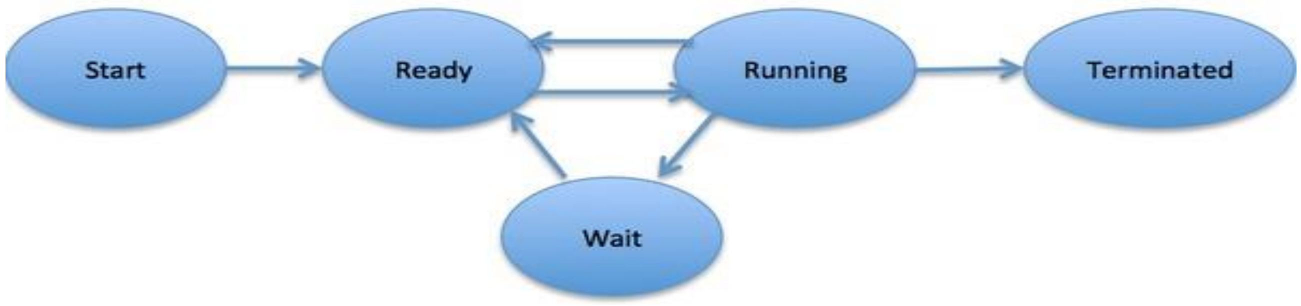
Chapter6: Processor Scheduling

- Scheduling Basics:
    - Process Control Block(PCB): **PCB holds all the information regarding a process**.
    - Process Control Block;
        - Current state
        - PC address
        - General Process Register
        - Pointer to the next PCB
        - Priority
        - Address address_space;
    - CPU Burst: Continuous CPU activity by a process before requiring an I/O operation (**Reading**)
    - I/O Burst: Activity initiated by the CPU on an I/O device (**Writing**)
    - Step involved in Scheduler:
        - Grab the attention of the processor
        - Save the state of the currently running process
        - Select a new process to run
        - Dispatch the newly selected process to run on the processor (Dispatch: loading the processor registers with the saved state of the selected process)
    - Ready queue: A linked list of the PCBs.
    - I/O queue of PCBs: Upon a blocking I/O request made by a process, the PCB of the associated process moves to the I/O queue and awaits I/O completion
    - Non-preemptive Algorithm: A process either executes to completion or give up the processor on its own accord – **that is, voluntarily, to perform I/O**.
    - Preemptive Algorithm: The scheduler yanks with the processor away from the current process to give it to another process
    - Thrashing: the dynamic memory usage of the processes currently in the ready queue exceeds the total memory capacity of the system. (过载)
- Performance Metrics
    - *Throughput*: Number of jobs executed per unit time
    - *Average turnaround time*: the average elapse time for a give job **entering and leaving** the system. (**turnaround time is completion time**)
    - *Average waiting time*:

- **Response time(turnaround time) = completion time – arrival time** 有 arriving time 的时候再减。
- **Wait time = response time – execution timeg**
- *Response time*: the elapse time for a give job (**completion time – arrival time**)
- **Starvation**: for some reasons, a job doesn't make any forward progress, in an unbounded response time
- **Convoy Effect**: the scheduling follows a fixed pattern(just like a military convey)

- Non-preemptive Scheduling Algorithm
  - First-Come First-Served(FCFS)
    - Intrinsic Property is the **arrival time**.
    - Uniquely: **Convey Effect, but NO starvation**
    - Based on the arriving time, the processes go the CPU. CPU 会处理最先到达的程序。当一个程序在 CPU 完成操作，谁先到达 I/O，谁就先使用资源完成任务。当一个程序在 I/O 中完成任务，它必须先回到 CPU。正在执行的程序会排在 CPU ready queue 的后面。直到一个程序完全执行完，它会从 ready queue 总移除，否则 ready queue 总是按照 arriving time 的顺序执行。
    - ***Ready Queue always follows the arriving time***
  - Shortest Job First(SJF)
    - Need to know the **CPU burst time**
    - SJF results a **better response time** for **short job**
    - SJF avoids the **convey effect**
    - ***Ready Queue always follows the shortest CPU burst time***
  - Priority
    - Priority: **an extrinsic property**
    - Priority: a small integer value, with each process to denote its relative importance, compared with other processes.
    - The scheduling within each level is FCFS.
- Preemptive Scheduling Algorithm
  - The scheduler can do two things:
    - Control of the processor anytime
    - Save the state of the currently running process for proper resumption
  - Shortest Remaining Time First(SRTF)
    - A special case of the SJF scheduler, with preemption added in

        - The scheduler has an estimate of the running time of each process. When a process rejoins the **ready queue**, the scheduler computers the remaining processing time of the job.
    - Round Robin Scheduler
        - Time-shared environment
        - Assume that there are ***n*** ready process. The scheduler assigns the processor in time quantum units, q, usually referred to as time-slice, to each process.
        - **Context Switch time**: The time spending while switching among these ready processes.
        - FCFS scheduler is a special case for Round Robin Scheduler while the time-slice is infinity.
- The state of a process:
    - **Start**: the initial state when a process is first started/created
    - **Ready**: <u>the process is **waiting to be assigned to a processor**.</u> Ready processes are waiting to have the processor allocated to them by the operating system so that they can run. Process may come into this state after **Start** state or while running it by but interrupted by the scheduler to assign CPU to some other process. (**Happens while waiting the processors**)
    - **Running**: once the process has been assigned to a processor by the OS scheduler, the process state is set to running and the processor execute its instructions.
    - **Waiting**: Process moves into the waiting state if it needs to **wait for a resource**, such as waiting for user input, or waiting for a file to become available**. (Happens while requesting some resources)**
    - **Terminated or exit**: once the process finished its execution, or it is terminated by the operating system, it is moved to the terminated state where it waits to be removed from main memory.



- Dispatcher: in round robin
    - Dispatches the process at the head of the ready queue on the processor, setting the timer to the time quantum ***q***. Then the timer will interrupt CPU once the time is right.
    - Round robin scheduling algorithm:

        - Get PCB at the head of the ready queue
        - Set timer
        - dispatch

Chapter7: Memory Management Techniques

- Broker
    - A piece of hardware between the CPU and the memory
    - It maps the CPU **generated memory address** into the **real memory address**
    - **Goals** for a good memory manager:
        - Require the minimal hardware support
        - Keep the impact on memory access low
        - Keep the memory-management overhead low
- Thrashing
    - If the system has to swap pages at such a higher rate that major chunk of CPU time is spent in swapping, then this state is known as thrashing
- Simple schemes for memory management
    - Separation of user and kernel:
        - A **mode** bit to mark kernel mode or user mode
        - A **privileged** instruction to flipping this bit
        - A **fence register** to separate the two mode
    - Static relocation:
        - The memory bounds, for a process, being set at the time of linking the program and creating an executable file.
    - Dynamic relocation:
        - Be able to place an executable into any region of the memory that can accommodate the memory need of the process
- Memory Allocation Schemes
    - Internal fragmentation: refers to the wasted space internal to fixed-size partitions that leads to poor memory utilization. For example, if a fixed memory chunk is 8K, while requesting 2K memory, there will be 6K internal fragmentation.
    - External Fragmentation: Sum of all total internal fragmentation of each partition, for variable-size partitions. For example, if there is a 13K memory chunk, while requesting 3K from middle, then there will be 10K external fragmentation by 8K + 2K.

    - Compaction: when the level of external fragmentation goes beyond tolerable limits. For example, for the previous example, we will move the 8K and 2K fragmentation together to get a contiguous memory size by 10K.
- Paged Virtual Memory
    - **As the size of memory growing, external fragmentation becomes a very acute problem.**
    - Assumption of **contiguous memory** present **in the user's view**. -> **Virtual Memory**
    - Paging is built to implement the virtual memory idea
    - The broker breaks up this contiguous view (**virtual memory**) into **equal logical entities** called **page**
    - Similarly, the **physical memory** consists of **page frames**, or called physical frames
    - Logical page and physical frame have the **same size**
    - Process control block:
        - State
        - PC address
        - Pointer to next PCB
        - Priority
        - PTBR address: OS use that to determine the memory footprint of a process
    - Page table:
        - Broker uses that to map between **logical page** to **physical frame**, translate the **virtual address** into a **physical address** by looking up in page table.
        - But still the page **might** cause the **internal fragmentation**
        - The virtual address generated by CPU consisting of two things: virtual page number(VPN) and offset within the page. **Page size related to offset**
        - If the page size is N, than log2N low-order bits of the virtual address form the page offset
        - Looking up the page table to find the ***physical frame number(PFN)*** corresponding to the ***virtual page number(VPN)*** in virtual address
        - The page table **resides in memory**, <u>one per **process**</u>
        - Page table base register(PTBR): CPU needs to know the location of the page table in memory. So that PTBR contains the **base address** of **page table** for **current running process**.
        - Page table set up:
            - The memory manager records the PTBR value for this process **in the associated PCB**
- Segmented Virtual Memory

  - The user's vie of memory is not a single linear address space, but it is composed of several distinct address space. Each of address space is called a segment. CPU generates address that have two parts(segment number, segment offset)

Chapter8: Details of Page-based Memory Management

- Demand paging
  - Personal Note: In demand paging, the data is not copied from the disk to the RAM until they are needed or being demanded by some program. The data will not be copied when the data is already available on the memory.

  - It's prudent for the memory manager **not** to **load** the **entire program** into **memory**, upon startup.
  - The basic idea is to load parts of the program that are **not in memory** on demand.
  - With demand paging, the page may **not be in memory yet**, so that we need **additional information** in **page table**. **A valid** bit for **page table entry**
  - Page fault:
    - Definition: **when a running program accesses a memory page that is mapped into the virtual address space, but not actually loaded into main memory**
    - The hardware fetches the **PFN** from the **page table** as a part of **address translation**
    - However, with demand paging, the page
  - Page fault handler: (***Bring the missing page from disk into memory***)
    - Find a free page frame
    - Load the faulting virtual page from the disk into the free page frame
    - Update the page table for the faulting process
    - Place PCB of the process back in to ready queue of the scheduler
  - Frame table:
    - Contain the reverse mapping
    - Given a frame number, it gives the Process ID and the virtual page number that currently occupies this page frame.
  - Details of a page fault
    - Find a free page frame:

        - Looks up the free-list of page frames
        - **If the list is empty**
        - Memory manager has to bring **the faulting page** from the **disk** into **physical memory**
        - Implying that we need to **make room in physical frame** to bring in **the faulting page**
        - The manager **selects** some ***physical memory*** as a **victim** to make room for the **faulting page**.
    - Pick the victim page:
        - Clean page and dirty page:
            - Clean page: the one that has **not** been modified by the program since it was brought from the disk
            - Dirty page: the one that has been modified by the program since it was brought from the disk
        - If this is a clean page, the manager needs to set the PTE corresponding to the page as

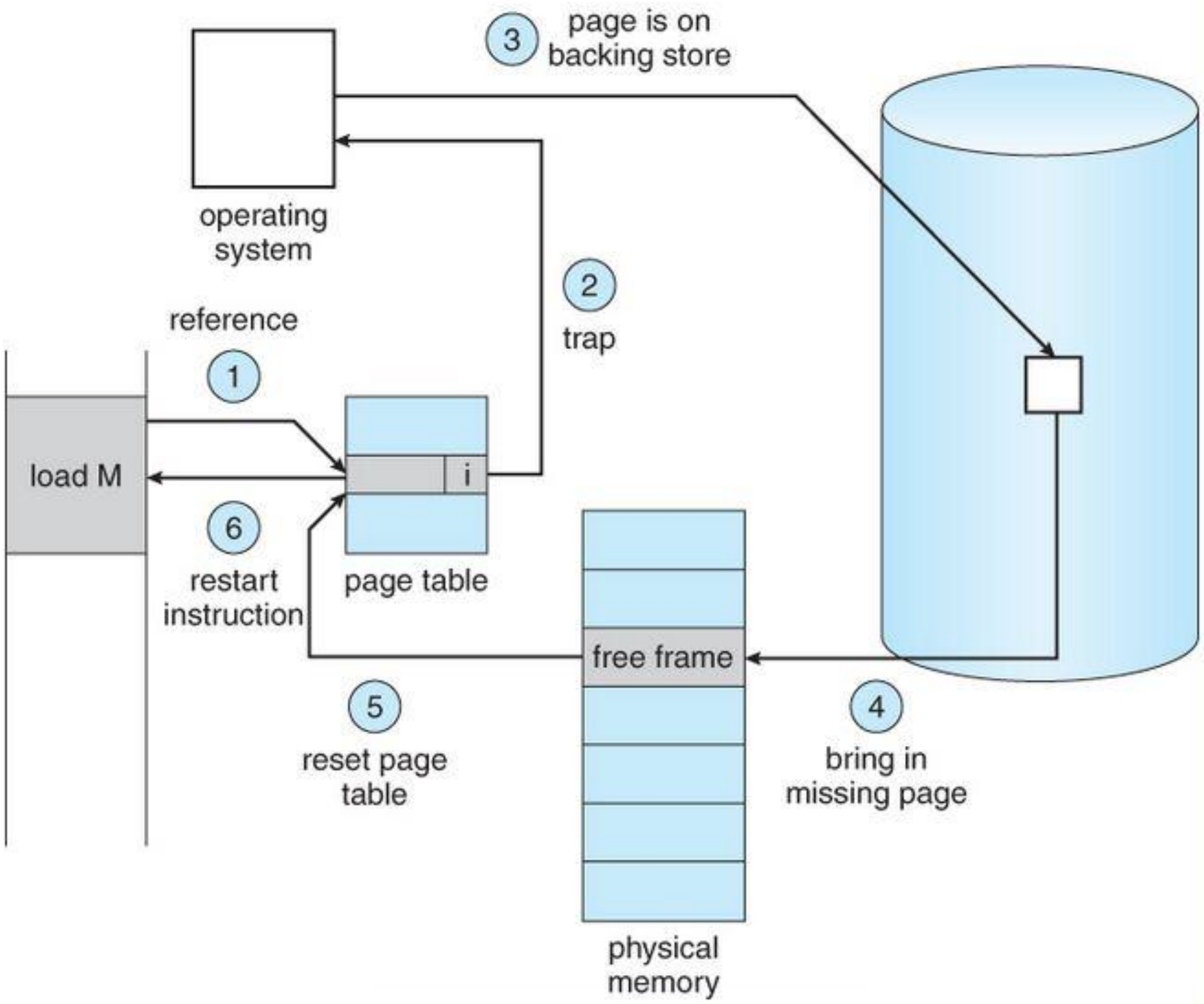


Figure 9.6 - Steps in handling a page fault

invalid because the contents of this page need not to be saved

        - Load the faulting page
        - Update the page table for the faulting process and the frame table
        - Restart faulting process

- Page replacement algorithm:
    - Belady's Min:
        - If we know the entire string of references, it's best if we replace the one that is not referenced for the longest time in the future.
        - In 1966, Laszlo Belady proposed the ***optimal replacement algorithm*** called Belady's Min, to serve as the gold standard for evaluating the performance of any page replacement algorithm
    - Random replacement:
        - Memory manager may default to random replacement in the absence of bookkeeping information sufficient to make a decision.
- Thrashing: Spent more time on paging than computing.
- Working Set:
    - Definition: the set of pages that defines the **locus**(轨迹) of activity of a program.
    - WSS: Working Set Size: denotes the number of distinct pages touched by a **process** in a window of time. For example, the WSS for this process is 2 in the interval t1 – t2.
    - Memory pressure: the summation of the WSS of all the processes currently competing for resources. (Just add all WSS together)
- Translation Lookaside Buffer(TLB)
    - A memory cache that is used to reduced the time taken to access page table which is in the main memory. Storing the recent access transition records which are highly possible to be access again

Chapter9: Memory Hierarchy

- Concept of Cache:
    - Memory can be small but fast or large but slow
    - DRAM: Dynamic Random Access Memory (**Large but slow**)
    - SRAM: Static Random Access Memory (**Small but fast**)
    - Main memory: the physical memory that is **visible** to the instruction set of the computer
    - Cache: a hidden storage. **TLBs** are a **special case of caches** used for holding address translation
- Locality:
    - Temporal Locality: high probability of a program accessing **recent memory** location (时间上)
    - Spatial Locality: high probability of a program accessing **adjacent memory** location (空间上)
- Basic Terminologies:
    - Hit: CPU find the content of the memory address in the cache. Hit rate is the probability of such a successful lookup of the cache by the CPU
    - Miss: CPU didn’t find the content of the memory address in the cache.
    - Miss penalty: time penalty associated with servicing a miss at any particular level of the memory hierarchy.
    - Effective memory access time(EMAT): EMAT = Tc + m * Tm. Where **Tc** is cache access time, **m** is missing rate, **Tm** is miss penalty
- Multilevel memory hierarchy:
    - L1(first-level), L2(second-level), L3(third-level)
    - T(i+n) > T(i+n-1) > … > T(2) > T(1)
    - EMAT(i) = T(i) + m(i) * EMAT(i+1)
    - EMAT = EMAT(L1)
- Cache Organization: *Placement, Algorithm for lookup, Validity*
- Direct-Mapped Cache Organization:
    - Definition: For each memory address entry, there is a specific cache location for it.
    - Terminology:
        - Cache entry: consists of cache
            - [Valid bit | Tag | Data]
            - This structure is also referred to as **cache line**, **cache entry**, or **cache block**
            - Tag: we need information in each cache entry to distinguish between multiple memory addresses that may map to the same cache entry

        - Memory interpretation:
            - [cache tag | cache index | offset   ]
            - Cache index: in direct mapped cache, index helps to locate each cache entry
            - Offset: helps to find the specific byte
- Fully associative cache:
    - Definition: the cache search through **all the entries** to see whether there is a match between **cache tag** in memory and **tag** in valid entry
    - Do not need index because all entries are in the same line
    - Memory interpretation:
        - [cache tag | block offset]
- Set associative cache:
    - **It's fine to understand that directed-map is 1-way set associated cache**
    - Definition: associated with a set of **cache blocks**
    - Terminology:
        - Associativity: the number of homes that a given memory block has in the cache.
        - S: cache size
        - B: block size
        - p: degree of associativity (it's two for a two-way set associated cache, it's four for a four-way set associated cache)
        - N: total number of data blocks
        - t: size of cache tag
        - n: size of cache index
        - b: size of block offset
        - L: number of cache lines
    - Memory interpretation:
        - [cache tag | cache index | block offset]
        -
    - Formula:
        - N = S/pB
- Effect of memory stalling due to cache misses on pipeline performance
    - Execution time = number of instructions executed * CPI avg * clock cycle time (in chapter 5)
    - Execution time = (number of instructions executed * (CPI avg + memory-stall avg)) * clock cycle time
    - Effective CPI = CPI avg + Memory-stall avg
    - Total memory stalls = number of instructions * memory – stalls avg
    - Memory-stall avg = misses per instruction avg * miss-penalty avg

- Miss:
    - Compulsory miss: if the cache is initially empty and we've never seen this tag & index combo before
    - Conflict miss: two items are competing for the position in a single cache line because they have the same index. This situation is very common for direct mapped caches, but never occur for fully associative caches
    - Capacity miss: **only** for fully associative cache

Internal register: stack register