Lecture 1

Introduction

12 August, 2011
Instructor

Ooi Wei Tsang

ooiwt@comp.nus.edu.sg
Office Hour
Fri 4 - 6pm
AS6 05-14
MODERN OPERATING SYSTEMS

ANDREW S. TANENBAUM

Required Textbook
Average Weekly Workload
(your mileage may vary)

- Lecture: 2 hr
- Tutorial: 1 hr
- Lab: 1 hr
- Preparation: 6 hr

Note that NUS officially lists the workload as 2-1-1-0-4 which is a typo (it does not add up to 10!)
Assessment

- Midterm: 20%
- Lab: 30%
- Exam: 50%
Important Dates

October 2011
Su  Mo  Tu  We  Th  Fr  Sa
  1
2  3  4  5  6  7  8
9 10 11 12 13 14 15
16 17 18 19 20 21 22
23 24 25 26 27 28 29
30 31

midterm

November 2011
Su  Mo  Tu  We  Th  Fr  Sa
  1  2  3  4  5
6  7  8  9 10 11 12
13 14 15 16 17 18 19
20 21 22 23 24 25 26
27 28 29 30

final
midterm and final are
semi-open book
(one 2-sided A4 sheet)
Lecture Format

1200-1400

- 45 minutes Lecture
- 5 minutes Break
- 45 minutes Lecture
- 25 minutes Dismiss

9
slides will be posted 1-2 days before lecture but
no lecture notes will be provided
students are expected to take notes during lecture
students are expected to read the assigned readings
no “model” answer will be posted
your responsibility: check for update frequently
(hint: subscribe via email or RSS)
do participate in online discussion

(and use your real name!)
screencast
will be posted
but expect 3-4 days delay and technical glitches do occur

(not a good reason to skip lecture)
pre-class activities
simple activities for you to do / think about before attending lecture.

might help improve your understanding in class

online discussion only
tutorials
a set of questions asked each week
you are expected to think through the answers before the attending the tutorial sessions
we will discuss your answers during tutorial sessions

we will conclude each discussion with the correct / best answer
labs
one lab exercise
(almost) every week
can do it at your own time

some are ungraded
lab sessions:

1. lab TAs available for assistance

2. discuss lab answers from past labs
some lab and tutorial questions are meant to let you discover new knowledge yourself

(only if you think through the answers)
warning: I am brutal in penalizing students who do not following instructions exactly for lab submissions
feedback

what your seniors from 2010 think of CS2106
31%

find CS2106

“Very Difficult”

Average is 14% for other 2000-level modules
lots of stuff to learn

“The scope of the module is quite large.”

“Too much content in too little time.”
the labs are difficult

“.. lab sessions can sometimes be really difficult hence time consuming.”

“The weekly labs can be quite stressful.”
but useful for learning

“weekly labs ensure that students really understand the concepts introduced in the lecture.”

“Labs are tough, but it is through the labs which I feel I learnt the most from”

“However the labs are also the real place that we actually learn”
independent learning

“The teaching material and most importantly style, is very conducive to independent learning”

“Also get chance to acquire independent learning through reading man page and google search”

“strengths: encourage a LOT of self-study”
what is CS2106 about?
NOT about how to use Mac OS X, MS Windows, Linux etc.
about basic concepts and design principles in OS
many different variations: for different OS and different architecture
but same concepts and principles
why should I learn OS?
I am not going to write another OS!
CS2106 is important because
complex software
abstraction +
interface design
concurrency
resource management
understand
performance issues
#define SIZE 10000
int a[SIZE][SIZE];

for (i = 0; i < size; i++)
    for (j = 0; j < size; j++)
        a[i][j] = 0;
#define SIZE 10000
int a[SIZE][SIZE];

for (i = 0; i < size; i++)
    for (j = 0; j < size; j++)
        a[j][i] = 0;
My computer is slow. Should I upgrade to

A. a faster CPU
B. more CPU core
B. a faster harddisk
C. a bigger harddisk
D. more memory
E. faster memory
what to learn from OS course
(beside OS):

1. complex systems
2. abstraction + interface design
3. concurrency
4. resource management
5. performance issues
Assumed Background
UNIX and C
why UNIX?
(Linux, Mac OS X, Sun OS etc.)
need a **concrete example** for the concepts and principles taught in CS2106
many OS concepts are **cleanly** manifested in UNIX
source code are available
why C?
UNIX is written (mostly) in C
intermediate-level language
(e.g., explicit memory allocation, bits manipulation)
CS2100
Computer Organization
how a program is executed

a brief review
to build and run a program:

pre-process → compile → link → load

foo.c → foo.o → a.out

memory
Loop:

1. fetch instruction located at PC
2. decode instruction
3. fetch data
4. execute instruction and update PC
int main()
{
    int x = 1;
    foo(x);
    x = x + 1;
}

int foo(int x)
{
    :
    :
}

int main()
{
    int x = 1;
    foo(x);
    x = x + 1;
}

int foo(int x)
{
    int y = x + 1;
    bar(y);
}

int bar(int x)
{  :  
}
CPU

- frame pointer
- stack pointer
- program counter

Memory

- stack
  - return address
  - saved frame pointer
  - local variables
  - function parameters

- data
- code
OS
Operating Systems
The **OS** is a layer of software that manages processors, storage and I/O devices and provide simple interfaces to the hardware to user programs.
OS

is everywhere
phone, car, robot, router, media player, game console, ..
consider the simple program:

1. read a number from a storage
2. print the number to screen

how to code in a world without OS?
is the number stored on a CD, thumbdrive, harddisk..?

location of the number on the storage?

is another program writing to the number at the same time?
what graphics chip is the system using?

what is the display resolution?

is another process displaying something at the same location?
OS hides all these details from programmers
x = read_number(" file.txt ");
print(x);
OS
as an extended machine
interfaces provided by OS are known as **system calls**
a bit in the program status word (PSW) keeps track of the current mode (user or kernel mode)
a system call is similar to a procedure call except:

1. a special instruction sets the kernel mode bit in PSW before executing the system call
2. a special instruction sets the user mode bit in PSW after executing the system call
3. CPU executes the OS “system call handler” for a given system call

    (more details later..)
CPU

- frame pointer
- stack pointer
- program counter
- program status word

Memory

- stack
  - return address
  - saved frame pointer
  - local variables
  - function parameters
- data
- code
- kernel
In **user** mode, certain privileged instructions cannot be executed, certain addresses cannot be accessed etc.

In **kernel** mode, there is no restriction.
operating system (a.k.a. kernel)

user mode

browser
compilers

calendar
editors

..
media player
.. shell

..

file
display
keyboard
mouse
printer
battery
socket

kernel mode

operating system (a.k.a. kernel)
OS as a resource manager
suppose that

the computer runs one task at a time, always completing it before running another task?
a task always have full use of all resources.

not efficient since not all resources are fully utilized at all time (e.g., CPU is idle when I/O is performed).
suppose that

the computer keeps multiple tasks in the memory. When the running task is idle, switch to another task (multi-programming)
now, resources are shared among the tasks.

how does CPU switch from one task to another?

how to prevent one task from corrupting the memory of another task?
what if there are multiple users using the system, and there is one CPU intensive task?
The computer keeps multiple tasks in the memory and switch between them frequently (regardless of whether the task is idle) (time-sharing)
time-multiplexing: CPU, printer

space-multiplexing: memory, disk, screen
OS is an extended machine and a resource manager