Operating Systems Lecture 6: OS Introduction (Part 6): The OS is a Resource Manager

Intro

So now we will go quickly through the different kinds of resource management that operating systems do. Why?

Process Management Activities

The operating system is responsible for the following activities in connection with process management:

- Creating and deleting both user and system processes — this is something that we will be discussing in chapter 3
- Suspending and resuming processes — chapter 3
- Providing mechanisms for process synhronization — chapter 5
- Providing mechnisms for process communication — chapter 3
- Providing mechanisms for deadlock handling — discussed in chapter 7

These are the acitivities that the OS does.

Memory management

To execute a program, all (or part) of the instructions must be in memory

- All (or part) of the data that is needed by the program must in the memory
- Memory management determines what is in memory and when
- Memory management activities:
  - Keeping track of whic parts of the memory are currently being used and by whom
  - Deciding which processes (or parts) and data to move into and out of memory
  - Allocating and deallocating memory space as needed

We will be spending 2 weeks on memory management trying to understand how an OS manages memory. How it allocates memory to user processes and how it manages that.

Storage management

How the OS manages the storage device — how it manages the disk, and how it provies the file system interface. If you have a text file with line 1, line 2 and line 3, it will look like a nice text file with some human readable lines on it. But on the device, it may not look like this. It may not be one contiguous piece of data. It could be stored on the device in different pieces or different blocks of data. And maybe one block could be somewhere in the middlle oe line#1. And the next block coud be physically stored in a totally different place. But to you, when you look at the file it will look like a nice text file which is contiguous and human readable and everything.

Performanc of various levels of storage

There are different levels of storage in the system.

Level	1	2	3	4	5
Name	register	cache	main memory	solid state disk	magnetic disk
Typical size	$<$ 1 KB	$<$ 16mb	$<$64GB	$<$ 1TB	$<$ 10TB
Implementation technology	custom memory with multiple ports CMOS	on-chip or off-chip CMOS SRAM	CMOS SRAM	flash memory	magnetic disk
Access time (ns)	0.25–0.5	0.5–2.5	80–250	25,000–50,000	5,000,000
Bandwidth (MB/sec)	20,000–100,000	5,000–10,000	1,000–5000	500	20–150
Managed by	compiler	hardware	OS	OS	OS
Backed by	cache	main memory	disk	disk	disk or tape

Movement between levels of storage hierarchy can be explicit or implicit. Registers can be accessed typically in the fraction of a nanosecond. And when we say that, what are we implying about the speed of the processor? Or in other words what clock speed corresponds to one nanosecond.
1 nanosescond = 1GHz

Also
1GHz = 10^9Hz in one second
which equivalent to
10^{-9}s /cycle
which is:
1 ns/cycle (or one nanosecond per cycle).

This means that if a processor does 10^9 cycles each second then this means that each cycle is one nanosecond.

Access to register

Usually access to register is done in one cycle. You cannot access anything in less than one cycle, so this is saying that the cycle ranges from 0.25 to 0.5 nanoseconds. So 0.25 nanoseconds here corresponds to 4GHz, we do it like this:
0.25 nanoseconds = 1GHz

25/100 = 1GHz
100Ghz/25 = # of gigahertz
answer : 4GHz

So if each cycle is a quarter of a nanosecond then the clock speed is 4 gigahertz. So, you can access the register in one cycle.

==========

Access to cache, main memory and sold state, and magnetic

==========

Then you can access the cache in a couple of cycles (usually 2 or 3 cycles) but if it is L2 or l3 cache then that may take longer. So if it takes 25 nanseconds then on a 4 gigahertz process will be a 100 cycles and that is probably an L3 cache. Access to main memory can take hundreds of cycles. Solid state can take tens of thousands and a magnetic disk can take million of nanoseconds or millions of gigahertz. So they vary in terms of speed. It is good to have a good estimate of how fast each device is. And of course we will, when we get to disk management, we will talk more about this.

The OS is responsible for managing main memory and disk. That is why, in this course, we have spent two weeks on memory management and two weeks on disk management. But we will not spend any time on register management and cache management becuase cash management is done by the hardware and register management is done by the compiler.

========

Cache management (by hardware)

==========

We understand that the OS manages the hardware. Anyways, deciding what to put in the cache (or updating the cache) is the job of the hardware. Also, what is the main decision that is being made in the cache management besides the architecture of the cache — how much cache we have at each level, and the hardware architecture of the cache itself. There is a very important decision that the hardware must make. What is it? It has to prioritize what data it needs to store in the cache versus having to store it in other things. And how does it prioritize this? Does it go through main memory and say that this is higher priority piece of data I'm going to put it in the cache. Does it like scan main memory and choose what to put in the cache? It might be knowing the frequency that it is used or knowing how recently it has been used? Does it do an analysis? Like does it sound like something a hardware would do? Let's think realistically... what can a hardware do. It is not going to scan the main memory and say that this is an important piece of memory and put it in the cache. What does the hardware do? It does this IMPLICITLY. It does this without actually scanning main memory.

========== 9:11

How does caching work

==========

Hardware cannot do sophisticated analyses of data. It is just that the hardware will start with an empty cache. Nothing is in the cache. It is a cold start. Then when a process starts loading or accessing memory location, whenever a memory location is accessed, the hardware is going to check the cache. If it is in the cache, it will be used. If it is not in the cache, the hardware is going to load it from the main memory into cache. Now as long as there is room in the cache, there is no issue. The hardware is just goint to load it from the main memory into the cache. But we have an issue, when? When the cache is full. So that is when the hardware will have to make a decision. It will have to decide what is in the cache. Now the cache is full and I need to load. Now what is the solution? There is only one solution. I will have to victimize one of these blocks or lines of cache in the cache and I have to override them with what I am loading. So, now I need some kind of replacement policy. So this replacement policy — so do we remember one good cash replacement policy?

==========

LRU policy

==========

LRU stands for "Least recently used". This is review material. But the point is that the hardware decides the replacement policy and the operating system has nothing to do with it.

========

Register management (by compiler) – (also discussing how the overlapping of multiple variable's live ranges affect the register allocation)

==========

If you have a program that defines a thousand variable:
`int x1; int x2; .....  int 10000;`

There are 10,000 vvariables in our program. Now when the compiler translates a program written in a high level language into machine code, the machine code doesn't understand variables. The whole notion of a variable is not there in the hardware. What the hardware knows are registesr and memory locations. So any piece of data should be in a CPU register or in the memory. The hardware doesn't understand the concept of a variable. So the compiler's job is to map these program variables into memory locations or registesr. And what is the preferred device? Registers, because they are faster. So hopefully, the compiler can map all these variables into registers. Now, it has a limited number of registers. Like for example, 32 registers on a hardware. So the compiler has 10,000 variables annd 32 registers. So the compiler will have to findd a good way to map these. It may even be able to map 10,000 registers onto 32 registers because not all variables overlap. If two variabes overlap in their live ranges then they cannot be mapped into the same register. But if they do not overlap (like you define a variable and then you are done with it and then you start another variable). Then these two variables can be placed in the same register. So solving this problem of mapping program variables into CPU registers or mapping as many variables into as many CPU registers as possible is the job of the compiler, in what we call "REGISTER ALLOCATION". So this is what we mean by saying that the compiler controls the registers. So register allocation is not an OS topic.

==========
I/O Subsystem
==========

We will be talking about I/O and how the Operating System tries to overlap CPU execution with I/O requests but we unfortunately in this course, we will not get to cover the internals of I/O devices. But we will understand basically the interface of I/O devices. Or we will basically understand how the OS interacts with an I/O device. And in-fact, we already have a reasonable understanding of how an OS interacts with an I/O device but we will not be studying the I/O subsystem itself.

=======
A bit more info on I/O subsystem
=========

- One purpose of OS is to hide peculiarities of hardware devices from the user.
- I/O subsystem is responsible for:
  - General device-driver interface
  - Drivers for specific hardware devices
  - Memory managment of I/O

==========
Kernel Data Structures
==========

It is very important to keep in mind that the OS is just a computer program. It is a special kind of program. It is a complex program. It is a program that controls the whole machine, but it is a program. And what is a program? A program is basically a combination of algorithms and data structures that are expressed using some programming language. That is what alll programs are. And the more complex a program is, the more complex the algorithms and the data structures that the program uses will be. So more complex programs use more complex algorithms and data structures. And since the Operating Systme is a very complex program and since it does many different and many challenging resource management tasks, it will be using many sophisticated algorithms and data structures. Ofcourse this depends on the OS itself. But there may be a certain data structure that one operating system implements as a binary search tree while another operating system implements as a hashtable. So different OSs vary in the way they implement certain functionaities or algorithms. They use different algorithms and data structures. Most of the typical data structures that OS is going to use is going to be like: BSTs, Hashtables, Lists,bitmap, and such.

Linux Data structures

Linux data structures are defined in: include files `<linux/list.h>`, `<linux/kfifo.h>`, `<linux/rbtree.h>`

Open source operating systems

These are made available in source-code format rather than just binary closed-source

- Counter to the "copy protection" and "Digital Rights Management (DRM)" movement
- Started by "Free Software Foundation (FSF)", which has "copyleft" GNU Public License (GPL)
- Exampless include GNU/Linux and BSD UNI (including core of Mac OS X), Open Solaris and many more
- Good for academic purposes (education and research).

We just need to be aware that some OS are open source. In theory, we can modify the OS (if we have the expertise of modifying an OS kernel). If it is a closed source OS then you just get the OS executable. Libraries, dynamic linked libraries etc are all executable.

Operating Systems Lecture 7: OS Services

Intro

In this chapter, we will take a look at the inside of an OS and how it is structure. But before we do that, we will take a greater look at the details that the OS provides.

So here we have an operating system sitting on top of a hardware. Standing between the user and the hardware. And between the user programs and the hardware. So these are the services that the OS does. So it does program execution, I/O, file systems, process communication, resource allocation, accounting, keeping track of resource usage. We will talk about many of these in greater details about these later in the corresponding chapters. After this, you have error detection and protection and security.

Now how do user programs access these services via system calls?

A user program will issue a system call to access a certain OS service. So these services include user interface. We already know CLI and GUI. Now here, we assume that we are familiar with the command line and the with the script in "CSC60". This assumes that we are comfortable using CLI and doing some scripting as well.

Example of a CLI system

A vax system with a command line interface is an example of a CLI based sysetm.

==========
Program execution
==========

Program execution is one of the main functionalities of an OS. So when you click on an icon, you are basically asking the OS to execute that program. On the command line, you just type the program name and then hit enter. By doing this, you are asking the command line to execute that program.

==========
I/O operations
==========

These are controlled and they are implemented by the Operating systems and access to the I/O devices is limited to the Operating Systems. We are not allowing user programs to access I/O programs directly. Why? Because of the following reason(s):

i) Easier implementation for application. The application will be easier to write and to implement. To hide the details of the I/O devices from the application programs. When an application program needs to read from the keyboard, the application program doesn't need to know about the specific details of that particular I/O device. It will just say read. And the OS will handle all the low level and hardware level details.

ii) To divide access between processes that may need to use a particular I/O service. To resolve conflicts in other words. If we allow user programs to access I/O devices directly then there will be lots of conflicts. What if two processes are trying to access the same I/O device at the same time? Then there will be conflicts. They may be overwriting each other. And so if we go through the operating system, the operating system is going to manage this and it will resolve conflicts in a way that ensure correct functionality.

==========
Is the fact that OS is the manager as opposed to implementing some kind of actual device itself as the manager an implementation detail?
==========

NO! This is a policy. It is a policy of giving control to the OS. Keeping the OS in control and allowing the OS to resolve conflicts because otherwise application programs will be creating lots of conflicts; they will be conflicting with each other when they are trying to access the devices. It is a very important policy that will keep the OS in control and ensure that the system will work correctly.

==========
Operating system Servicess (Cont.)
==========

The different services that OS offers, it would include File Systems manipulation. Communications, process communication, we will be studying different methods for inter process communication includingg shared memory and message parsing. Shared memory will be shared in detail and we will be using it in Assignment#1. We will be doing that through the Operating System. Message passing will be left for the networking class. Error detection is very important for the OS to be aware of all the erorrs that may occur in the system whether they are user errors or hardware errors or device errors. THe OS should be aware of these and take the right action. Resource allocation, accounting — keeping track of how much of each resource a process has used (especially the CPU, and we will see this in the CPU scheduling).

==========
Detailed notes Operating system Servicess (Cont.)
==========

One set of operating system services provides functions that are help to the user (Cont.):

- File-system manipulation : the file system is of particular interest. Programs need to read and write files and directories, create and delete them, search them, list file information, permission management.
- Communications : Processes may exchange information, on the same computer or between computers over a network.
  - Comunications may be via shared memory or through message passing (packets moved by the OS).
- Error detection: OS needs to be constantly aware of possible errors.
  - May occur in the CPU and memory hardware, in I/O devices, in user program
  - For each type of error, OS should take the appropriate action to ensure correct and consistent computing.
  - Debugging facilities can greatly enhance the user's and programmer's abilities to efficiently use the system.

==========
Anoter set of OS functions
==========

Another set of OS functions exists for ensuring the efficient operation of the system itsellf via resource sharingg:

- Resource allocation : when multiple users or multiple jobs running concurrently, resources must be allocated to each of them.
- Accounting : to keep track of which users use how much and what kinds of computer resources.
- Protection and security: The owners of infomration stored in a mutiuser or networked computer system may want to control use of that information, concurrent processes should not interfere with each other.
  - Protection: Involves ensuring that all access to system resources is controlled.
  - Security : Security of the system from outsiders requires user authentication, extends to defending external I/O devices from invalid access attempts.