The University of Melbourne
School of Computing and Information Systems

COMP30023
Computer Systems

Semester 1, 2017

Introduction

Welcome to COMP30023

Lecturer: **A/Prof Michael Kirley**

Workshops will start in the second week of semester. Your personal timetable will tell you which tutorial / lab you should attend.

Assessment:

- A two-hour written exam worth 60%
- A mid-semester test worth 10%
- Two projects (with multiple parts) worth a total of 30%

There is a 50% hurdle on each assessment component: to pass the subject, you must get at least 35/70 for the exam and mid-semester test combined, and at least 15/30 for the projects.

Syllabus

- Operating Systems
    - introduction/overview
    - processes and scheduling
    - memory management
    - file systems
    - synchronization
- Network Services
    - network fundamental
    - layered protocols; TCP/IP
    - socket programming
    - applications and services
- Security
    - cryptography
    - authentication
    - security in practice

Resources

- Tanenbaum, A.S. *Modern operating systems (Fourth Edition)*. Pearson/Prentice Hall
- Silberscatz, A., Galvin, P.B. and Gagne, G. *Operating System Concepts, (Eighth Edition)*, John Wiley & Sons.
- Tanenbaum, A.S. *Computer networks (Fifth Edition)*. Pearson Education International / Prentice Hall
- Kurose, J.F and Ross, K.W. *Computer Networking: A Top-Down Approach Featuring the Internet (Fifth Edition)*. Addison Wesley.
- Stevens, W.R. and Rago, S.A. Advanced programming in the UNIX environment. Addison Wesley

Acknowledgement

The slides were prepared by Michael Kirley based on material developed previously by: Zoltan Somogyi, Rao Kotagiri, James Bailey and Chris Leckie.

Some of the images included in the notes were supplied as part of the teaching resources accompanying the text books listed on the previous slides.

How to be successful

- Understand the material, don't just memorize it.
- If you fall behind, try to catch up as fast as possible.
- Attempt the workshop tasks every week. You should attempt the theory/tutorial questions before you attend your workshop.
- Check the LMS for announcements and discussion board posts.

Discuss the following questions:

- What is an Operating System?
- You have just finished writing a C program, and you want to run the program. What are the different steps that occur?
- When you use a browser to visit your favourite web site, what mechanisms are involved in transferring data between your local device and the source?
- What is a digital signature? How is it used?

We will discuss your answers further in the workshops over the coming weeks.

What are the Internet and WWW?

Neither the Internet nor the WWW is a computer network.

Simple answers:

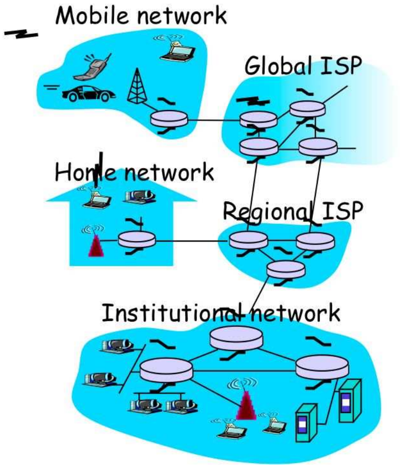
- The internet is not a single network but a network of networks!
- The WWW is a distributed system that runs on top of the internet

A related question:

- What are network services?

  One possible answer – network services are installed on one or more servers to provide shared resources to client computers.

Nuts and bolts view of the Internet



Nuts and bolts view of the Internet

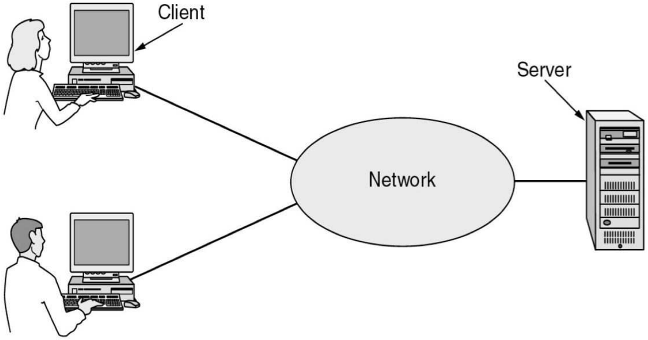
- Millions of connected computing devices
  - hosts = end systems, running network applications
- Communication links: fiber, copper, radio, wireless, satellite
  - transmission rate = bandwidth
- Routers: forward packets (chunks of data)
- Protocols control the sending and receiving of messages
  - e.g., TCP, IP, HTTP, Ethernet
- Internet standards
  - RFC: Request for comments
  - IETF: Internet Engineering Task Force

Service view of the Internet

- Communication infrastructure enables distributed applications:
  - Web, VoIP, email, games, e-commerce, file sharing
- Communication services provided to applications:
  - reliable data delivery from source to destination
  - "best effort" (unreliable) data delivery
- Client-Server model
  - the client host requests a service, it then (hopefully) receives service from always-on server
  - e.g. Web browser/server; email client/server

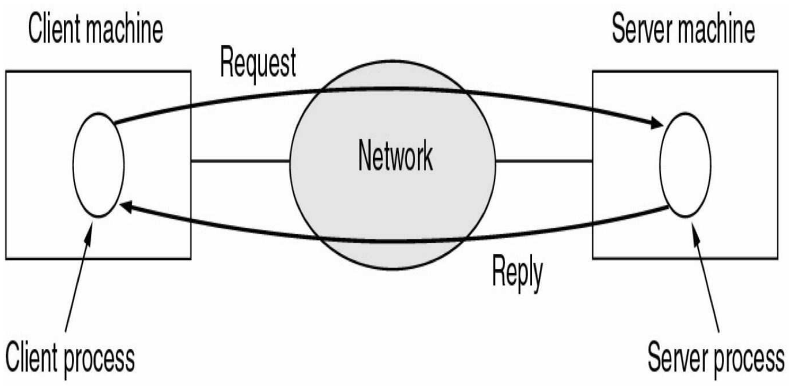
... more details to come (starting in Week 7)

A simple Client-Server Model



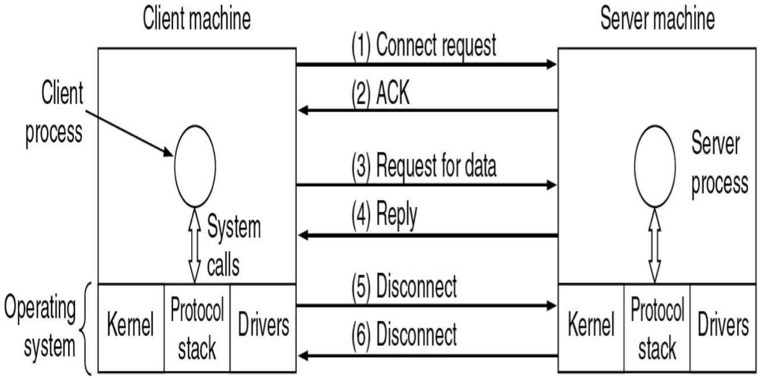
A network with two clients and one server.

Requests/Replies in a Client-Server Model



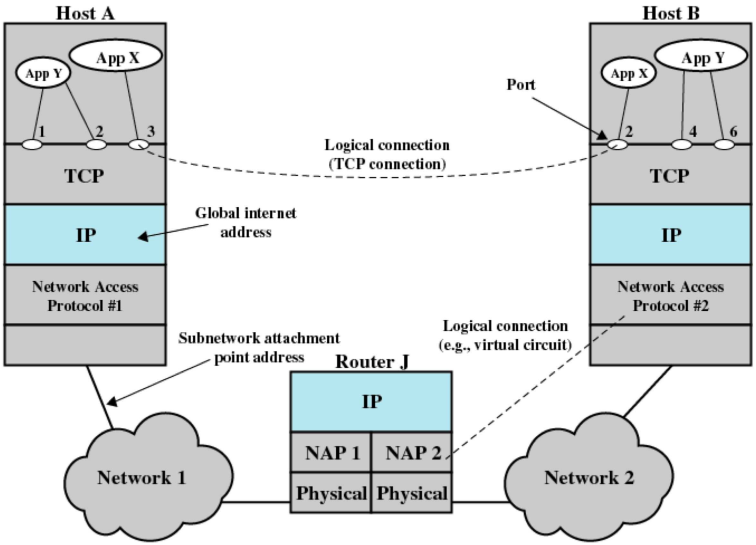
The client-server model involves requests and replies.

Simple Client-Server Interaction



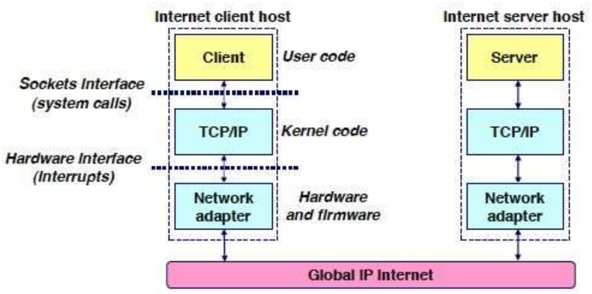
Packets sent in a simple client-server interaction on a connection-oriented network.

Network Services – using TCP/IP


Host A
App Y
App X
1
2
3
TCP
IP
Network Access Protocol #1
Global internet address
Logical connection (TCP connection)
Port
Host B
App X
App Y
2
4
6
TCP
IP
Network Access Protocol #2
Subnetwork attachment point address
Logical connection (e.g., virtual circuit)
Router J
IP
NAP 1
NAP 2
Physical
Physical
Network 1
Network 2

Socket programming

When sending message from one process to another, the message must traverse the underlying network. A process sends and receives through a **socket** – in essence, the doorway leading in/out of the "application"



Main functions of an operating system

- *OS as extended machine*
  Reading data from a file using raw hardware facilities is cumbersome: you have to know on which sector of which track of which platter of which disk the data resides, what you need to do to give the disk device commands and get its status, etc.
  The OS provides a much higher level interface: just say which bytes of which file you want.
- *OS as resource manager*
  You don't want two people, or even two applications running for the same person, to decide simultaneously that they will use the same sector on a disk for two different purposes.
  The OS plays referee, preventing people from stepping on each other toes, either accidentally or deliberately.

Resources

The operating system manages several physical resources, typically including processors, main memory, disk and tape drives, keyboards, screens, mice, clocks and network connections.

The OS also extends or transforms some of these resources to make them more useful. For example,

- the CPU scheduler provides the illusion that multiple programs can be run on one CPU at the same time,
- virtual memory provides an address space that is larger than the size of the physical memory, and
- the file system allows programmers to think in terms of files rather than sets of sectors on a disk.

Over time, operating systems have provided and managed more and more resources, both real and virtual.

Developments: 1950s and 1960s

Computers were much more expensive than humans, so the aim of OSs was to maximize hardware utilization.

Initially, each programmer had the computer exclusively to himself (rarely, herself) for a booked slice of time (typically 30 minutes). He loaded his program by toggling switches on the front panel, ran it, gave it input, looked at the output, and usually debugged it. Programs had to control the hardware directly.

Monitors automated the cycle of *load program* / *run program* / *print output*, and included libraries of functions for accessing the hardware. Programs were still run one after another.

Multiprogramming allowed more than one program to be loaded into memory, so that when one program started an I/O operation and waited for its result, the CPU could switch to executing another program. This avoided idling the expensive CPU.

Developments: 1970s and 1980s

As the cost of computing power started to fall, the primary aim of OSs became making programmers and users more effective.

Multiprogramming mainframe OSs required users to submit jobs as decks of punched cards. The jobs were run in batches; users got their output back in a few hours.

Timesharing gave users interactivity: they could type commands on keyboards and get responses back on the attached screen only seconds later.

One machine supported dozens (sometimes hundreds) of simultaneous connected users. Services were sometimes deliberately limited (e.g. no full-screen editors for students) to avoid overwhelming the machine.

Virtual memory and other services (e.g. the first file systems) made programming easier.

Developments: 1990s and 2000s

Computers became cheap enough to dedicate to one user or function. Text based terminals talking to a shared mainframe were replaced by individual PCs with graphical displays and mice.

Command line interfaces were in many cases replaced by GUIs, making computers more accessible to non-experts.

Networks connected machines together, at first only within workgroups (e.g. to file and print servers), but later to the Internet. This made security much more important, much harder to attain, and harder even to define: consider e.g. viruses, spyware, adware, pop-up blockers and DRM (Digital Rights/Restrictions Management).

The most recent developments are support for multimedia (audio/video), and for mobile computing (wireless connections, intermittent connections, low power operations).

Why Unix?

We will mostly use Unix and its variants as the source of examples, not Windows, even though most computers run Windows. The reasons are these.

- There is far more information available about the implementation of Unix variants (most of which are open source) than about the implementation of Windows (which is owned by Microsoft).
- Unix is much more stable. Most of the system calls from the late 1970s are still unchanged, and only a relatively small number of new ones has been added. Windows has been completely reorganized at least twice since the 1990s.
- The Unix design is much simpler (e.g. it has many fewer system calls) and has *elegance*.

Developments: 2010s and beyond

MS Windows version XXX (current release)

Linux version (current release) – is a Unix-like computer operating system that dates back to early 1990s; open source software (see Ubuntu for further information).

Mac OS X – uses a hybrid structure (a layered system). The top layers includes a GUI and a set of applications environments and services (Coca specifics the API). Below this is the kernel environment (consist primarily of the Mach microkernel and BSD Unix kernel).

iOS – is a mobile operating systems used to run smart phones, iPhone, iPad, structured on Mac OSX with functionality pertinent to mobile devices. A media service layer provides support for graphics/audio/video; core service layer provides additional support.

Developments: 2010s and beyond

Android – runs on a wide variety of mobile platforms; open-source. It is similar to iOS in that it is a layered stack of software. At the bottom is the Linux kernel (modified version by Google). Android runtime environment includes a set of core libraries as well as the Dalvik virtual machine.

Security: key components

Unix

- permissions

Basics of cryptography

- public-key cryptography
- digital signatures

Authentication

- protocols

Web/Internet security

- SSL – secure socket layer
- fire walls
- denial of service attacks

Social implications