Multi-core Architecture and Programming

Yang Quansheng(杨全胜)
http://www.njyangqs.com/

School of Computer Science & Engineering
Southeast University

Process, threads and Parallel Programming

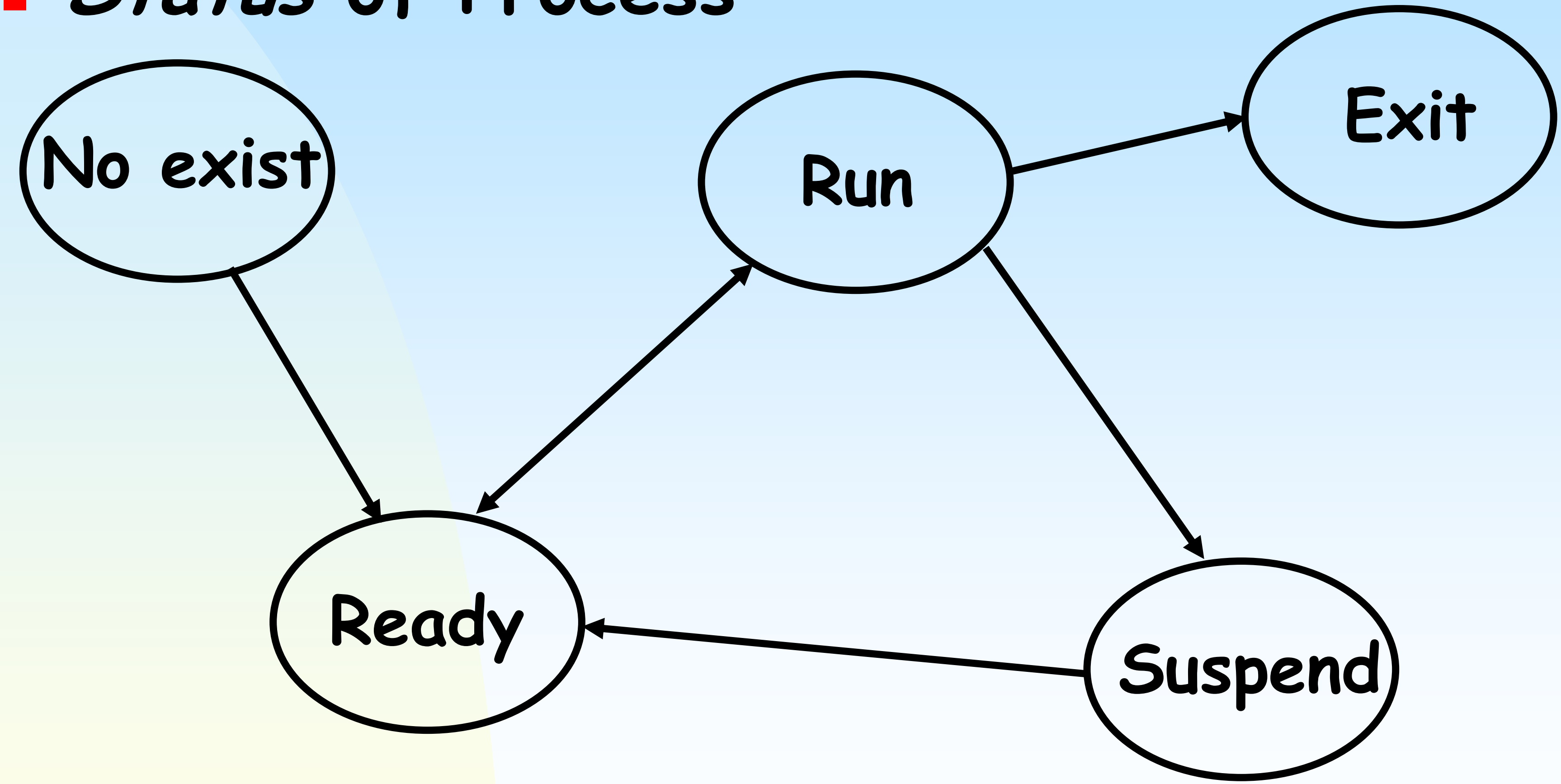
- Content
  - Concepts of Process
  - What is a threads
  - Designing for threads
  - Mutual Exclusion and Synchronization
  - Common Parallel Programming Problems

Concepts of Process

- A running of a program
- A quadruple (P,C,D,S)
  - P-Program
  - C-Control state
  - D-Data
  - S-Executing state
- Characteristics
  - Own the resource
  - Be scheduled by OS

Concepts of Process

- *Status of Process*



Concepts of Process

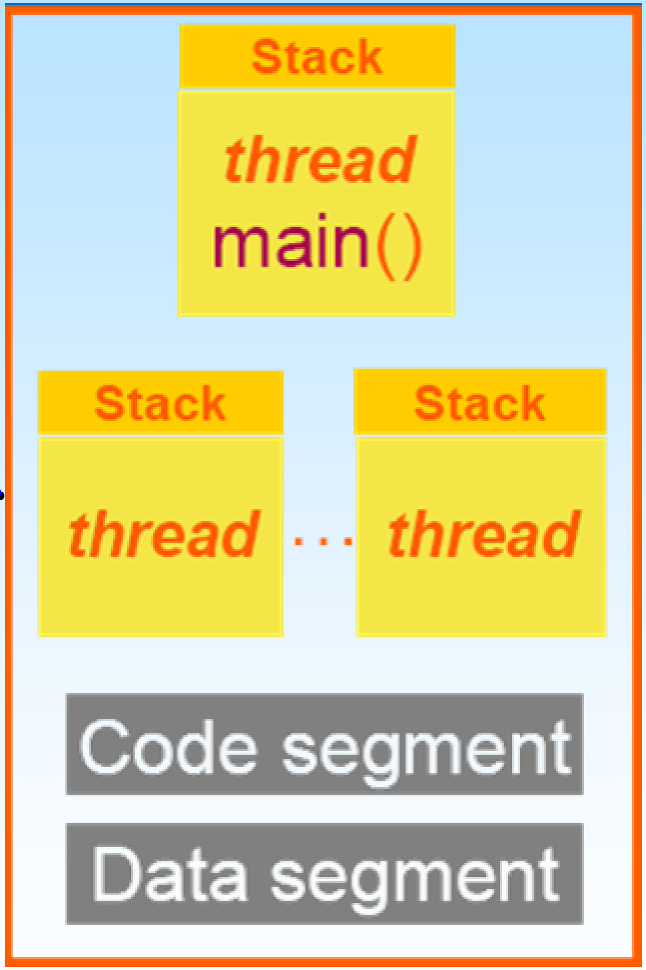
- Communication among the process
  - Mode
    - communicate
    - synchronization
    - congregate
  - In share memory mode, communication can realized via read/write the share buffer supported by OS
  - In distributed memory mode, communication depend on the network

Process, threads and Parallel Programming

- Content
  - Concepts of Process
  - What is a threads
  - Designing for threads
  - Mutual Exclusion and Synchronization
  - Common Parallel Programming Problems

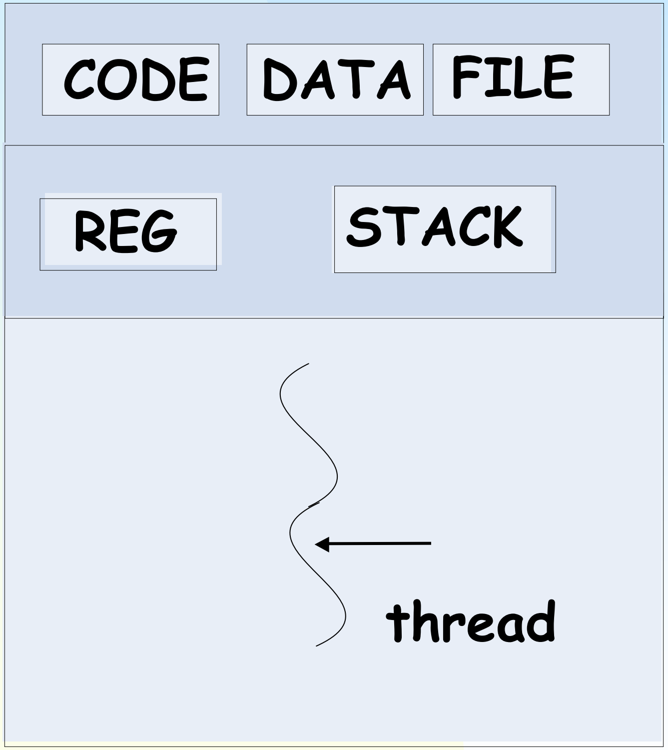
What is a threads

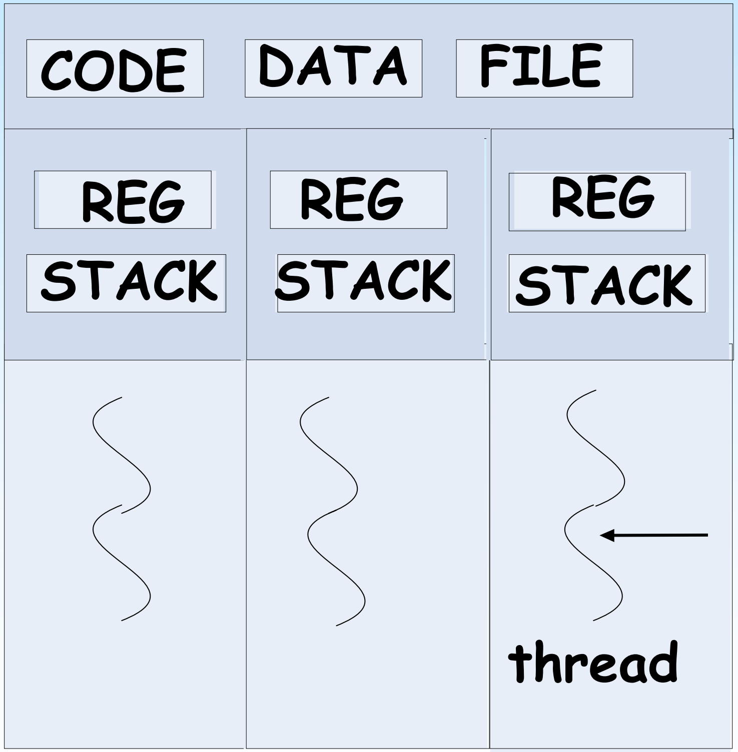
- Processes and Threads
  - Modern operating systems load programs as processes
    - Resource holder
    - Execution
  - A process starts executing at its entry point as a thread
  - Threads can create other threads within the process
    - Each thread gets its own stack
  - All threads within a process share code & data segments



What is a threads

- Thread is the light weight process. It is the basic unit can be scheduled by OS.





What is a threads

- The level of threads
  - User threads
    - Threads in applications
    - Created and managed by threading API
      - OpenMP
      - Pthreads
      - Windows thread API
  - Kernel threads
    - Different kernel threads within process can ran in different CPU or core
  - Hardware threads
    - SMT: Hyper-Threading
    - CMT: Multi-core

What is a threads

- Mapping mode from thread to processor
  - One thread to one processor
    - Preemptive multi-threading
    - Linux, Windows XP
  - M threads to one processor
    - Cooperative multi-threading
    - Need a thread scheduler to select one thread into processor
  - M threads to N processors

Process, threads and Parallel Programming

- Content
  - Concepts of Process
  - What is a threads
  - Designing for threads
  - Mutual Exclusion and Synchronization
  - Common Parallel Programming Problems

Designing for threads

- Threading for Functionality or Performance?
  - Threading for Functionality
    - Assign threads to separate functions done by application
      - Easiest method since overlap is unlikely
    - Example: *Building a house*
      - Bricklayer, carpenter, roofer, plumber,…
  - Threading for Performance
    - Increase the performance of computations
    - Thread in order to improve turnaround or throughput
    - Examples: Searching for pieces of Skylab
      - Divide up area to be searched

Designing for threads

- Threading for turnaround or throughput
  - Turnaround
    - Complete single task in the smallest amount of time
    - Example: *Setting a dinner table*
      - One to put down plates
      - One to fold and place napkins
      - One to place utensils
        - Spoons, knives, forks
      - One to place glasses

Designing for threads

- Threading for turnaround or throughput
  - Throughput
    - Complete the most tasks in a fixed amount of time
    - Example: *Setting up banquet tables*
      - Multiple waiters each do separate tables
      - Specialized waiters for plates, glasses, utensils, etc

Designing for threads

- Task Decomposition
- Data Decomposition
- Data Flow Decomposition

Decomposition	Design	Comments
Task	Different activities assigned to different threads	Common in GUI apps
Data	Multiple thread performing the same operation but on different blocks of data	Common in audio processing, imaging, and in scientific programming
Data Flow	One thread's output is the input to a second thread	Special care is needed to eliminate startup and shutdown latencies

Designing for threads

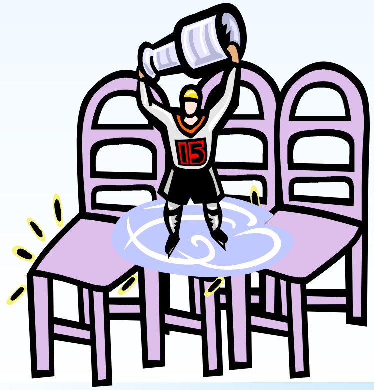
- Benefit of multi-threads
  - Create a thread cost less than process
  - Switching between threads cost less than that between process
  - Take full advantage of multi-processor and multi-core
  - Sharing data through memory more efficient than message-passing
- Risks
  - Increases complexity of application
  - Difficult to debug (data races, deadlocks, etc.)

Process, threads and Parallel Programming

- Content
  - Concepts of Process
  - What is a threads
  - Designing for threads
  - Mutual Exclusion and Synchronization
  - Common Parallel Programming Problems

Mutual Exclusion and Synchronization

- Race Conditions
  - Threads "race" against each other for resources
    - Execution order is assumed but cannot be guaranteed
  - Storage conflict is most common
    - Concurrent access of same memory location by multiple threads
      - At least one thread is writing
  - Example: Musical Chairs



Mutual Exclusion and Synchronization

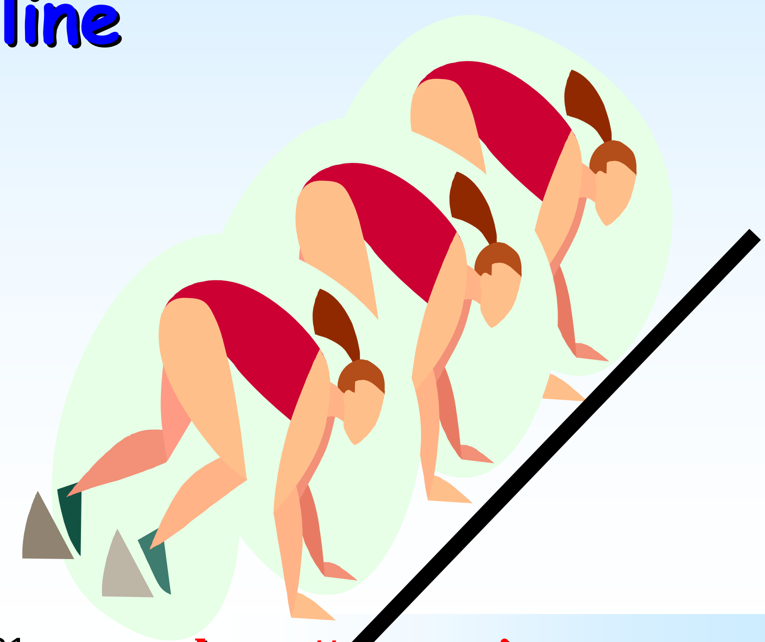
- Mutual Exclusion
  - Critical Region
    - Portion of code that accesses (reads & writes) shared variables
  - Mutual Exclusion
    - Program logic to enforce single thread access to critical region
    - Enables correct programming structures for avoiding race conditions
  - Example: Safe Deposit box
    - Attendants ensure mutual exclusion

Mutual Exclusion and Synchronization

- Synchronization
  - Synchronization objects used to enforce mutual exclusion
    - Lock, semaphore, critical section, event, condition variable, atomic
    - One thread "holds" sync. object; other threads must wait
    - When done, holding thread releases object; some waiting thread given object
  - Example: Library book
    - One patron has book checked out
    - Others must wait for book to return

Mutual Exclusion and Synchronization

- Barrier Synchronization
  - Threads pause at execution point
    - Threads waiting are idle; overhead
  - When all threads arrive, all are released
  - Example: Race starting line



Mutual Exclusion and Synchronization

- Deadlock
  - Deadlock occurs whenever a thread is blocked waiting on a resource of another thread that will never be released.
  - According to the circumstances, different deadlocks can occur:
    - Self-deadlock
    - Recursive deadlock
    - Lock-ordering deadlock

Mutual Exclusion and Synchronization

- Starvation
  - Starvation occurs whenever a thread is waiting on a resource of another thread that will never available to it in spite of be released. If the waiting state is everlasting, it means starve to death.
  - Example:
    - A large file in a printing system that small file first, maybe wait printer for ever.
- Livelock
  - Starvation occurs during busy waiting

Mutual Exclusion and Synchronization

- Synchronization Primitives
  - **Semaphore**
    - Semaphore can be represented by an integer and can be bounded by two basic atomic operations:
    - P: proberen, which means test
    - V: verhogen, which means increment

P(mutex);
Critical section;
V(mutex);

Mutual Exclusion and Synchronization

- Synchronization Primitives
  - **Locks**
    - Locks are similar to semaphores except that a single thread handles a lock at one instance.
    - Two basic atomic operations get performed on a lock:
      - Acquire(): Atomically waits for the lock state to be unlocked and sets the lock state to lock.
      - Release(): Atomically changes the lock state form locked to unlocked

Mutual Exclusion and Synchronization

- Synchronization Primitives
  - Condition Variables
    - Condition variables are also based on Dijkstra's semaphore semantics, with the exception that no stored value is associated with the operation.
      - Wait: Atomically releases the lock and waits, where wait returns the lock been acquired again.
      - Signal: Enables one of the waiting threads to run, where signal returns the lock is still acquired.
      - Broadcast: Enables all of the waiting threads to run, where broadcast returns the lock is still acquired.

Process, threads and Parallel Programming

- Content
  - Concepts of Process
  - What is a threads
  - Designing for threads
  - Mutual Exclusion and Synchronization
  - Common Parallel Programming Problems

Common Parallel Programming Problems

- Too Many Threads?
  - Degrade with too many threads
  - The impact comes in two ways:
    - The overhead of starting and terminating threads swamps the useful work
    - Overhead from having to share fixed hardware resources
  - Useful practices
    - Let OpenMP do the work
    - Use a thread pool
    - Work stealing

Common Parallel Programming Problems

- Synchronization
  - Data Races
  - Deadlocks
  - Live Locks

Common Parallel Programming Problems

- Heavily Contended Locks
  - Priority inversion
    - A low-priority thread blocks a high-priority thread from running
  - Solutions
    - Replicate the resource
    - Partitioning the resource and using a separate lock to protect each partition
    - Fine-grained locking

Common Parallel Programming Problems

- Non-blocking Algorithms
  - Three different non-blocking guarantees
    - Obstruction freedom
    - Lock freedom
    - Wait freedom
  - ABA problem
  - Cache line Ping-ponging
  - Memory reclamation problem

Common Parallel Programming Problems

- Memory Issues
  - Bandwidth
    - Pack data more tightly
    - Move data less frequently between cores
  - Working in the cache
    - Minimizing data movement
      - Cache-oblivious blocking

Common Parallel Programming Problems

- Cache-related Issues
  - False sharing
    - Cache line ping ponging
    - Lower Performance
  - Memory consistency
  - Intel Architecture
    - IA-32 Architecture
    - Itanium Architecture