architectures

Threads - Benefits

- Responsiveness
  - Multithreading interactive application may allow program to continue running even if it is blocked or performing a lengthy operation, hence increased responsiveness
  - E.g. a multithreaded web browser could allow user interaction in one thread while image is loaded by another thread

- Resource sharing
  - By default threads share the same memory as the process to which they belong (See CDK p215)
  - Benefit allows application to have several different threads of activity within the same address space

- Economy
  - Allocation of Memory and resources for process creation is costly
  - Threads share the resources of the process to which they belong, hence more economical to create and context switch threads
  - SGG p131 - indicate (empirically):
    - can be difficult to gauge difference in overhead for creating and maintaining process over thread
    - In general much more time consuming to create and manage processes than threads
      - Solaris 2 creating threads 30 times quicker than creating processes
      - Context switching 5 times faster with threads

- Utilisation of multiprocessor architectures
  - Benefits of multithreading can be greatly increased on a multiprocessor architecture
  - Each thread may run in parallel on a different processor
  - Single-threaded processes can only run on one CPU, no matter how many CPU’s are available
  - Multithreading on multi-CPU machines increases concurrency
  - Single processor architecture CPU generally moves between each thread creating illusion of parallelism, but only one thread is running at a time

Threads – User and Kernel Threads

- Support may be provided at:
  - User level, for user threads, Kernel level for kernel threads

- User threads (Supported above kernel level)
  - Implemented by thread library at user level
    - Provides support for thread creation, scheduling, and management with no support from the kernel
    - Thread creation and scheduling carried out in user space without need for kernel intervention
    - Hence generally fast to create and manage
    - Drawbacks exist e.g. if kernel single threaded then any user level thread performing a blocking system call will cause entire process to block even if others can run within application
Threads – User and Kernel Threads

- Kernel threads
  - Supported directly by O/S
    - Kernel performs thread creation, scheduling, and management in kernel space
    - Thread management carried out by O/S
    - Hence generally slower to create and manage than user threads
  - Advantages exist
    - Since kernel manages thread, if thread performs a blocking system call, kernel can schedule another thread in the application for execution
    - In a multiprocessor environment, kernel can schedule threads on different processors

- User thread Libraries include:
  - POSIX Pthreads (SGG Section 5.6)
  - MACH C-threads, Solaris 2 UI-threads

- Most contemporary O/S’s support kernel threads
  - Windows NT, Windows 2000 (SGG Section 5.6)
  - Solaris 2 (SGG Section 5.5), BeOS
  - Tr64 UNIX (formerly Digital UNIX)
  - For discussions on LINUX and JAVA (JVM) support for threads (SGG Section 5.7-5.8)

Multithreading Models

- Many O/S’s provide support for both user and kernel threads
- Resulting from this different multithreading models have emerged
- We consider three such models:
  - Many to 1 Model
  - 1 to 1 Model
  - Many to Many Model

Many – 1 Model

- Maps many user threads to one kernel thread
- Thread management occurs in user space
  - Efficient, from discussion above (Advantage)
  - Entire process blocks if thread makes blocking system call (Disadvantage)
  - Multiple threads unable to run in parallel on multiprocessors since only one thread can access kernel at a time (Disadvantage)
- Users of Many – 1 Model
  - Green Threads – a thread library available for Solaris 2
  - User thread libraries for O/S’s that do not support kernel threads

1 to 1 Model

- For discussions on LINUX and JAVA (JVM) support for threads (SGG Section 5.7-5.8)
1 – 1 Model

- Each user thread maps to a single kernel thread
- Provides more concurrency than Many – 1 Model
  - Since other threads may run if a thread issues a blocking system call (Advantage)
  - Allows multiple threads to run in parallel on multiprocessors (Advantage)

Disadvantages
- Each time a user thread is created a corresponding kernel thread has also to be created
- Overhead of creating kernel threads can burden performance of an application

Implementations of Model
- Most restrict number of threads supported by system
- Windows 2000, Windows NT, OS/2 all implement this model

Multithreading Models

Many to Many Model
- Multiplexes many user level threads to smaller or equal number of kernel threads
- Number of kernel threads may be specific to a particular
  - Application
  - Machine
- Example
  - Application may be allocated more kernel threads on multiprocessor than on uniprocessor

Many – 1 Model
- Although model allows developer to create as many user threads as wished for
  - True concurrency is NOT achieved since kernel can schedule only one thread at a time
- 1 – 1 model
  - Allows for greater concurrency
  - Developer has to be careful not to create too many threads within an application (developer may be limited to number that may be created)

Many – Many Model
- Does not suffer from above shortcomings
- Developer may create as many user threads as required
- Corresponding kernel threads can run in parallel on multiprocessor
- If thread performs blocking system call
  - Kernel can schedule another thread for execution
- Implementations of Model
  - Solaris 2, IRIX, HP-UX, Tru64 UNIX