

Personnel

Lecture 1

Introduction to Operating Systems

In this introduction, we will see why operating system is an indispensable piece of software that should be carefully studied.

Text:

- **Operating Systems Concepts** [OSC], 6th Ed. Silberschatz and Galvin. John Wiley and Sons Inc.

References:

- **Understanding the Linux Kernel** [ULK]. Bovet & Cesati. O Reilly.
We will use "OSC" and "ULK" to refer to these books from now on.

POS(0230A)

Students: CSIS and BBA(IS)/BEng(SE) double degree

Lecturer:

- Dr. Isaac To Kar Keung (kkto@csis.hku.hk), CYC407(2859-2173).

Tutors:

- Cheng Lok Lam (lcheng@csis.hku.hk).
Consultation hour: Tue 1500–1600.
- Felix Cheung Kai Man (kmcheung@csis.hku.hk).
Consultation hour: Tue 1100–1200.
- Zhang Ming Hua (mhzhang@csis.hku.hk).
Consultation hour: Tue 1400–1500.

Web site: <http://www.csis.hku.hk/~c0230a/>

POS(0230A)-1.1

Calling for help

If you have difficulties, you can do one of the followings:
In the order of preference...

- Post a news article to our **newsgroup**, hku.csis.CSIS0230A. This is preferred: all of you can answer, and all of you benefit from the answer. Questions asked in any other means may be answered (only) in newsgroup, so **please keep watching the newsgroup**.
- Send us a **mail** at c0230a@csis.hku.hk. The mail will end up in the mailbox of me and all tutors.
- Contact us during **tutorials**.
- Contact us directly. Send a personal mail before visiting our office to make an **appointment**.
This is the order in which I deal with them. I.e., news are first cleared, and only when all news are dealt with I'll deal with mails to c0230a, and only after that I'll look at personal mails.

POS(0230A)-1.2

Objectives

- What is an Operating System? What services it provides?
e.g.: What is a process?
How I can make different programs interact?
- Learn the theory in designing the Operating System.
e.g.: Why I need to mount my drives?
Should I allocate memory if it is typically not used in 2 hours?
How to prevent programs from waiting for one another forever?
- Gain solid knowledge about a practical OS, namely Linux.
Other OS are similar, and it is better to learn deeply into one rather than learn superficial things in many. We just learn one which is most available.
e.g.: How to modify the kernel to suit my needs?
How to make effective use of protection bits?
What are those files in `/proc`?

POS(0230A)-1.3

Mode of instruction

Lectures

- 24 hours (Every Monday, 1600–1800). Voluntary.
- Primarily uni-directional flow of information. I teach, you listen. There is a little room for you to participate, though.

Small Group Tutorials

- Provides supervised lab experience in 13 hours. **Compulsory**.
But many find that they overrun tutorials, so be prepared to have 26 hours here.
- **Study the reading material well** before going to tutorials.
They will be available a week in advance of each tutorial.
- Indicate preference to the time for tutorial slot after this lecture.
But since tutorials can't be large, expect to make compromises, i.e., you might get a slot on an otherwise off day.

POS(0230A)-1.4

Lecture conduct

Lectures are free-form, relaxed but serious.

- **Talking** is allowed **only on course matters**.
- **Attend lectures** only if you are willing to learn. Remember that lectures are voluntary.
- **Everything** about the lecture **needs to be understood**, preferably during the lecture, and absolutely no later than the end of the day.
- **Ask** whenever you find something you don't understand. Don't delay until the end of the lecture, or even worse until an assignment comes.
- **Don't worry** that you're asking a stupid question. Even the most stupid question is welcome during lectures.
If you ask after lecture, you will probably be scolded for your stupidity. =)
- **Ask me to stop or repeat** if you find something you need time to digest.

POS(0230A)-1.5

Assessment

Assignments (30%)

- 4 programming assignments, in Groups of 1–2.
- Don't ask us to debug your programs, but you can ask any question.
Well... debug at most once per group per assignment.

Quiz (10%), To be announced

- 30 min quiz of very easy questions. No problems if you have done the assignments and understands the tutorials well.
This is perhaps a big if, though.

Tutorial assessment (10%) during 13 weeks

- Subjective assessment by TAs and lecturer.
- No more than one absense. Each further absense results in a loss of 1/12 of all 50% coursework marks.

Exam (50%)

POS(0230A)-1.6

Policy: late submission, plagiarism

Late submission

- Requests for postponing deadlines must be made **at least 3 days** before deadline.
- Assignments late for at most **48 hours** will **score 80%** of the score it would score if it was submitted on-time.
- Assignments late for more than 48 hours are marked as normal, but will not contribute to the final score.

Plagiarism

- Any discussion within group are allowed.
- Discussions across group or to outside groups are limited to sharing the concepts. No actual code or numbers may be exchanged.
- Plagiarism is **punished as described by departmental guidelines**.

POS(0230A)-1.7

Advice on learning

- **Learn the concepts first (most important).** Implementation details are not as important as the concepts.
We usually show code for illustration purpose. But if you are not familiar with computer code, it is okay to skim them.
- **Understand, not just memorize.** Find the underlying ideas that give rise to concepts. Find unifying themes.
- **Keep up with lectures.** Each lecture builds on top of the previous ones, so if you miss just one lecture, it will be significantly more difficult to catch up with upcoming lectures.
- **Do the assignments early.** Programming assignments are usually more difficult than what you first think. And assignments clear up your concepts for easy understanding of the coming lectures.
- **Read the book.** Lecture is too short to explain everything in the book, and the function of lectures is simply to guide you to read.

POS(0230A)-1.8

Lecture schedule

IGNORE MODE ON

You probably won't understand all these now anyway.

- Wk 1: Introduction
- Wk 2: External interface
- Wk 3: Internal structure
- Wk 4: Process
- Wk 5: Process communication
- Wk 6: (Holiday)
- Wk 7: SMP and deadlocks
- Wk 8: Scheduling
- Wk 9: Addressing, Quiz
- Wk 10: Virtual memory
- Wk 11: I/O
- Wk 12: Filesystems
- Wk 13: User protection

IGNORE MODE OFF

Now, back to "understand everything during lecture".

POS(0230A)-1.9

Why OS: Overview

- In most modern computers, there are **multiple programs** running at the same time, on behalf of **different users**.
- There are many tasks that can only be done efficiently by **having all programs cooperating**.
- Typically, such tasks involve the different programs **sharing** the limited **resources** of the computer.
- **If one program does not cooperate, the system fails completely.**

An OS allows such tasks to be done:

- It acts as a **government** to make sure that the programs cooperate, preventing misbehaving programs from affecting well-behaved ones.
- It enables such cooperation in a **efficient** and **convenient** manner, so that such restriction does not limit the usefulness of computers.

POS(0230A)-1.10

Why OS Part 1: Resources management

- How resources can be shared among different users and programs?
- Basically, programs **never directly use** the physical resources.
- Instead, the OS enforces some **conventions**, and make out "logical resources" from physical resources.
- Each program is the sole owner of the logical resources, which is **mapped** to physical resources to each program in turn.
This also makes it convenient to use the resources: program writers don't have to consider other programs that also need the resources.
- OS **makes sure** that only logical resources are used, and that physical resources can be shared among logical resources.
- **Some such resources:** CPU, Input and output devices, memory, real-time clock, network, etc.

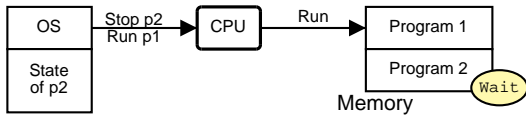
POS(0230A)-1.11

Example

How multiple programs can safely **share one CPU**?

Convention: each program runs only a small fraction of a second each time, and then give up the CPU for other programs to use.

If one program don't give up the CPU, the system fails!



The OS **controls** the CPU, and switch it between processes. It saves and restores states before switching processes.

The program sees a **virtual CPU**, one which **always** runs the program. Of course, the virtual CPU runs for a while, stops for a while, and repeat...

POS(0230A)-1.12

Why OS Part 2: Convenience

- To things simple enough for those building computer, the computer usually provides resources that are **too primitive** for the users. Why? Simple things can be make fast easily at hardware level. Computers tends to run faster if complexities are in the OS and libraries rather than hardware.
- The OS sits between the machine and the user programs, providing a **conceptual view** of the resources that is convenient to use.
- Some such concepts: **file, process, terminal, network connection**.
- For safe sharing of resources, programs mostly seem like running **by itself**. But sometimes programs want to interact. If sharing is unsafe, programs easily achieve interaction, but so easily that unwanted interactions can occur easily as well.
- OS provides ways for interactions to take place safely, under control.
- Some such interactions: **shared memory, messages, signal, etc**.

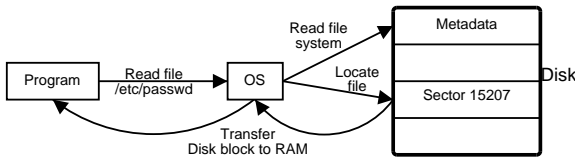
POS(0230A)-1.13

Example 1

The disk is a device consisting of blocks, identified by numbers.

It is very difficult to memorize all these numbers. It is more convenient if storage is organized as resizable files containing related data.

It would be very **difficult to share and manipulate data** if each program have their own way to map from numbers to filenames .



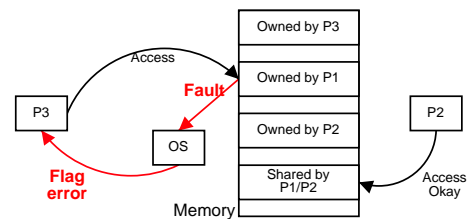
The **OS organizes the disk into a filesystem**, and the user programs runs as if the computer provides such a convenient system of storage.

POS(0230A)-1.14

Example 2

At times **programs running in the same computer need to interact by sharing some memory**. E.g., a program calculating the position of an atom might need to write to the memory of a visualization program.

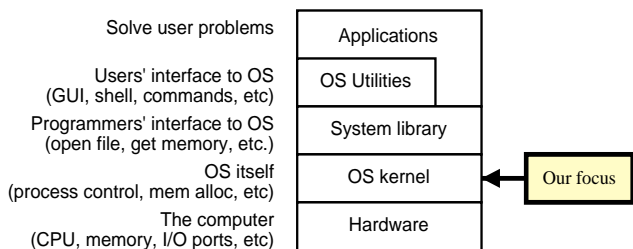
However, we only want **some, not all, memory** to be **shared** like this.



OS establishes a distinction between different programs, and maintain interfaces so that they interact only in predefined ways.

POS(0230A)-1.15

Typical structure of OS



This shows how each components *rely* on each other, e.g., OS kernel uses the services provided by the hardware.

Our focus will be in the OS kernel, a program that keep running all the time to provide the OS services to user program through the system library.

POS(0230A)-1.16

Overview of services provided by an OS

- **Process management:** create, delete and schedule processes
- **Process communication:** create and delete channels, read from and write to channels, synchronize processes, etc
- **Memory management:** allocate, configure and deallocate memory, put memory content to disk temporarily, load them back as needed, share memory among processes, etc
- **User protection:** identify users, checking capabilities, etc
- **Filesystem:** Map file requests to disk I/O requests, maintain a cache to improve performance, delay writes until disk is idle, etc
- **I/O device handling:** Read from and write to I/O devices, handle events from devices, allocating memory for devices, etc.
- **Network management:** Establish connections, route messages, etc.

POS(0230A)-1.17

Example OS usage

Suppose we write a program to copy everything from standard input to standard output...

```
#include <iostream>
using namespace std;

int main() {
    while (cin.peek() != EOF)
        cout.put(cin.get());
}
```

Then we run our program like this:

```
> a.out < myfile > copied_file
```

Where we used the OS? What is its importance?

Answer: *the OS is used extensively.*

POS(0230A)-1.18

POS(0230A)-1.19

Example OS usage (cont'd)

- An OS utility called the **shell** reads the command we typed.
- At the beginning, when we run "a.out < myfile > copied_file", a new **process is created by the OS**, still running the shell program.
For now, process = the place where a program runs (e.g., contain virtual CPU).
- The newly created process uses the OS to **open the files** myfile for reading and copied_file for writing.
- The newly created process uses the OS to **load and run the program** a.out.
- The old program (also running the shell program) uses the OS to **wait for the completion of the newly created process**.
- After the new process starts running a.out, it calls the OS to **allocate memory** for the program to use, in particular for *cin* and *cout*.

Example OS usage (cont'd)

- The new process finally starts running our code. It uses the OS to **read** the standard input.
- This will need some time, for reading the disk. The OS **block the new process**, i.e., sends it waiting until the data is ready. The CPU is reallocated to other processes.
- When disk data becomes available, the OS **wakes up** our process, so that it continue executing, sending the data to disk.
- Since writing to disk is slow, the OS **caches the data** instead of writing it to the disk immediately. The cycle repeats until all data are read.
- Our program tells the OS that it is done. The OS **collects all resources used by the program**, and **wakes up** the waiting shell program.
- At some time later, the OS **invisibly writes** the data back to the disk to free up the memory used by the cache.

POS(0230A)-1.20

A quick historical overview of OS

- **Early systems:** Only acts as program loader and device driver.
- **Simple Batch System:** Allow multiple programs to be waiting for execution. Load and setup programs when another is executing.
- **Multiprogrammed Batched Systems:** Load multiple programs, so that when one do I/O, another program is executed to keep the CPU busy. This is a big change: most of the OS concepts arises directly from **running multiple programs at the same time**.
- **Time-Sharing Systems:** Interrupt jobs involuntarily to create an illusion of concurrency. Implemented user separation and multi-processing. Many OS concepts need to be refined.
- **Desktop Computer:** Reduce the complexity by removing user separation and multi-processing. *Only temporary effect*. Everything removed (multi-user, multi-tasking, multi-processing, etc) are eventually pulled back when the web and SMP machines becomes affordable.

POS(0230A)-1.21

New trends

OS designs must change to meet new challenges.

- **Parallel Systems:** Use a large number of closely interacting CPUs to speed up complex computations. *Too expensive to be effective*.
- **Real time Systems:** Deals with tight timing constraints of tasks when a delay cannot be tolerated. *Need different scheduling methods*.
- **Distributed Systems:** Sharing information over a computer network, in a way that hide the underlying network topology. Main challenge: *how to maintain reliability and efficiency*.
- **Clusters:** A special distributed system to speed up computations by many computers. *Cheapest way to compute large problems*, but only suitable for those problems that can be "partitioned" easily.
- **Hand-held Systems:** Reduce a computer to pocket size. Once again remove most of the functionality of the OS to fit the tight constraints.

POS(0230A)-1.22

Some popular OS

- Early research work: **Multics**. Never really there, but all the great ideas of multitasking, multiprocessing and multiuser come here.
It is so complex that development never finishes before it is abandoned.
- Unix workstation and server: **Solaris, FreeBSD, MacOSX**. Stripped Multics so that it can be reasonably implemented.
- Unix-like OS: **Linux, Mach/GNU Hurd**. Make Unix free.
- Desktop "OS": **DOS, Windows 98/ME**. "Library" is probably a better word than "OS", as protection is real weak.
- Microsoft server OS: **Windows NT/2000/XP**. A real OS that looks a bit like DOS and Windows.
- Distributed OS: **Amoeba, Sprite**. Designed primarily for research.
- PDA (hand-held) "OS": **PalmOS, Windows CE**. Again, sort of "library".
The power-saving CPU does not provide most protection mechanisms.

POS(0230A)-1.23