CS203 Programming with Data Structures

Introduction to Threads and Synchronization

Chengyu Sun
California State University, Los Angeles

Processes

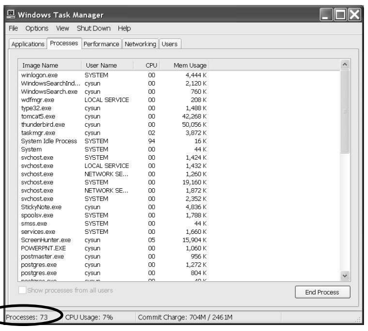

Windows Task Manager
File Options View Shut Down Help
Applications Processes Performance Networking Users
Show processes from all users
End Process

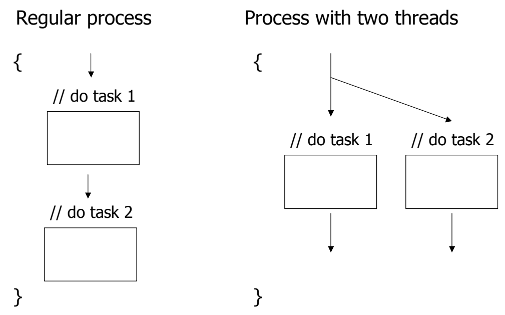
Image Name	User Name	CPU	Mem Usage
winlogon.exe	SYSTEM	00	4,444 K
WindowsSearchInd...	cysun	00	2,120 K
WindowsSearch.exe	cysun	00	760 K
wdfmgr.exe	LOCAL SERVICE	00	208 K
type32.exe	cysun	00	1,488 K
tomcat5.exe	cysun	00	42,268 K
thunderbird.exe	cysun	00	50,056 K
taskmgr.exe	cysun	02	3,872 K
System Idle Process	SYSTEM	94	16 K
System	SYSTEM	00	44 K
svchost.exe	SYSTEM	00	1,424 K
svchost.exe	LOCAL SERVICE	00	1,432 K
svchost.exe	NETWORK SE...	00	1,260 K
svchost.exe	SYSTEM	00	19,160 K
svchost.exe	NETWORK SE...	00	1,872 K
svchost.exe	SYSTEM	00	2,352 K
StickyNote.exe	cysun	00	4,836 K
spoolsv.exe	SYSTEM	00	1,788 K
smss.exe	SYSTEM	00	44 K
services.exe	SYSTEM	00	1,660 K
ScreenHunter.exe	cysun	05	15,904 K
POWERPNT.EXE	cysun	00	1,060 K
postmaster.exe	cysun	00	956 K
postgres.exe	cysun	00	1,272 K
postgres.exe	cysun	00	804 K

Processes: 73 | CPU Usage: 7% | Commit Charge: 704M / 2461M

Multitasking

- What is multitasking?
- Why do we need multitasking?
  - A long running process should not block all other processes
  - Fully utilize the resources of a computer
    - CPUs, graphic card, hard drives etc.

Multitasking within a Process – Threads



Thread Example

- A program performs two tasks
  - Calculate Fibonacci(n)
  - Download a web page
- Without thread: `ThreadTest1.java`
- With thread: `ThreadTest2.java`

Creating A Thread

- Subclass `Thread` class
  - http://java.sun.com/j2se/1.5.0/docs/api/java/lang/Thread.html
- Implement `Runnable` interface
  - http://java.sun.com/j2se/1.5.0/docs/api/java/lang/Runnable.html

Subclass `Thread` Class

- `class Foobar` **`extends Thread`**
  - Override `run()` method
- `Thread newThread = new Foobar();`

Implement `Runnable` Interface

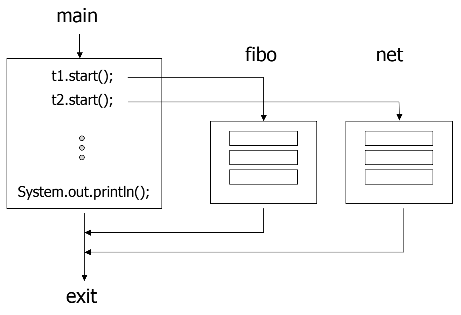
- `class Foobar` **`implements Runnable`**
  - Implement `run()` method
- `Thread newThread =` **`new Thread(`**`new Foobar()`**`)`**

How do we choose between these two approaches??

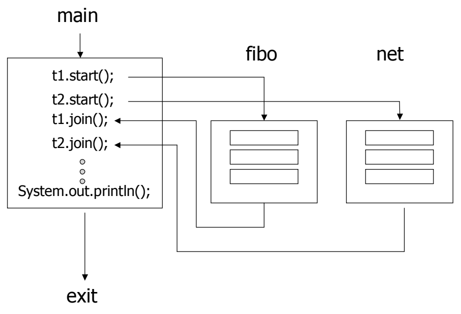
Run a Thread

- `start()` in the `Thread` class
- `start()` is *non-blocking*

Without `join()`



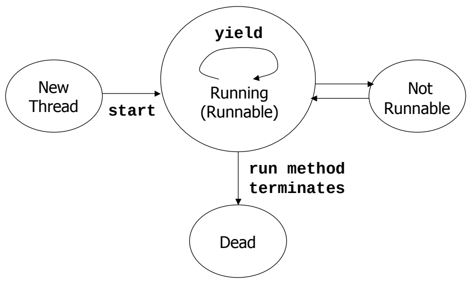
With `join()`



Collaboration between Processes/Threads

- Processes
  - Do not share address space
  - Collaborate through message passing
- Threads
  - Share address space
  - Collaborate through shared memory (usually faster than message passing)

Life Cycle of a Thread



Scheduling

- What happens in the running/runnable state?
- Scheduling – pick a thread from the runnable threads and run it
  - Time slicing
  - JVM default: *Fixed Priority Scheduling*

Fixed Priority Scheduling

- Threads with higher priority are run first
- Threads with the same priority are run in a round-robin manner.
- Threads with lower priority are only run when high priority threads are either *dead* or *not runnable*.
- Preemptive – current thread may be stopped if there's a thread with higher priority is runnable

Runnable → Not Runnable

- `sleep()` method is invoked
- `wait()` method is invoked
- Blocked on I/O

Not Runnable → Runnable

- Sleep time expires
- `notify()` or `notifyAll()` method is invoked
- I/O is completed

Producer/Consumer Example

- A *producer* thread writes 0, 1, 2,…, 9 into a buffer
- A *consumer* thread reads from the buffer
- If two threads are perfectly synchronized, the consumer thread should read 0, 1, 2, 3,…, 9, but …

From Non-synchronized to Synchronized

- `Thread.sleep(1000)` – just to make things more interesting
- `wait()` and `notify()`
- `synchronized`

Beyond Basics

- High-level Thread API
  - `Timer` and `SwingWorker`
- Semaphores, locks, conditions
- Scheduling
- Deadlock and starvation

- *So take CS440*