Computer Organization

COMP2120A

Qi Zhao

April 17, 2024

Operating System Support

Operating System (OS)

- The OS is a **software** that controls the execution of programs on a processor and manages the processor's resources.
- **Convenience**: An OS makes a computer more convenient to use.
- **Efficiency**: An OS allows the computer system resources to be used in an efficient manner.

OS as a User/Resource Interface

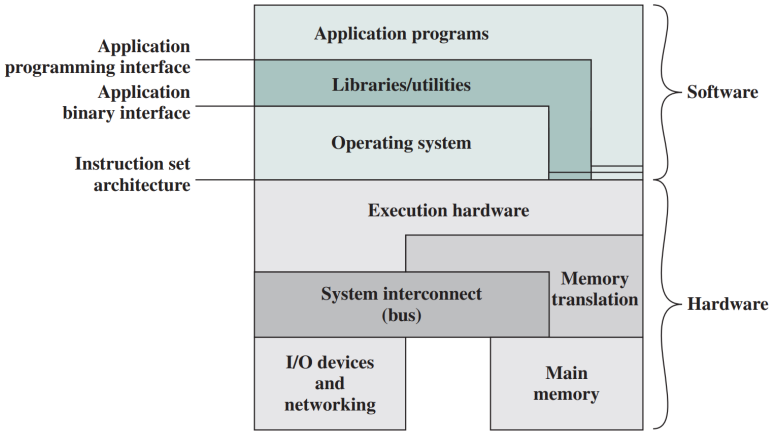


Figure 8.1 Computer Hardware and Software Structure

Services provided by OS

- **Program creation** — in the form of utility programs that are not actually part of the OS, but are accessible through the OS, e.g. Editor (vi, emacs), compiler (gcc), debugger (gdb), etc.
- **Program execution** — to execute a program, system need to read the program from the hard disk to the main memory, prepare resources, such as process table, etc. The user just type the program name, and the OS will do the rest.
- **I/O access** — the OS provides a uniform I/O interface to the programs, such as `open()`, `read()`, `write()`, `close()`, while the detail is left to the OS, e.g. via the device driver. Also handle sharing of the devices among users to prevent users from holding the device indefinitely

Services provided by OS

- **File system management** — how files and directory are created, modified, etc is hidden from the user. Also file protections, file sharing etc. is managed by the OS.
- **System access** — control access to the system and to specific system resources, protect these resources/data from unauthorized users,
  e.g. Memory protection: segmentation fault when you try to access memory that you are not authorized to.
- **Error detection and response** — report error to programs, and their remedial action.
- **Accounting** — collect usage statistics for various resources and monitor performance parameters such as response time, page fault rate etc.

Protection Scheme

- The CPU will execute in different mode to facilitate protection. Most systems uses 2 modes – user mode and supervisor (kernel) mode, although some system may have as many as four different modes to fine-tune protection/priviledges.
- Some resources (such as I/O devices) or actions (some special instructions) can only be run in kernel mode, so that users will not have access to them.
- OS is supposed to be well-tested, while bugs exist in user programs. If user programs are allowed to handle system resources, they may hang the system, or hold the resources indefinitely, if their program is buggy.
- Only the OS (which execute in kernel mode), will have the rights.

OS Call Mechanism

- System has to change from user mode to kernel mode to run/use the protected resources.
- Users cannot change themselves to kernel mode.
- The OS functions are usually accessed via special entry points called system calls.
- The OS will change to kernel mode, and execute with higher privileges than user program.

Multitasking and Time-sharing System

- CPU idle when I/O is performed.
- To fully utilize CPU resource, CPU is shared among processes (multiple tasks). Each process is allocated a time slice of 0.1 sec, say. When process uses up its time slice, it will stop execution, and let the next process to be executed.
- When a process cannot proceed (e.g. perform I/O), it will also stop execution

Process Scheduling

- The system maintain a queue of processes.
- The order of the processes in the queue is controlled by many factors — process priority, how long the process has been waiting, whether the process has used up its previous time slices (i.e. CPU bound job), the system loading etc.
- The first process in the process queue is picked for execution

Problems

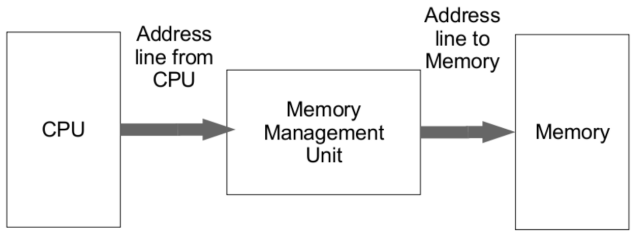
- Main memory (Physical memory) is of limited size, especially in a machine with a multitasking OS.
- What if you need more? ISA can have an address space greater than the physical memory size.
- Should the programmer be concerned about the size of code/ data blocks fitting physical memory?
- Should the programmer manage data movement from disk to physical memory?

Virtual Memory

- **Physical Address** — addresses that are used to actually access the memory, which can be smaller, e.g. 1GB only.
- **Logical address** — addressing space of your program, e.g. if 32-bit address is used, then addressing space is from 0 to 4GB.
  It is logical, in contrast to physical addressing space. It is the space seen by the program, but not in physical memory.
- Programmer sees virtual memory. Programmer does not need to know the physical size of memory nor manage it
- A small physical memory can appear as a huge one to the programmer
- Life is easier for the programmer

Memory Management Unit (MMU)

- Similar to Cache memory, it is based on the principle of locality, and we use address mapping to map logical addresses (of programs) to physical addresses (of memory).
- Mapping done by Memory Management Unit (MMU).



Paging

- Both addressing space are divided into pages (blocks) of fixed size, say, 4KByte pages.
- Each process will have its own logical addressing space. Hence all program can start at address 0, if desired. They will be mapped to different places in the physical addressing space.
- Each process will need to keep track of a data structure (page table), which will provide information on how to perform memory mapping.

Page Table Entries

- Each page in logical address space will have a corresponding entry in the Page Table

Page 0				
Page 1				
Page 2				
. . .				
	V	P	D	Physical Frame Number

Page Table Entries

Each PTE (Page Table Entry) contains information as follows.

- **V**: If the page is in memory or not (Valid bit, V).
  If yes, where it is — physical frame number.
- **P**: Protection information —, usually more than 1 bit, indicating the protection mode, such as r/w and supervisor/user accesses.
- **D**: If the page has been changed (dirty bit, D) — to prevent unnecessarily write back when the page has not been changed. (write back to hard disk is expensive).

How to store this PTE?

Address Mapping Example

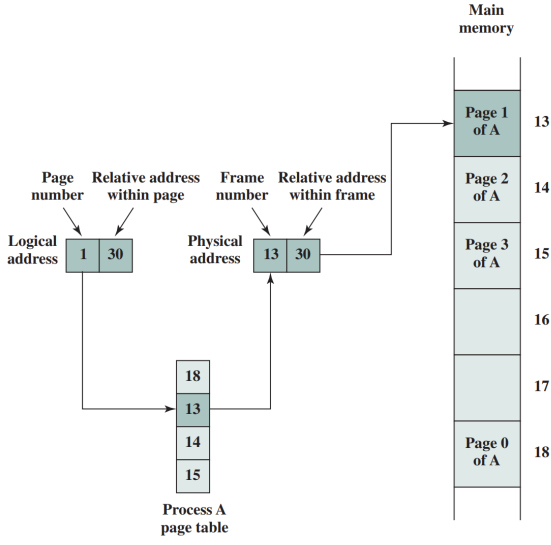


Figure 8.16 Logical and Physical Addresses

Demand Paging

- When program starts, nothing is in memory. We only bring in pages on demand (when the page is needed).
- **Advantage** — fast response, because we do not need to bring in the entire program for it to start.
- **Disadvantage** — a lot of page faults, until a stable working set of the program has been brought into the memory. Use prepaging to bring in a number of pages at the beginning to minimize this.

Cache/Virtual Memory Analogues

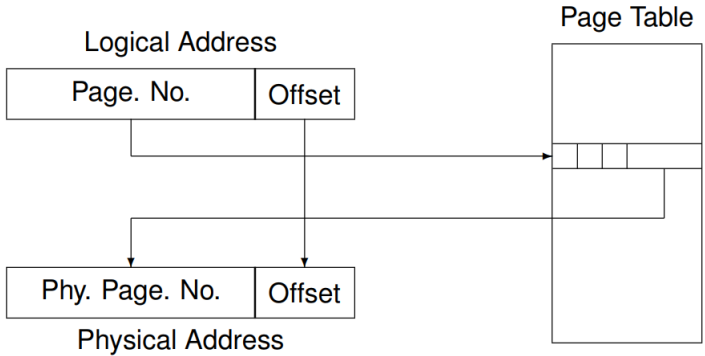
Cache	Virtual memory
Block	Page
Block Size	Page Size
Block Offset	Page Offset
Miss	Page Fault
Tag	Virtual Page Number

Address Translation

When the program access a memory location via the CPU address bus (can either be code or data),

- extract the logical page number (the most significant n-bits)
- find the corresponding PTE and see if the page is in physical memory.
- If YES, then get the physical frame number and append the offset within the page to it.
- If NO, then generate a page fault

Address Translation



Address Mapping

- If the page is not in main memory, then a page fault is generated. The Operating System will take over, fetch the required page from hard disk, put it in the memory (possibly with a replacement) and restart the program.
- Since hard disk I/O is very slow, the process involved will usually be suspended and the CPU will switch to another process.

Page Fault Handling

- Find the page from hard disk.
- Find a free page in the physical memory. If there is none, find one to replace.
- if the page (to be replaced one) is dirty, write back to hard disk.
- Invalid the page table entry of that page.
- Write the page to that physical page
- Modify the PTE of the new page to contain the correct info.

Page Replacement

- First-in-first-out (FIFO)
- Least Recently Used (LRU)

Write Strategy

- Write back strategy is always used.
- Write through to the HDD is too expensive.

Translation Lookaside Buffer(TLB)

- Every virtual memory reference can cause two physical memory accesses:
  one to fetch the appropriate page table entry,
  one to fetch the desired data
- The Page Table is cached in the Translation Lookaside Buffer (TLB)
- Perform address translation via Translation TLB, the cache of Page Table. If PTE not in TLB, then use the Page Table, and put the corresponding entry into TLB. (Only valid pages will be in TLB.)

Address Mapping

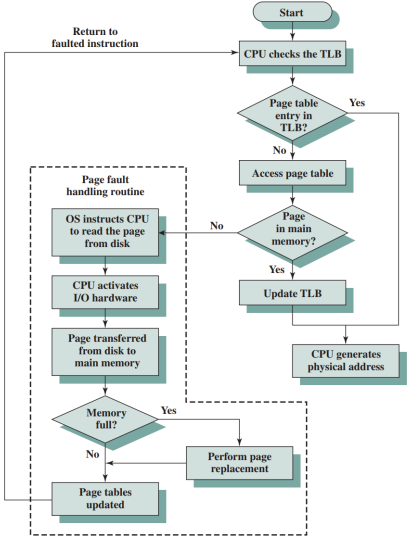


Figure 8.18 Operation of Paging and Translation Lookaside Buffer (TLB)

Combining Cache and Virtual Memory

- (Not TBL) after obtaining the physical address, we use this physical address to access cache memory, and cache miss handling

Combining Cache and Virtual Memory

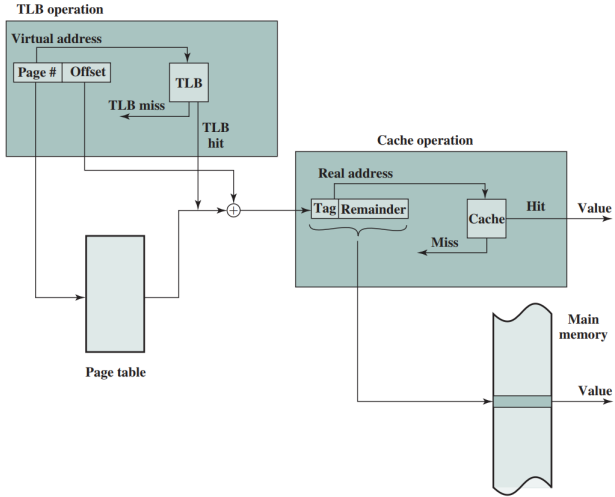


Figure 8.19 Translation Lookaside Buffer and Cache Operation