CS 475: Concurrent & Distributed Systems

Prof. Sanjeev Setia
Computer Science Dept
George Mason University

About this Class

- Focus: designing and writing moderate-sized concurrent and distributed applications
  - Fundamental concepts
  - Multi-threaded and distributed programs
- See syllabus for course learning outcomes
- Prerequisites:
  - CS 367 (Computer Systems & Programming)
  - High level of competence in C and Java

What you will learn

"*I hear and I forget, I see and I remember, I do and I understand*" – Chinese proverb

- Fundamental concepts in the development of concurrent & distributed software
- Developing Concurrent Programs
  - Threads, semaphores, condition variables, monitors
- Middleware technology for distributed applications
  - Network programming using TCP/IP Sockets
  - RPC/RMI
    - Web Services

Logistics

- Grade: 50% projects and homework, 50% exams
  - 15% quizzes, 15% midterm, 20% final (cumulative)
  - Tentative date of midterm – Oct 23
- Four programming assignments and (at least) one paper and pencil
  - First three assignments roughly same weight, fourth higher weight
  - Can be done in groups of two
  - First three assignments use C, fourth Java
  - Assignments will be graded on VS&E Linux server (zeus)
    - If you do your development elsewhere, your responsibility to make sure it runs correctly on zeus
- Homework problems
  - To be done individually

Logistics cont'd

- Online Assignment submission
  - Blackboard (mymason.gmu.edu)
  - Grades posted on Blackboard
- Lateness Policy
  - Four "slip" days collectively for four programming asssignments
  - Can use at most two slip days for an assignment
  - Late submissions not accepted for "paper and pencil" homework assignments
- Honor Code
- Classroom Policy: Use of laptops/PDAs not permitted

CS 475 5

Logistics cont'd

- Office Hrs
  - Thursday, 1-3 pm, Room 4300, Engineering Bldg
- Email: setia@gmu.edu
- GTA: Nusha Mehmanesh
  - Email nmehmane@gmu.edu
  - Office hrs: Monday 1-3 PM, Tuesday 2-4 PM, Wednesday 10 AM – 12 PM
  - Office: TBA
- Piazza – discussion forum
- Class materials will be posted on Blackboard page for course

CS 475 6

Readings

- Recommended books
  - Computer Systems & Programming (Bryant & O'Halloran) – used in CS 367
  - Operating Systems: Three Easy Pieces (Arpaci-Dusseau and Arpaci-Dusseau) – online text
    - http://pages.cs.wisc.edu/~remzi/OSTEP/
  - Distributed Systems: Concepts & Design (Coulouris et al)
- Read class slides & notes

Programming Assignments

- Assignment 1: Shell Lab - C
  - Topic: Creating and managing concurrent processes
- Assignment 2: Multithreaded programming Lab – C
  - Topic: Concurrent programming, synchronization
- Assignment 3: Network programming lab - C
  - Topic: network programming, multi-threaded programming, synchronization
- Assignment 4: Calendar Lab - Java
  - Topic: RMI, distributed application development

Schedule (tentative)

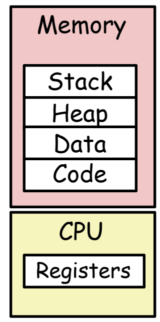
- Concurrent Programming
- Process Synchronization
- Parallel processing on Multicores (introduction)
- Introduction to Networking
- Sockets; Application-level network protocols
- Introduction to distributed systems
- RPC/RMI
- Web Services (introduction)
- And if we have time….. which is unlikely
  - Peer-to-peer computing (introduction)
  - Parallel Programming on multi-computers (introduction)
  - Map Reduce…

Hardware Architectures

- Uniprocessors
- Shared-memory multiprocessors
- Distributed-memory multicomputers
- Distributed systems

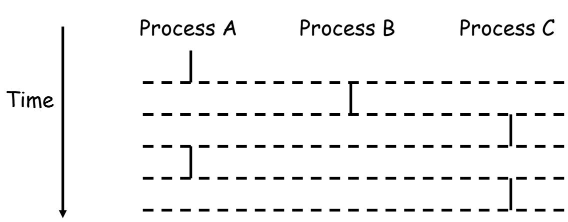
Processes

- Def: A *process* is an instance of a running program.
  - One of the most profound ideas in computer science.
  - Not the same as "program" or "processor"
- Process provides each program with two key abstractions:
  - Logical control flow
    - Each program seems to have exclusive use of the CPU.
  - Private address space
    - Each program seems to have exclusive use of main memory.
- How are these illusions maintained?
  - Process executions interleaved (multitasking)
  - Address spaces managed by virtual memory system



Concurrent Processes

- Two processes *run concurrently* (*are concurrent)* if their flows overlap in time.
- Otherwise, they are *sequential.*
- Examples:
  - Concurrent: A & B, A & C
  - Sequential: B & C



Cooperating Concurrent Processes

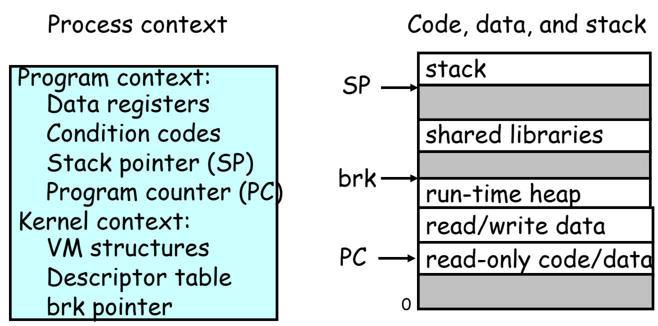
- Concurrent processes part of the same application
- Processes "cooperate" on task
- Motivation
  - Support inherent concurrency in application
    - Window systems, web servers
  - Improved performance - can make use of multiple processors

Concurrent Programming

- Process = Address space + one thread of control
- Concurrent program = **multiple threads of control**
  - Multiple single-threaded processes
  - Multi-threaded process

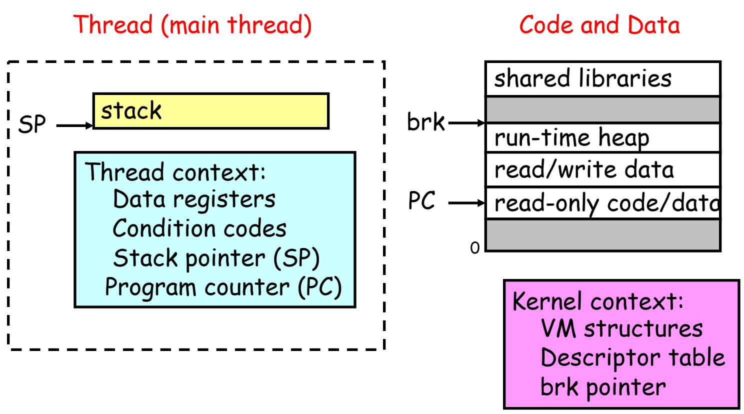
Traditional View of a Process

- Process = process context + code, data, and stack



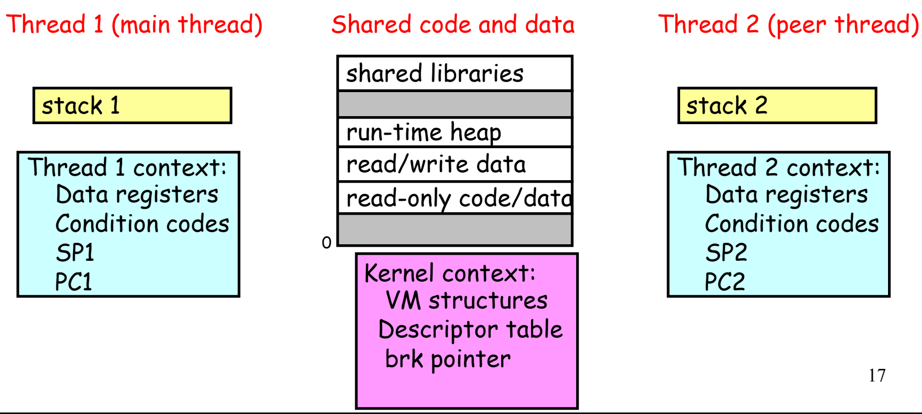
Alternate View of a Process

- Process = thread + code, data, and kernel context



A Process With Multiple Threads

- Multiple threads can be associated with a process
  - Each thread has its own logical control flow (sequence of PC values)
  - Each thread shares the same code, data, and kernel context
  - Each thread has its own thread id (TID)



Threads: Motivation

- Traditional processes created and managed by the OS kernel
- Process creation expensive - fork system call in UNIX
- Context switching expensive
- Cooperating processes - no need for memory protection (separate address spaces)

Threads

- Execute in same address space
  - separate execution stack, share access to code and (global) data
- Smaller creation and context-switch time
- Can exploit fine-grain concurrency

Challenges in multi-threaded/concurrent programming

- Synchronizing multiple processes/threads
  - Locks
  - Semaphores
  - Monitors
  - Deadlocks
  - Livelocks
- Testing/debugging concurrent applications is a lot harder!

Application classes

- Multi-threaded Programs
  - Processes/Threads on same computer
  - Window systems, Operating systems
- Distributed computing
  - Processes/Threads on separate computers
  - File servers, Web servers
- Parallel computing
  - On same (multiprocessor) or different computers
  - Goal: solve a problem faster or solve a bigger problem in the same time

Distributed systems

- "Workgroups"/Intranets
- ATM (bank) machines
- World wide web
- Multimedia conferencing
- Ubiquitous network-connected devices
  - Internet of Things

Distributed applications

- Applications that consist of a set of processes that are distributed across a network of machines and work together as an ensemble to solve a common problem
- In the past, mostly "client-server"
  - Resource management centralized at the server
- Peer-to-peer applications represent "truly" distributed applications

Goals/Benefits

- Resource sharing
- Scalability
- Fault tolerance and availability
- Performance
  - Parallel computing can be considered a subset of distributed computing

Challenges (Differences from Local Computing)

- Heterogeneity
- Latency
  - Interactions between distributed processes have a higher latency
- Memory Access
  - Remote memory access is not the same as local memory access
    - Local pointers are meaningless outside address space of process

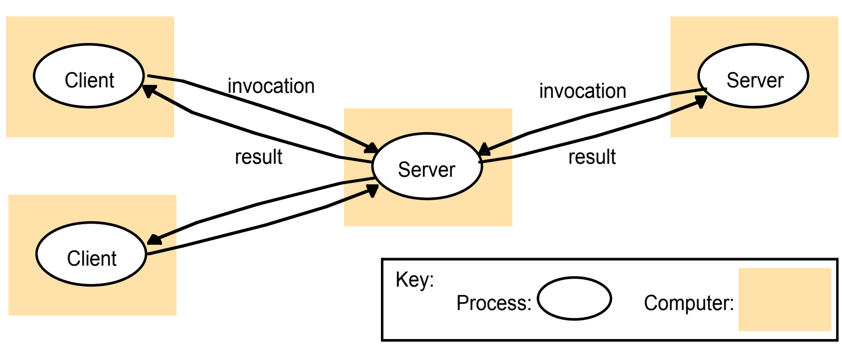
Challenges cont'd

- Synchronization
  - Concurrent interactions the norm
- Partial failure
  - Applications need to adapt gracefully in the face of partial failure
  - Leslie Lamport (a famous computer scientist) once defined a distributed system as "One on which I cannot get any work done because some machine I have never heard of has crashed"

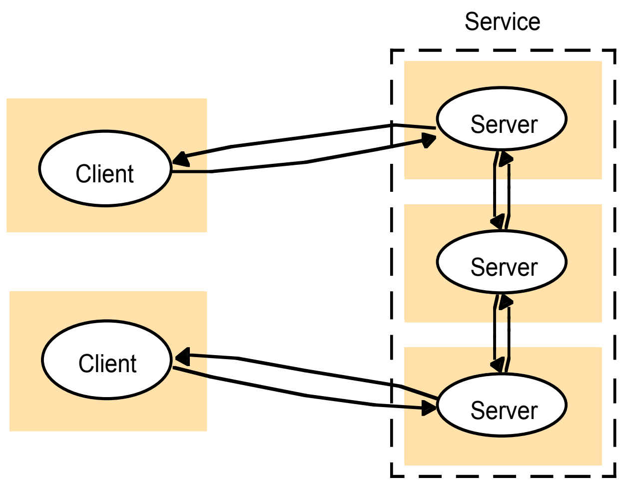
Communication Patterns

- Client-server
- Group-oriented
  - Applications that require reliability
- Function-shipping
  - Java applets

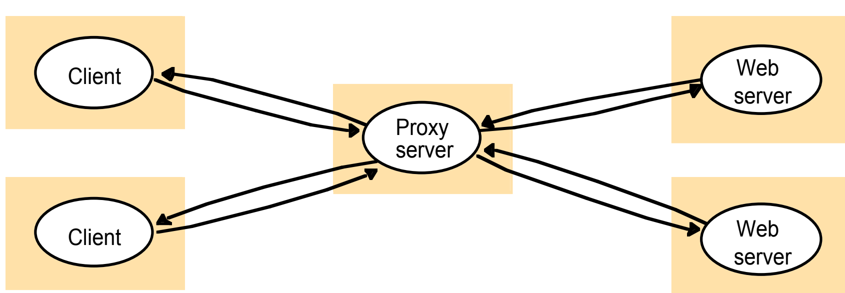
Clients invoke individual servers



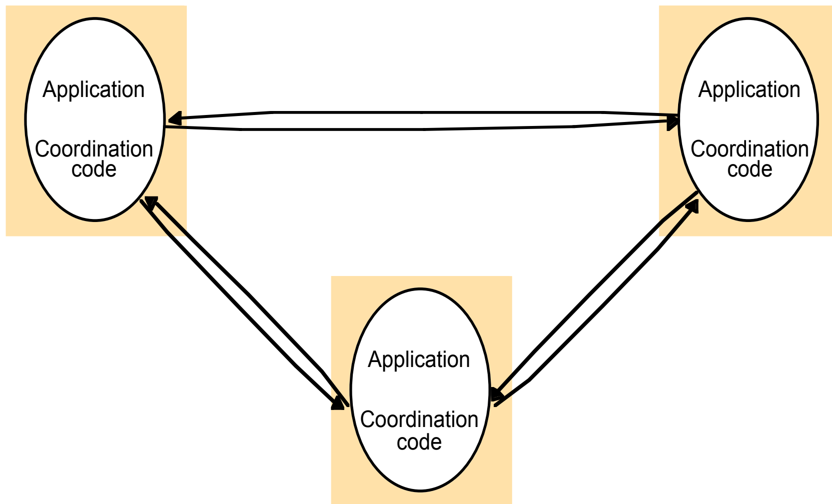
A service provided by multiple servers



Web proxy server

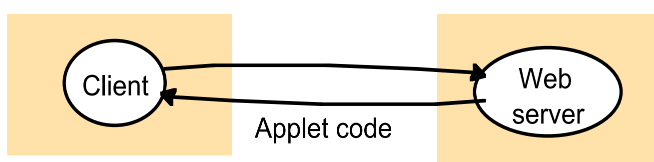


A distributed application based on peer processes



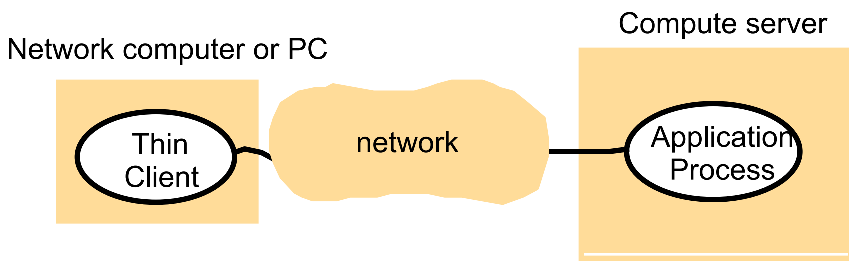
Web applets

a) client request results in the downloading of applet code



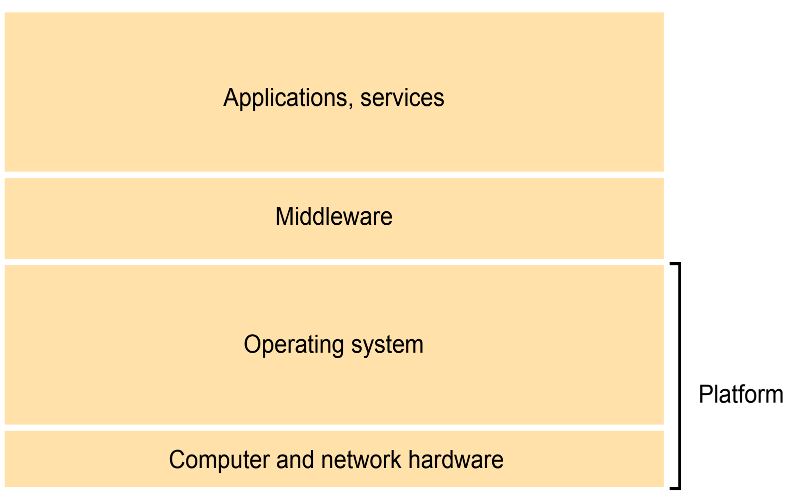
b) client interacts with the applet

Thin clients and compute servers



Cloud Computing

Software and hardware service layers in distributed systems



Road Map

- Next class: Processes and Threads
- Next week: Processes and Signals
  - Continuation of material introduced in CS 367
- Week 3-5: Concurrent programming
- Assignment 1: Shell Lab (officially assigned next week but may be posted on Blackboard this week)