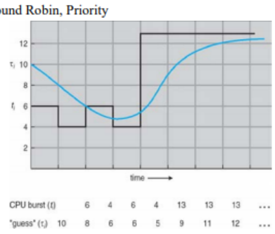
Ch.6 – CPU Scheduling

- Process execution consists of a cycle of CPU execution and I/O wait
- CPU scheduling decisions take place when a process:
  - Switches from running to waiting (nonpreemptive)
  - Switches from running to ready (preemptive)
  - Switches from waiting to ready (preemptive)
  - Terminates (nonpreemptive)
- The dispatcher module gives control of the CPU to the process selected by the short-term scheduler
  - Dispatch latency- the time it takes for the dispatcher to stop one process and start another
- Scheduling algorithms are chosen based on optimization criteria (ex: throughput, turnaround time, etc.)
  - FCFS, SJF, Shortest-Remaining-Time-First (preemptive SJF), Round Robin, Priority
- Determining length of next CPU burst: Exponential Averaging:
  1. t_n = actual length of n^{th} CPU burst
  2. τ_{n+1} = predicted value for the next CPU burst
  3. α, $0 \le \alpha \le 1$ (commonly α set to 1/2)
  4. Define: $\tau_{n+1} = \alpha * t_n + (1-\alpha)\tau_n$



- Priority Scheduling can result in starvation, which can be solved by aging a process (as time progresses, increase the priority)
- In Round Robin, small time quantums can result in large amounts of context switches
  - Time quantum should be chosen so that 80% of processes have shorter burst times that the time quantum
- Multilevel Queues and Multilevel Feedback Queues have multiple process queues that have different priority levels
  - In the Feedback queue, priority is not fixed → Processes can be promoted and demoted to different queues
  - Feedback queues can have different scheduling algorithms at different levels
- Multiprocessor Scheduling is done in several different ways:
  - Asymmetric multiprocessing: only one processor accesses system data structures → no need to data share
  - Symmetric multiprocessing: each processor is self-scheduling (currently the most common method)
  - Processor affinity: a process running on one processor is more likely to continue to run on the same processor (so that the processor's memory still contains data specific to that specific process)
- Little's Formula can help determine average wait time per process in any scheduling algorithm:
  - $n = \lambda \times W$
  - n = avg queue length; W = avg waiting time in queue; λ = average arrival rate into queue
- Simulations are programmed models of a computer system with variable clocks
  - Used to gather statistics indicating algorithm performance
  - Running simulations is more accurate than queuing models (like Little's Law)
  - Although more accurate, high cost and high risk

Ch.7 – Deadlocks

- Deadlock Characteristics: deadlock can occur if these conditions hold simultaneously
  - Mutual Exclusion: only one process at a time can use a resource
  - Hold and Wait: process holding one resource is waiting to acquire resource held by another process
  - No Preemption: a resource can be released only be the process holding it after the process completed its task
  - Circular Wait: set of waiting processes such that P_{n-1} is waiting for resource from P_n, and P_n is waiting for P_0
    - "Dining Philosophers" in deadlock

Process control information is used by the OS to manage the process itself. This includes:

- **The process scheduling state:** The state of the process in terms of "ready", "suspended", etc., and other scheduling information as well, like priority value, the amount of time elapsed since the process gained control of the CPU or since it was suspended. Also, in case of a suspended process, event identification data must be recorded for the event the process is waiting for.
- **Process structuring information:** process's children id's, or the id's of other processes related to the current one in some functional way, which may be represented as a queue, a ring or other data structures.
- **Interprocess communication information:** various flags, signals and messages associated with the communication among independent processes may be stored in the PCB.
- **Process Privileges** in terms of allowed/disallowed access to system resources.
- **Process State:** State may enter into new, ready, running, waiting, dead depending on CPU scheduling.
- **Process Number (PID):** A unique identification number for each process in the operating system (also known as Process ID).
- **Program Counter (PC):** A pointer to the address of the next instruction to be executed for this process.
- **CPU Registers:** Indicates various register set of CPU where process need to be stored for execution for running state.
- **CPU Scheduling Information:** indicates the information of a process with which it uses the CPU time through scheduling.
- **Memory Management Information:** includes the information of page table, memory limits, Segment table depending on memory used by the operating system.
- **Accounting Information:** Includes the amount of CPU used for process execution, time limits, execution ID etc.
- **I/O Status Information:** Includes a list of I/O devices allocated to the process.

File control block

El FCB (*"File control block"*, en español *"Bloque de control de archivo"*) es el método utilizado en el sistema operativo MS-DOS de Microsoft para mantener información en memoria de un archivo abierto antes de que existiesen los directorios.

Descripción técnica [editar]

En MS-DOS los ejecutables COM se cargan en memoria situando en la posición 80H de la memoria del proceso, un espacio en donde reside una estructura de datos para mantener ahí la información referente a que archivos tiene abiertos ese proceso, ese espacio de memoria recibe el nombre de DTA y los datos allí almacenados reciben el nombre de FCB.

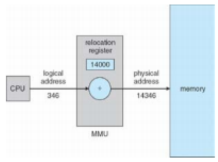
FCB permite al igual que MS-DOS, longitudes de nombres de archivo de 8 caracteres como máximo y 3 caracteres adicionales para la extensión que identifica el tipo de archivo.

En FCB la estructura que almacena eso mide 36 bytes por cada archivo abierto en FCB descrito en el DTA.

FCB fue desechado con la aparición de los directorios en el sistema de archivos de MS-DOS, debido a que FCB no permite más que nombres de archivo de 8 caracteres máximo y no servía para almacenar rutas a archivo incluyendo directorios, fue sustituido entonces por los File Handlers (*"Manejadores de archivo"*).

Ch.8 – Main Memory

- Cache sits between main memory and CPU registers
- Base and limit registers define logical address space usable by a process
- Compiled code addresses bind to relocatable addresses
  - Can happen at three different stages
    - Compile time: If memory location known a priori, absolute code can be generated
    - Load time: Must generate relocatable code if memory location not known at compile time
    - Execution time: Binding delayed until run time if the process can be moved during its execution
- Memory-Management Unit (MMU) device that maps virtual to physical address
- Simple scheme uses a relocation register which just adds a base value to address
- Swapping allows total physical memory space of processes to exceed physical memory
  - Def: process swapped out temporarily to backing store then brought back in for continued execution

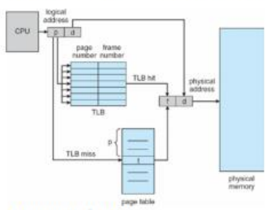


- Backing store: fast disk large enough to accommodate copes of all memory images
- Roll out, roll in: swapping variant for priority-based scheduling.
  - Lower priority process swapped out so that higher priority process can be loaded
- Solutions to Dynamic Storage-Allocation Problem:
  - First-fit: allocate the first hole that is big enough
  - Best-fit: allocate the smallest hole that is big enough (must search entire list)→ smallest leftover hole
  - Worst-fit: allocate the largest hole (search entire list) → largest leftover hole
- External Fragmentation: total memory space exists to satisfy request, but is not contiguous
  - Reduced by compaction: relocate free memory to be together in one block
    - Only possible if relocation is dynamic
- Internal Fragmentation: allocated memory may be slightly larger than requested memory
- Physical memory divided into fixed-sized frames: size is power of 2, between 512 bytes and 16 MB
- Logical memory divided into same sized blocks: pages
- Page table used to translate logical to physical addresses
  - Page number (p): used as an index into a page table
  - Page offset (d): combined with base address to define the physical memory address
- Free-frame list is maintained to keep track of which frames can be allocated

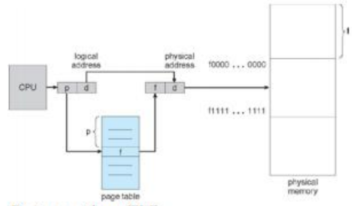
page number	page offset
p	*d*
m − n	*n*

For given logical address space 2^m and page size 2^n

- Transition Look-aside Buffer (TLB) is a CPU cache that memory management hardware uses to improve virtual address translation speed
  - Typically small – 64 to 1024 entries
  - On TLB miss, value loaded to TLB for faster access next time
  - TLB is associative – searched in parallel

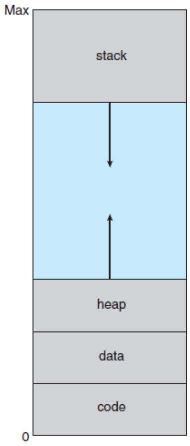


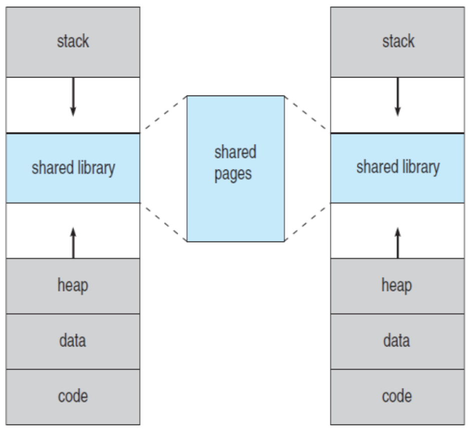
Paging with TLB



Paging without TLB

- Effective Access Time: EAT = $(1 + \varepsilon)\,\alpha + (2 + \varepsilon)(1 - \alpha)$
  - ε = time unit, α = hit ratio
- Valid and invalid bits can be used to protect memory
  - "Valid" if the associated page is in the process' logical address space, so it is a legal page
- Can have multilevel page tables (paged page tables)
- Hashed Page Tables: virtual page number hashed into page table
  - Page table has chain of elements hashing to the same location
  - Each element has (1) virtual page number, (2) value of mapped page frame, (3) a pointer to the next element
  - Search through the chain for virtual page number
- Segment table – maps two-dimensional physical addresses
  - Entries protected with valid bits and r/w/x privileges

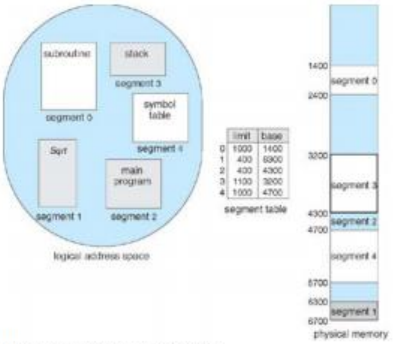




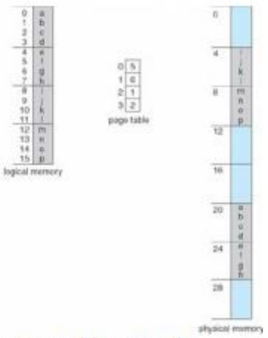
9.2 Virtual address space.

Figure 9.3 Shared library using virtual memory.

Address space: Address space is the amount of memory allocated for all possible addresses for a computational entity, such as a device, a file, a server, or a networked computer. Address space may refer to a range of either physical or virtual addresses accessible to a processor or reserved for a process.



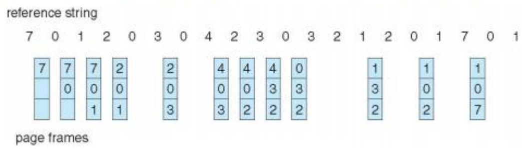
Segmentation example



Page table example

Ch.9 – Virtual Memory

- Virtual memory: separation of user logical memory and physical memory
  - Only part of program needs to be in memory for execution → logical address space > physical address space
  - Allows address spaces to be shared by multiple processes → less swapping
  - Allows pages to be shared during fork(), speeding process creation
- Page fault results from the first time there is a reference to a specific page → traps the OS
  - Must decide to abort if the reference is invalid, or if the desired page is just not in memory yet
    - If the latter: get empty frame, swap page into frame, reset tables to indicate page now in memory, set validation bit, restart instruction that caused the page fault
  - If an instruction accesses multiple pages near each other → less "pain" because of locality of reference
- Demand Paging only brings a page into memory when it is needed → less I/O and memory needed
  - Lazy swapper – never swaps a page into memory unless page will be needed
  - Could result in a lot of page-faults
  - Performance: EAT = [(1-p)*memory access + p*(page fault overhead + swap page out + swap page in + restart overhead)]; where Page Fault Rate 0 ″ p ″ 1
    - if p = 0, no page faults; if p = 1, every reference is a fault
  - Can optimize demand paging by loading entire process image to swap space at process load time
- Pure Demand Paging: process starts with no pages in memory
- Copy-on-Write (COW) allows both parent and child processes to initially share the same pages in memory
  - If either process modifies a shared page, only then is the page copied
- Modify (dirty) bit can be used to reduce overhead of page transfers → only modified pages written to disk
- When a page is replaced, write to disk if it has been marked dirty and swap in desired page
- Pages can be replaced using different algorithms: FIFO, LRU (below)
  - Stack can be used to record the most recent page references (LRU is a "stack" algorithm)

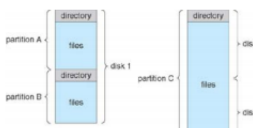


  - Second chance algorithm uses a reference bit
    - If 1, decrement and leave in memory
    - If 0, replace next page
- Fixed page allocation: Proportional allocation – Allocate according to size of process
  - s_i = size of process P_i, $S = \Sigma s_i$, m = total number of frames, a_i – allocation for P_i
  - $a_i = (s_i/S)*m$
- Global replacement: process selects a replacement frame from set of all frames
  - One process can take frame from another
  - Process execution time can vary greatly
  - Greater throughput
- Local replacement: each process selects from only its own set of allocated frames
  - More consistent performance
  - Possible under-utilization of memory
- Page-fault rate is very high if a process does not have "enough" pages
  - Thrashing: a process is busy swapping pages in and out → minimal work is actually being performed
- Memory-mapped file I/O allows file I/O to be treated as routine memory access by mapping a disk block to a in memory
- I/O Interlock: Pages must sometimes be locked into memory

Ch.11 – File-System Interface

file type	usual extension	function
executable	exe, com, bin or none	ready-to-run machine-language program
object	obj, o	compiled, machine language, not linked
source code	c, cc, java, pas, asm, a	source code in various languages
batch	bat, sh	commands to the command interpreter
text	txt, doc	textual data, documents
word processor	wp, tex, rtf, doc	various word-processor formats
library	lib, a, so, dll	libraries of routines for programmers
print or view	ps, pdf, jpg	ASCII or binary file in a format for printing or viewing
archive	arc, zip, tar	related files grouped into one file, sometimes compressed, for archiving or storage
multimedia	mpeg, mov, rm, mp3, avi	binary file containing audio or A/V information

- File – Uniform logical view of information storage (no matter the medium)
  - Mapped onto physical devices (usually nonvolatile)
  - Smallest allotment of nameable storage
  - Types: Data (numeric, character, binary), Program, Free form, Structured
  - Structure decided by OS and/or program/programmer
- Attributes:
  - Name: Only info in human-readable form
  - Identifier: Unique tag, identifies file within the file system
  - Type, Size
  - Location: pointer to file location
  - Time, date, user identification
- File is an abstract data type
- Operations: create, write, read, reposition within file, delete, truncate
- Global table maintained containing process-independent open file information: open-file table
  - Per-process open file table contains pertinent info, plus pointer to entry in global open file table
- Open file locking: mediates access to a file (shared or exclusive)
  - Mandatory – access denied depending on locks held and requested
  - Advisory – process can find status of locks and decide what to do
- File type can indicate internal file structure
- Access Methods: Sequential access, direct access
  - Sequential Access: tape model of a file
  - Direct Access: random access, relative access
- Disk can be subdivided into partitions; disks or partitions can be RAID protected against failure.
  - Can be used raw without a file-system or formatted with a file system
  - Partitions also knows as minidisks, slices

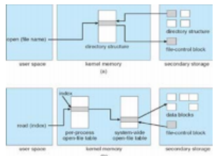


File-System Organizatio

- Volume contains file system: also tracks file system's info in device directory or volume table of contents
- File system can be general or special-purpose. Some special purpose FS:
  - tmpfs – temporary file system in volatile memory
  - objfs – virtual file system that gives debuggers access to kernel symbols
  - ctfs – virtual file system that maintains info to manage which processes start when system boots
  - lofs – loop back file system allows one file system to be accessed in place of another
  - procfs – virtual file system that presents information on all processes as a file system
- Directory is similar to symbol table – translating file names into their directory entries
  - Should be efficient, convenient to users, logical grouping
  - Tree structured is most popular – allows for grouping
  - Commands for manipulating: remove – rm<file-name> ; make new sub directory - mkdir<dir-name>
- Current directory: default location for activities – can also specify a path to perform activities in
- Acyclic-graph directories adds ability to directly share directories between users
  - Acyclic can be guaranteed by: only allowing shared files, not shared sub directories; garbage collection; mechanism to check whether new links are OK
- File system must be mounted before it can be accessed – kernel data structure keeps track of mount points
- In a file sharing system User IDs and Group IDs help identify a user's permissions
- Client-server allows multiple clients to mount remote file systems from servers – NFS (UNIX), CIFS (Windows)
- Consistency semantics specify how multiple users are to access a shared file simultaneously – similar to synchronization algorithms from Ch.7
  - One way of protection is Controlled Access: when file created, determine r/w/x access for users/groups

Ch.12 – File System Implementation

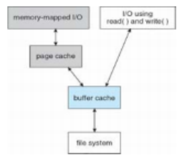
- File system resides on secondary storage – disks; file system is organized into layers →
- File control block: storage structure consisting of information about a file (exist per-file)
- Device driver: controls the physical device; manage I/O devices
- File organization module: understands files, logical addresses, and physical blocks
  - Translates logical block number to physical block number
  - Manages free space, disk allocation
- Logical file system: manages metadata information – maintains file control blocks
- Boot control block: contains info needed by system to boot OS from volume
- Volume control block: contains volume details; ex: total # blocks, # free blocks, block size, free block pointers
- Root partition: contains OS; mounted at boot time
- For all partitions, system is consistency checked at mount time
  - Check metadata for correctness – only allow mount to occur if so
- Virtual file systems provide object-oriented way of implementing file systems
- Directories can be implemented as Linear Lists or Hash Tables
  - Linear list of file names with pointer to data blocks – simple but slow
  - Hash table – linear list with hash data structure – decreased search time
    - Good if entries are fixed size
    - Collisions can occur in hash tables when two file names hash to same location



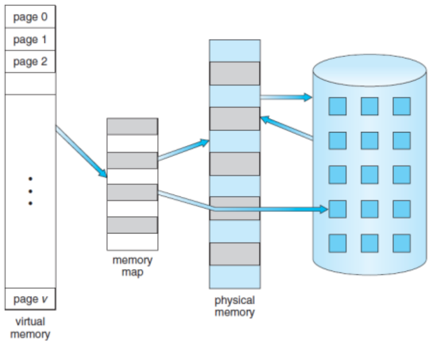
(a) open() *(b) read()*

- Contiguous allocation: each file occupies set of contiguous blocks
  - Simple, best performance in most cases; problem – finding space for file, external fragmentation
  - Extent based file systems are modified contiguous allocation schemes – extent is allocated for file allocation
- Linked Allocation: each file is a linked list of blocks – no external fragmentation
  - Locating a block can take many I/Os and disk seeks
- Indexed Allocation: each file has its own index block(s) of pointers to its data blocks
  - Need index table; can be random access; dynamic access without external fragmentation but has overhead
- Best methods: linked good for sequential, not random; contiguous good for sequential and random
- File system maintains free-space list to track available blocks/clusters
- Bit vector or bit map (n blocks): block number calculation → (#bits/word)*(# 0-value words)+(offset for 1st bit)
-
  - Example:

    block size = 4KB = 212 bytes
    disk size = 240 bytes (1 terabyte)
    n = 240/212 = 228 bits (or 256 MB)
    if clusters of 4 blocks -> 64MB of memory



- Space maps (used in ZFS) divide device space into metaslab units and manages metaslabs
  - Each metaslab has associated space map
- Buffer cache – separate section of main memory for frequently used blocks
- Synchronous writes sometimes requested by apps or needed by OS – no buffering
-
  - Asynchronous writes are more common, buffer-able, faster
- Free-behind and read-ahead techniques to optimize sequential access
- Page cache caches pages rather than disk blocks using virtual memory techniques and addresses
  - Memory mapped I/O uses page cache while routine I/O through file system uses buffer (disk) cache
- Unified buffer cache: uses same page cache to cache both memory-mapped pages and ordinary file system I/O to avoid double caching



ıre 9.1 Diagram showing virtual memory that is larger than physical men

swapping is a mechanism in which a process can be swapped temporarily out of memory to a backing store (Swap OUt) and then brought back into memory for continued execution.(Swap In). In other words, when the amount of physical memory (RAM) is full

page table when some pages are not in memory

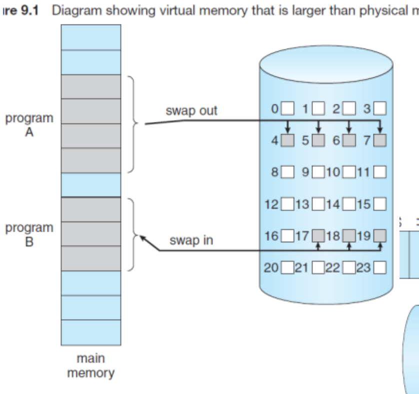


Figure 9.4 Transfer of a paged memory to contiguous disk space.