CS1101: Lecture 35

The OSM Level: The Operating System and Virtual Memory

Dr. Barry O'Sullivan
b.osullivan@cs.ucc.ie

Course Homepage
http://www.cs.ucc.ie/~osullb/cs1101

Lecture Outline

- What is an Operating System?
- System Calls
- OSM as an Interpreter
- OSM Topics of Interest
- What is a Process?
- Memory Management – Paging
  - An Introduction to Paging
  - Mapping Addresses to Memory
  - The Need for Paging
  - How Paging Works
  - Implementation of Paging
  - Pages and Page Frames
- **Reading**: Tanenbaum, Chapter 6, Section 1.

What is an Operating System?

- An **operating system** is a program that, from the programmer's point of view, adds a variety of new instructions and features, above and beyond what the ISA level provides.
- Normally, the operating system is implemented in software

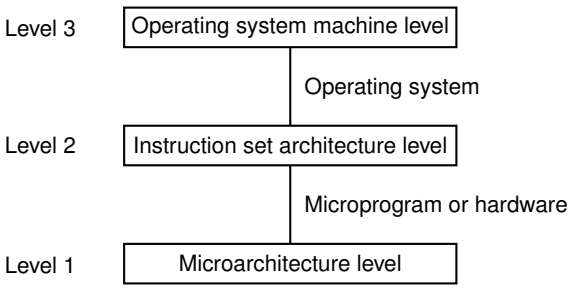

Figure 6-1. Positioning of the operating system machine level.

System Calls

- Although the OSM level and the ISA level are both abstract (in the sense that are not the true hardware level), there is an important difference between them.
- The OSM level instruction set is the complete set of instructions available to application programmers.
- It contains nearly all of the ISA level instructions, as well as the set of new instructions that the operating system adds.
- These new instructions are called **system calls**.
- A system call invokes a predefined operating system service, effectively, one of its instructions.
- A typical system call is read some data from a file.

OSM as an Interpreter

- The OSM level is always interpreted.
- When a user program executes an OSM instruction, such as reading some data from a file, the operating system carries out this instruction step by step – same way as a microprogram
- However, when a program executes an ISA level instruction, it is carried out directly by the underlying microarchitecture level, without any assistance from the operating system.

OSM Topics of Interest

- We will focus on three topics of importance.
- The first is **virtual memory**, a technique provided by many operating systems to make the machine appear to have more memory than it really does.
- The second is **file IO**, a higher-level concept than the IO instructions that we studied in the previous chapter.
- The third and last topic is **parallel processing** – how multiple processes can execute, communicate, and synchronize.
- A **process** can be thought of as a running program and all its state information (memory, registers, program counter, IO status, and so on).

Memory Management

- In the early days of computers, memories were small and expensive.
- In those days the programmers spent a lot of time trying to squeeze programs into the tiny memory.
- Often it was necessary to use an algorithm that ran a great deal slower than another, better algorithm simply because the better algorithm was too big
- That is, a programme using the better algorithm could not be squeezed into the computer's memory.

An Early Solution

- The traditional solution to this problem was the use of secondary memory, such as disk.
- The programmer divided the program up into a number of pieces, called **overlays**, each of which could fit in the memory.
- To run the program, the first overlay was brought in and it ran for a while.
- When it finished, it read in the next overlay and called it, and so on.

The Implications

- The programmer was responsible for breaking the program into overlays, deciding where in the secondary memory each overlay was to be kept, arranging for the transport of overlays between main memory and secondary memory, and in general managing the whole overlay process without any help from the computer.
- Although widely used for many years, this technique involved much work on connection with overlay management.
- A new approach was devised – **virtual memory**.

An Introduction to Paging

- Paging is based on an idea which separates the concepts of **address space** and **memory locations**.
- Consider, as an example, a computer having a 16-bit address field in its instructions and 4096 words of memory.
- A program on this computer could address 2^{16} (65536) words of memory.
- The **address space** for this computer consists of the numbers 0, 1, 2, ..., 65535.
- However, the computer may well have fewer than 65535 words of memory.
- Note that there is a distinction between **address space** and **memory addresses**.

Mapping Addresses to Memory

- The idea of separating the address space and the memory addresses is as follows.
- At any instant of time, 4096 words of memory can be directly accessed, but they need not correspond to memory addresses 0 to 4095!
- We could choose that *address* 4096 refers to *memory word* 0, address 4097 refers to memory word 1.
- In other words, we can define a mapping from the address space onto the actual memory addresses.

Mapping Addresses to Memory

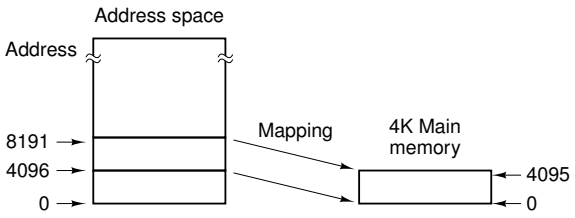

Figure 6-2. A mapping in which virtual addresses 4096 to 8191 are mapped onto main memory addresses 0 to 4095.

The Need for Paging

- Thus, a 4K machine without virtual memory simply has a fixed mapping between the addresses 0 to 4095 and the 4096 words of memory.
- What happens if a program branches to an address between 8192 and 12287?
- On a machine without virtual memory, the program would cause an error trap, give an error and terminate the program.
- On a machine with virtual memory, a sequence of steps would be followed based on **paging**

How Paging Works

- Here are the steps that would be followed:
  - The contents of main memory would be saved on disk.
  - Words 8192 to 12287 would be located on disk.
  - Words 8192 to 12287 would be loaded into main memory.
  - The address map would be changed to map addresses 8192 to 12287 onto memory locations 0 to 4095.
  - Execution would continue as though nothing unusual had happened.
- This technique for automatic overlaying is called **paging**.
- The chunks of program read in from disk are called **pages**.

Implementation of Paging

- One essential requirement for a virtual memory is a disk on which to keep the whole program and all the data.
- We can regard the copy of the program on the disk as the original one and the pieces brought into main memory as copies.
- When changes are made to the copy in main memory, they should also be reflected in the original.
- The virtual address space is broken up into a number of equal-sized pages.

Implementation of Paging

- Page sizes ranging from 512 to 64K bytes per page are common at present, although sizes as large as 4 MB are used occasionally.
- The page size is always a power of 2.
- The physical address space is broken up into pieces in a similar way, each piece being the same size as a page, so that each piece of main memory is capable of holding exactly one page.
- These pieces of main memory into which the pages go are called **page frames**.

Pages and Page Frames

Page	Virtual addresses
≈	≈
15	61440 – 65535
14	57344 – 61439
13	53248 – 57343
12	49152 – 53247
11	45056 – 49151
10	40960 – 45055
9	36864 – 40959
8	32768 – 36863
7	28672 – 32767
6	24576 – 28671
5	20480 – 24575
4	16384 – 20479
3	12288 – 16383
2	8192 – 12287
1	4096 – 8191
0	0 – 4095

(a)

Page frame	Bottom 32K of main memory Physical addresses
7	28672 – 32767
6	24576 – 28671
5	20480 – 24575
4	16384 – 20479
3	12288 – 16383
2	8192 – 12287
1	4096 – 8191
0	0 – 4095

(b)

Figure 6-3. (a) The first 64K of virtual address space divided into 16 pages, with each page begin 4K. (b) A 32K main memory divided up into eight page frames of 4K each.