CS 320
Ch 8 – Operating Systems

An operating system c**ontrols execution of applications and acts as an interface between the user and the computer.**
A typical OS provides the following services:

- Program Creation – **Editors, compilers, and utilities**
- Program Execution – **Load, initialize, run**
- Access I/O
- Controlled access to files
- System Access – **Resolve resource contention. Provide protection**
- Error detection response
- Accounting – **statistics for various resources**
- Instruction Set Architecture - **Assembly language and special instructions used for memory protection etc.**
- Application binary interface – **Defines system call interface to the operating system and hardware**
- Application programming interface – **Gives programs access to the hardware with library calls, etc.**

When an application runs is the operating system is still sort of running**. Things like the system clock continue on an interrupt basis but the op system itself is asleep.**

With regard to operating systems the kernel consists of the m**ost frequently used functions in the operating system and it kept in main memory.**

A batch operating system is a s**ystem that runs a larger group of applications serially without intermediate user input.**

The term multiprogramming means that the p**rocessor appears to run more than one program at a time and does so fast enough to make it seems like they are running simultaneously.**

The earliest operating systems used a monitor to run batch jobs. The m**onitor stayed in memory and loaded a job into another memory area. It then turned control over to that job and recovered when the job terminated or it had an error.**

Monitor can load multiple programs in memory and runs each in sequence or switches among them to increase efficiency.

The hardware features are necessary for a multiprogrammed environment include: **Interrupt I/O and preferably DMA.**

A time sharing system works by allowing the u**ser to interact through a terminal instead of as a batch job. CPU switches between users to make it appear that all users are running simultaneously. More efficient than a batch system.**

With regard to a scheduler, a process is a **program in execution.**

There are four types of scheduling (schedulers?): Long term scheduling, medium term scheduling, short term scheduling, and I/O scheduling.

Long term scheduling **determines which programs are admitted to the system for processing.**

Medium term scheduling m**anages the swapping function as to what data and programs are swapped in from disk.**

Short term scheduling d**ecides which job to do next.**

I/O Scheduling d**ecides who gets to use the I/O device when.**

For the short term scheduler j**obs are assigned a state and a process control block. The short term scheduler keeps track of this data for each job. In general, the short term scheduler runs the job with the highest priority that is in the ready state next.**

The process control block contains p**rocess identifier, current state new, ready, etc.), priority, program counter, memory pointers, context data, I/O Status information, accounting information.**

Context data is **mostly register contents from when a job was suspended.**

Accounting information is the **Analysis numbers such as clock time used, time limits, etc.**

In the short term scheduler the process states are **New, Ready, Running, Waiting, and Halted. The short term scheduler manages these states using a Queue data structure.**

The waiting state means that a **process has been halted because of say, I/O.**

If an OS is running a process at some point it may revert back to running code in the OS. This might be caused by **I/O request by the process, an interrupt occurs possibly due to a fault, or some outside event happens such as an interrupt or an I/O process that needs immediate attention.**

A high priority job may get suspended if it is w**aiting on I/O.**

MEMORY MANAGEMENT

Memory management is t**he dynamic decision by the operating system as to what process gets which memory resources.**

Swapping is the process whereby the op system moves (swaps) processes between disk and main memory so that those which are ready to run can be executed and those that are waiting on I/O are on disk.

Partitioning is the subdivision of memory into partitions suitable for process execution. Ideally each process would get a memory partition that is the size that it needs and no bigger.

Partitioning can produce holes in memory. This can happen when a **small process may complete and be taken out leaving a place where another process could go. But if the partition is too little then the area remains open.**

Compaction **is an attempt to solve the partition holes problem by occasionally restructuring the partitions so as to remove the holes and consolidate the free memory into one partition.**

A logical address is t**he address of a memory location relative to the beginning or base address of a process.**

A physical address is t**he actual physical address in memory where variables an instructions are stored.**

A page in memory **tend to be small and of fixed size. Partitions are made up of groups of pages. Pages can never overlap, i.e. page addresses are contiguous.**

A Page frame is **a group of addresses in memory that can hold a page of data.**

Pages of a process need not be contiguous.

The CPU keep track of which pages belong to which process by using **a page table in memory.**

Figure 8.16 p. 268 explains how a logical address in a paged system is converted to a physical address. **Logical address has two parts. The upper part gives the page address in the page table and the lower part gives the relative address of the data within the page. The upper part of the logical address is used to access the page table and thereby find the frame number in main memory. The physical address then consists of the frame number plus the relative address (added to the base) of the data within the page.**

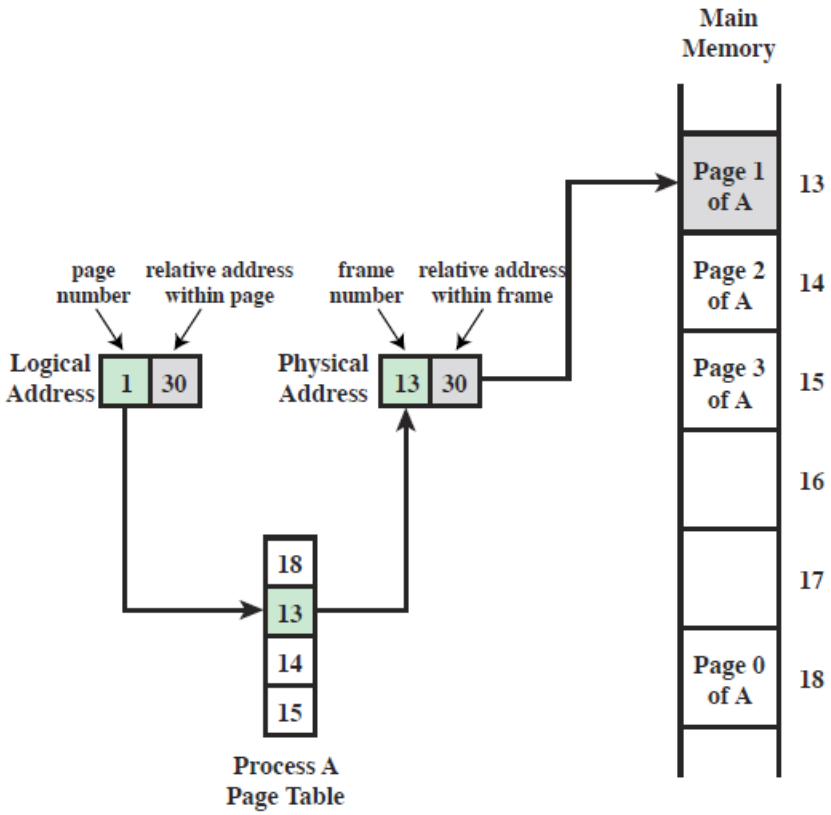

Figure 8.16 Logical and Physical Addresses

Paging does not solve the hole problem but with paging **the largest hole is less than the size of a page since pages need not be contiguous.**

VIRTUAL MEMORY
In demand paging **Each page of a process is brought in as it is needed.**

A page fault occurs when a **program needs a page which is not in main memory.**

Page thrashing occurs when t**he op system throws away a page that is needed in the near future. After throwing away this page and reloading it, it in turn throws away another page that is needed in the near future.**

Demand paging makes it possible for a system to run a program that is larger than all of main memory?**. Only the pages that are immediately executing need be in memory at any one time.**

The difference between real memory and virtual memory is that **Real memory is the amount of main memory. Virtual memory is the amount of memory a program appears to have. Much of it may be on disk.**

Since the page table can be kept in memory, for a given process, we can calculate the number of table entries by dividing the logical address size by the page size. **For example, if a given architecture can support a 31 bit logical address then the largest process can be 2^{31} bytes. If a page is 512 bytes = 2^9 then $2^{31}/2^9 = 2^{22}$ page table entries.**

A page table can therefore be much too large to keep in main memory. So a **portion of the page-table is kept in memory. Effectively the page-table is itself cached and the pages-table is itself paged.**

When a process is running it is not necessary that the page table be in real memory. But **At least part of the page table must be in real memory – that part that refers to the currently accessed addresses.**

There is a two-level approach to page table management. It works like this: **A page directory is established to keep track of where the page table is. If there are X entries in the page directory and Y entries in the page table then a process can access X * Y page frames. The page directory entries are pointers to page tables.**

In principle, every virtual memory address can cause a user to access two physical memory address. **One for the page table entry and one for the data.**

A translation look aside buffer is a c**ache for page table entries.**

The TLB is stored **In the memory management hardware typically as part of the CPU.**

Figure 8.18 p. 271 explains how a TLB works.

The worst case access time is a **TLB fault + page fault + memory full.**

Note that the flow chart has two entries for updating the page table. **One has to delete something from memory before loading a new page and the other doesn't.**

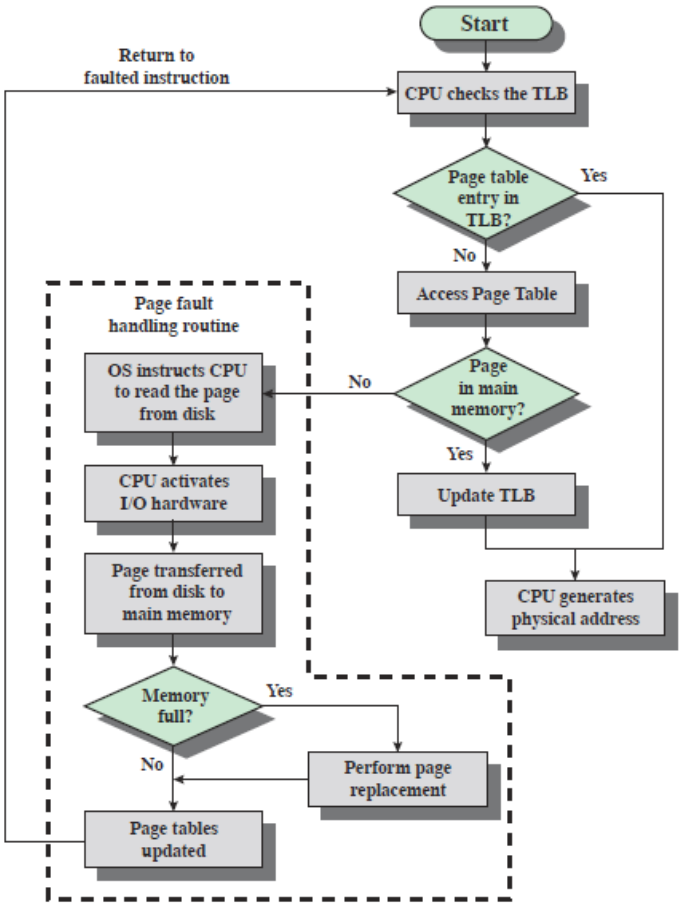

Figure 8.18 Operation of Paging and Translation Lookaside Buffer (TLB) [FURH87]

When the CPU does a memory access it goes to the TLB before checking the cache for a particular data item in memory because the **cache needs a real memory address and CPU is generating a virtual address. TLB finds page etc to allow translation of virtual into physical address.**

Memory is sometimes divided into segments. **Segments are of variable size and need not be contiguous. In fact segments can overlap. Segments are for logical organization of a program and pages are for physical organization of a program.**

PENTIUM II MEMORY MANAGEMENT HARDWARE
The Pentium II use segmentation: **There are 4 choices: unsegmented and unpaged, unsegmented paged, segmented unpaged, and segmented paged.**

The PII has **2^{32} = 4 Gbytes of real address space. Address bus is 32 bits wide.**

The PII has a **16 bit segment register which gets combined with a 32 bit offset to form a 48 bit number. Two of the 48 bits are used to establish one of four priority levels. The remaining 46 bits form the virtual address. 2^{46} = 64 terabytes.**

According to IBM about 88 terabytes.

The Pentium II has a 32 bit real address and a 46 bit logical address. If the page size is 4 K we can compute the number of entries in the page table:**$2^{46}/2^{12} = 2^{34}$ = 16 Gpages. Typically there are 4 bytes per page table entry so $2^{34} * 4 = 2^{36}$ bytes = 64 Gbytes just to hold the page table. Note that the 46 bit virtual address gives 64 tera bytes of virtual address space per process. On a more practical note, most systems don't have 64 terabytes of hard disk space to provide that much virtual memory anyway so the practical size of the page table is somewhat smaller.**

In some sense the Pentium II can have more than 64 terabytes of virtual address space. **In theory each process is limited to just 64 terabytes so the ultimate limit is defined by how many processes you want to run which, in turn, is defined by how slow you are willing to let the system get.**

Memory protection implemented on the Pentium II usng t**he least 2 bits of the segment register give the priority level. There are 4 levels with 0 = kernel level, 1 = most of the operating system level, and levels 2 and 3 are for applications.**

Use Figure 8.21 p. 278 explains how paging and segmentation work on the Pentium II?

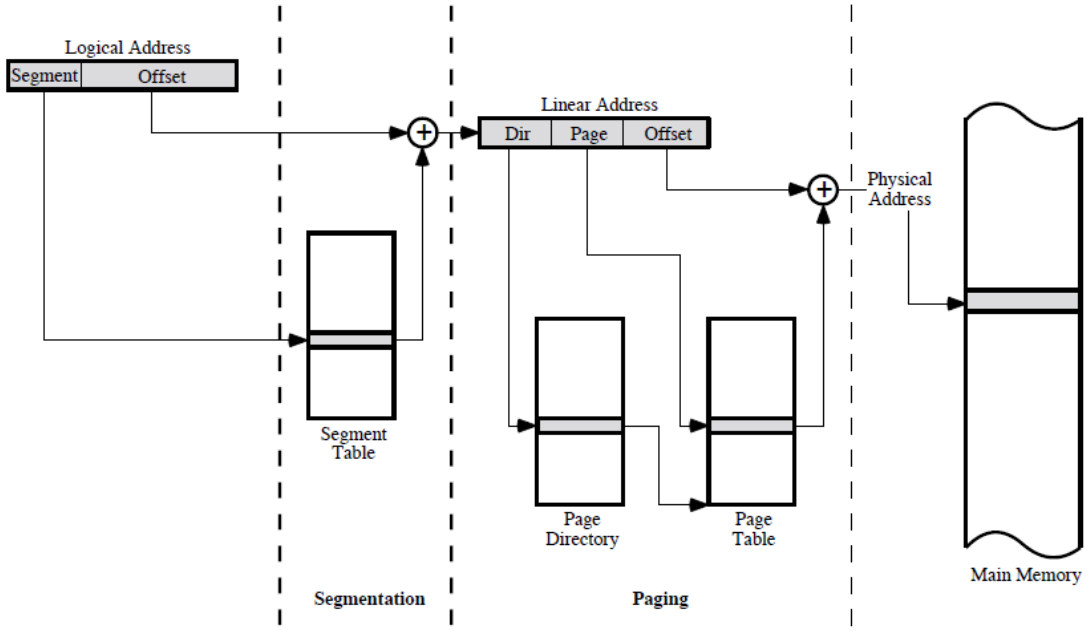

Figure 8.21 Pentium II Memory Address Translation Mechanisms

CS 320 **February 14, 2018**
In Class
Chapter 8

1. Given the page table below in which all numbers are decimal and everything is numbered starting from 0 and all addresses are byte addresses. The page size is 1024. To get a physical address from a logical address you must

a. Split the binary address into a virtual page number and an offset.
b. Use the VPN as an index into the page table.
c. Extract the PFN
d. Concatenate the offset as in PFNx1024 + offset to form a physical address.

Virtual Page Number	Valid bit	Reference bit	Modify bit	Page Frame Number
0	1	1	0	4
1	1	1	1	7
2	0	0	0	-
3	1	0	0	2
4	0	0	0	-
5	1	0	1	0

A) If the virtual address is 1052, what is the physical address that this corresponds to?
1052 = 1024 + 58 so on page 1. From table page frame number is 7 so
7 x 1024 + 28 = 7196

B) The virtual address 2221 produces a page fault. Why?
2221 = 2x1024 + 173 which is page 2 which is not valid.

C) If the virtual address is 5499, what is the physical address that this corresponds to?
5499 = 5 x 1024 + 379 on page 5. From table page frame number is 0 so
0 x 1024 + 379 = 379.