**Port:** Connection point for a device

**BUS:** Move data around the system between points

PCI: Most common and fast

Expansion bus: Used to connect slow devices

**Controller:** Control and operate a port, bus or device

Devices have specified registers to place commands, read data or write data

**Polling** is the process where the CPU waits for an external device to enter ready state before passing commands to it.

**Busy-wait** is where a process repeatedly checks to see if a condition is true.

**Interrupts:** Used to inform CPU that a certain event has occurred

**Interrupt handler:** Handles all interrupts and can mask some low priority ones

**Direct memory access** (**DMA**) allows hardware subsystems to access main memory independent of the CPU. Needs a **DMA Controller** that has source and destination addresses and steals **PCI BUS** time from CPU when it is not being used

**CPU->Driver->Device Controller->DMA Controller->CPU**

**Blocking IO -** process suspended until I/O completed

**Nonblocking I/O:** call returns as much as available **Implemented via multi-threading**

**Asynchronous -** process runs while I/O executes

**Vectored I/O** allows one system call to perform multiple I/O operations by having gathered collection. Decreases context switching

**Buffering** - store data in main memory while transferring between devices

**Caching:** Store data in fast memory

**Spooling:** Hold instructions for device e.g. printing

**Device Reservation:** reserve device for some time. May cause dead lock

**STREAM** – a full-duplex communication channel between a user-level process and a device.

**STEPS TO IMPROVE PERFORMANCE:**

* Reduce number of context switches
* Reduce data copying
* Reduce interrupts by using large transfers, smart controllers, polling
* Use DMA
* Use smarter hardware devices
* Balance CPU, memory, bus, and I/O performance for highest throughput
* Move user-mode processes / daemons to kernel threads

**Magnetic Disks:** Rotate at 60 to 200 times per second

**Transfer rate** is rate at which data flow between drive and computer

**Positioning time** (**random-access time**) = SEEK TIME + ROTATIONAL LATENCY

**Seek time** is time to move disk arm to cylinder with desired sector

**Rotational latency** time is for desired sector to rotate under the disk head

**Head crash** results from disk head contacting the disk surface

**SECTOR 0: This is the first sector on the outer most track in the outer most cylinder**

**Buses to connect Drives: EIDE, ATA, SATA, USB, Fiber Channel, SCSI**

**Network Attached Storage:** Storage over the network. Using **NFS and Internet File System.**

**Log structured (or journaling) file systems record** each update to the file system as a transaction .All transactions are written to a log. A transaction is considered committed once it is written to the log. The transactions in the log are asynchronously written to the file system. When the file system is modified, the transaction is removed from the log. If the file system crashes, all remaining transactions in the log must still be performed.

**Disk bandwidth** is the total number of bytes transferred, divided by the total time between the first request for service and the completion of the last transfer

**DISK SCHEDULING:**

**FCFS: First Come First Serve:**  Sectors are accessed in the order they are requested

**SSTF: Shortest Seek Time First:** Selects request with minimal seek time for the current head position. May cause starvation

**SCAN (Elevator): S**can down towards the nearest end and then when it hits the bottom it scans up servicing the requests that it didn't get going down. If a request comes in after it has been scanned it will not be serviced until the process comes back down or moves back up.

**C-SCAN:** The head moves from one end of the disk to the other, servicing requests as it goes. When it reaches the other end, however, it immediately returns to the beginning of the disk, without servicing any requests on the return trip

**C-LOOK:** Arm only goes as far as the last request in each direction, then reverses direction immediately, without first going all the way to the end of the disk

**Low-level formatting**, or **physical formatting** — Dividing a disk into sectors that the disk controller can read and write

**Partition** the disk into one or more groups of cylinders

**Logical formatting** or “making a file system”

**Bootstrap Loader:** Saved in the ROM and contains instructions and memory addresses from where to load the OS

**Swap-Space:** This is when Virtual Memory uses the disk as an extension for the main memory

**RAID** (**Redundant Array of Independent Disks**) combines multiple physical disk drive components into one or more logical units for data redundancy and performance improvement.

**Mirroring or shadowing (RAID 1)** keeps duplicate of each disk.

**Striped mirrors (RAID 1+0)** or mirrored stripes (RAID 0+1) provides high performance and high reliability.

**Block interleaved parity (RAID 4, 5, 6)** uses much less redundancy.

**HSM (Hierarchal Storage Management):**  Uses tertiary storage mediums to extend the file system.

**Sustained bandwidth** – average data rate during a large transfer; # of bytes/transfer time. Data rate when the data stream is flowing

**Effective bandwidth** – average over the entire I/O time, including seek () or locate (), and cartridge switching  
Drive’s overall data rate

**Boot Control Block:**  contains info to boot OS from a volume

**Volume Control Block:**  Contains info about that volume

**File Control Block:**  One for each file containing information about the file e.g. file size, permissions, owner, blocks and pointers to other blocks. Can be stored in directory structure or independently.

**File Descriptor table:** One for each process and has an array of pointers each pointing to a file opened by that process pointing to the **Open File Table**.

**Open File Table:** Contains pointers to all instances a file has been opened. Contains pointer to **Inode Table**, byte offset and how the file is used and the number of pointers from the FDT

**Inode Table:** One entry per open file and contains the files inode and number of pointers from OFT. Only one per system

**Inode:**  One per file and has permission bits, location addresses, UID, GID of owners

TO OPEN FILE: Start from INODE going out towards FTD

TO CLOSE FILE: Start from FTD going inwards to INODE

**Directory implementation:**

Linear list: Simple but time consuming

Hash Table: quick access but many collisions and fixed size

**Memory Block Allocation:** how disk blocks are allocated to files

**Contiguous:** Each file occupies a set of adjacent blocks. Fixed size for files, wasted space. New algorithm allows splitting file over multiple adjacent blocks

**Linked:** File is split and saved as linked blocks. Need only first block. If one block is lost, file may become corrupted

**Indexed:** File is stored in random blocks and each block is indexed in the index table

In actual, File systems use a combination of all these allocation schemes to save files.

**Lazy Swapper:**  Bring a page into memory only when needed

Each page table entry has a valid bit to show that the page exists in main memory or not

**Copy on Write:**  Two processes share the same page in memory. In case it is changed by any one, the page is then copied, and the processes then refer their own copies.

* In case the main memory is full, and more pages are to be swapped in, the oldest unused pages are removed first.
* Use Locality of reference to load pages into memory to reduce the number of page faults.
* Use dirty bits to indicate changed pages. Only pages with dirty bits are then written to disk.

**PAGE REPLACEMENT ALGORITHMS:**

**FIFO:**  Pages are loaded into main memory as they are requested

**Optimal:** Pages that will not be used for the longest time are swapped out first. (Theory)

**LRU (Least Recently used):**  Pages that have not been used for a long time are swapped out first

**Page Counting:** Use counting bits to keep a counter of the number of times a page has been used.

**Page Buffering:**  Keep some free frames always to avoid having to concurrently write out and load in new pages.

**PROCESS FRAME ALLOCATION:**

Each process needs a certain number of frames. Two major ways:

**Priority:** Frames are allocated based on priority

**Fixed:** Each process is given equal frames

**NUMA:** Non-Uniform Memory Access

**Thrashing:** Process is waiting for CPU to swap pages out and load required pages into memory.

**Working Set Model:** Keeps track of the different pages referenced in most recent working set window.

**Memory mapped I/O:**  This allows I/O to be treated as routine memory access by mapping a disk block to a page in memory.

**Buddy System:** Memory is allocated in powers of two. It is divided into two buddies of equal size until the block size is just big enough to accommodate the page, but it causes fragmentations

**Slab Allocator:** A slab is a collection of contiguous pages. **Cache** consists of multiple slabs. It allows fast access and reduces fragmentation.

**Base and Limit registers:**  Base register defines the start address and limit defines the length of the register.

**Logical Address Space** is generated by the CPU

**Physical Address Space** is the address in main memory

Logical and physical are the same in compile time and loading time but differ in execution

**Memory Management Unit:** Hardware device to map Logical addresses to physical addresses.

**Dynamic Linking:** Linking is done at run time. A **Stub** is used to locate the library code to execute and replaces itself with the address of the library code.

**Backing Store:** is a large disk to hold copies of main memory for all processes when the process is swapped out of the CPU

**Dynamic Memory Allocation:**

**Best Fit:** Allocate in the smallest hole that is big enough to hold the size

**Worst Fit:** Allocate the largest hole

**First Fit:** Allocated in the first hole big enough to hold the size

**External Fragmentation:** Total memory space exists to satisfy a request but is split into small blocks. Reduced by compaction: I.e. rearranging memory to make free blocks adjacent to each other.

**Internal Fragmentation:** Allocated memory may be larger that requested memory and the left-over space is not used.

**Segmentation:** Memory management scheme that supports user view of memory. Each segment is a logical unit in the code such as a variable, function, object, class, array etc.

**Segment Table:** Maps physical address. Each table entry has a base (start address of segment) and a limit (length of segment)

**Segment Table Base Register (STBR)**: Points to the segments location in memory

**Segment Table Length Register (STLR):** Shows the number of segments used by a program. Segment is legal only if its number s < STLR

**Frames:** Divided Physical memory blocks (size is of power 2)

**Pages:** Divided Logical memory blocks

**Page Table:** Stores addresses of pages mapped to frames

**Page number:** an index to a page table

**Page Offset:** Size of page

Page table is kept in main memory

**Page-table base register (PTBR)** points to the page table

**Page-table length register (PRLR)** indicates size of the page table

**Hit Ratio:** percentage of times that a page number is found in the associative registers

Hit ratio = α

Associative Lookup = ε time unit

**Effective Access Time** (EAT)

EAT = (1 + ε) α + (2 + ε)(1 – α)

= 2 + ε – α

**Valid Bit:** added to each frame entry to indicate valid or invalid.

**Shared Code:** One copy of the code is shared amongst processes

**Private Code:**  each process keeps a separate copy of code and data

**Hierarchal Page Tables:** Break up logical addresses into multiple page tables. The deeper the hierarchy, the longer the time to access the memory. Allows referencing large amount of memory

**Hashed Page Tables:** The virtual page number is hashed into a hash table index. Fixed size though

**Inverted age Tables:** One entry for each real page of main memory. Reduces page table size

Intel Pentium supports with segmentation and segmentation with paging

**DEADLOCKS:** Occurs when processes hold resources waiting for the other process to free the resource. **(most OSes do not prevent or deal with deadlocks)**

**DEADLOCK SCENARIOS:**

* **Mutual exclusion:** only one process at a time can use a resource
* **Hold and wait:** a process holding at least one resource is waiting to acquire additional resources held by other processes.
* **No preemption:** a resource can be released only voluntarily by the process holding it, after that process has completed its task
* **Circular wait:** there exists a set {*P*0, *P*1, …, *P*n} of waiting processes such that *P*0 is waiting for a resource that is held by *P*1, *P*1 is waiting for a resource that is held by *P*2, …, *Pn*–1 is waiting for a resource that is held by *P*n, and *P*n is waiting for a resource that is held by *P*0

**DEADLOCK PREVENTION:**

**Mutual Exclusion –** Allocate resources at startup

**Hold and Wait –** must guarantee that whenever a process requests a resource, it does not hold any other resources. Low resource utilization; starvation possible

**No Preemption:** If a process that is holding some resources requests another resource that cannot be immediately allocated to it, then all resources currently being held are released. Preempted resources are added to the list of resources for which the process is waiting. Process will be restarted only when it can regain its old resources, as well as the new ones that it is requesting

**Circular Wait** – impose a total ordering of all resource types, and require that each process requests resources in an increasing order of enumeration

**DEADLOCK AVOIDANCE:**

* Each process declares its required resources before

System is in **safe state** if there exists a sequence <*P1, P2, …, Pn*> of ALL the processes is the systems such that for each Pi, the resources that Pi can still request can be satisfied by currently available resources + resources held by all the *Pj*, with *j* < *I*

That is:

* If Pi resource needs are not immediately available, then *Pi* can wait until all *Pj* have finished.
* When *Pj* is finished, *Pi* can obtain needed resources, execute, return allocated resources, and terminate.
* When *Pi* terminates, *Pi* +1 can obtain its needed resources, and so on.

**Critical Section:** This is section of code that cannot be executed by more than one process at the same time.

**Rules to form a critical section**

1. No two processes may be simultaneously inside their CS
2. No assumptions are made about relative process speed or number of CPU’s
3. A process outside a CS should not block other processes
4. No process should be able to wait forever before entering its CS

Critical section can lead to Starvation and Deadlocks. **Starvation** meaning a process may not get to run the Critical section.

**Deadlock** means that two processes are waiting for an event that will never occur.

**Critical Section Solutions**

**Mutual Exclusion -** If process Pi is executing in its critical section, then no other processes can be executing in their critical sections.

**Progress** - If no process is executing in its critical section and there exist some processes that wish to enter their critical section, then the selection of the processes that will enter the critical section next cannot be postponed indefinitely.

**Bounded Waiting –** A bound must exist on the number of times that other processes are allowed to enter their critical sections after a process has made a request to enter its critical section and before that request is granted.

**Deadlock Solutions:**

* **Semaphores:**
* **Mutex Locks**
* **Monitors**

**An OS is a program that acts as an intermediary between a user of a computer and the computer hardware**

**OS is:**

**Resource allocator**: decides between conflicting requests for efficient and fair resource use

**Control program**: controls execution of programs to prevent errors and improper use of computer

**Kernel**: the one program running always on the computer

**Bootstrap program**: loaded at power-up or reboot stored in ROM or EPROM to load kernel

**I/O and CPU can execute concurrently**

Device controllers inform CPU that it is finished w/ operation by causing an interrupt.

Interrupts transfer control to the interrupt service routine generally, through the interrupt vector, which contains the addresses of all the service routines

**Interrupt handler** receives interrupts **Maskable** to ignore or delay some interrupts

**Interrupt vector** to dispatch interrupt to correct handler

**OS & MEMORY MANAGEMENT:**

**Keeping track of which parts of memory** are currently being used and by whom

**Deciding which processes** (or parts thereof) and data to move into and out of memory

**Allocating and deallocating** memory space as needed

**Incoming interrupts are disabled** while another interrupt is being processed

**Trap** is a software generated interrupt caused by error or user request

**Polled interrupts do not specify which device generated an interrupt while a vectored interrupt specifies which device generated an interrupt**

**Main memory** – random access, volatile

**Secondary storage** – extension of main memory that provides large non-volatile storage

**Disk** – divided into tracks which are subdivided into sectors. Disk controller determines logical interaction

**Caching** – copying information into faster storage system

Multiprocessor Systems: Increased throughput, economy of scale, increased reliability. **Can be asymmetric or symmetric**

**Context Switch:** Done when CPU switches from one process to another. Saves state of currently executing process and loads state of process to be executed

**Protection** – mechanism for controlling access of processes or users to resources defined by the OS

**Security** – defense of a system against attacks

User IDs (UID), one per user, and Group IDs, determine which users and groups of users have which privileges

**Process control system calls**: end, abort, load, execute, create/terminate process, wait, and allocate/free memory

**File management system calls**: create/delete files, open/close file, read, write, get/set attributes

**Device management system calls**: request/release device, read, write, and logically attach/detach devices

**Information maintenance system calls**: get/set time, get/set system data, get/set process/file/device attributes

**Communications system calls**: create/delete communication connection, send/receive, and transfer status information

**Virtual machine**: uses layered approach, treats hardware and the OS kernel as though they were all hardware.

**Application failures** can generate core dump file capturing memory of the process

**Operating system failure** can generate crash dump file containing kernel memory

Each process contains a program counter, stack, and data section.

**Text section**: program code itself

**Stack**: temporary data (function parameters, return addresses, local variables)

**Data section**: global variables

**Heap**: contains memory dynamically allocated during run-time

**Process Control Block (PCB)**: contains information associated with each process: process state, PC, CPU registers, scheduling information, accounting information, I/O status information

**Program Counter:** Specifies address of next instruction to execute.

**Types of processes:**

**I/O Bound**: spends more time doing I/O than computations, manyshort CPU bursts

**CPU Bound**: spends more time doing computations, few verylong CPU bursts

**When CPU switches to another process, the system must save the state of the old process (to PCB) and load the saved state (from PCB) for the new process via a context switch**

◦ **Fork** () system call creates new process

▪ **Exec** () system call used after a fork to replace the processes' memory space with a new program

**Advantages of multiprocessing:** include:

* Increased throughput (more work done in less time)
* Economy of scale (less cost due to peripheral sharing)
* Increased reliability (failure of one processor will not halt the system)

**Clustered Systems:**

**Asymmetric clustering** has one machine in hot-standby mode

**Symmetric clustering** has multiple nodes running applications, monitoring each other

**Multi-threading models**: Many-to-One, One-to-One, Many-to-Many

**Many-to-One**: Many user-level threads mapped to single kernel thread

**One-to-One**: Each user-level thread maps to kernel thread

**Many-to-Many**: Many user-level threads mapped too many kernel threads

**Advantages of Multi-Threading:**

**Responsiveness -** Interactive applications can be performing two tasks at the same time (rendering, spell checking)

**Resource Sharing -** Sharing resources between threads is easy

**Economy -** Resource allocation between threads is fast (no protection issues)

**Utilization of MP Architectures -** seamlessly assign multiple threads to multiple processors

**Processes have copies of all code and data whereas for Threads, code and data are shared**

**Two models of IPC**

**-Shared memory:** Process share a block of memory. Writing and reading to the block should be mutually exclusive i.e one at a time

**-Message passing:** Processes share a mailbox to send and receive messages. Message passing may be either blocking or non-blocking

**Blocking** is considered **synchronous**

**Blocking send** has the sender block until the message is received

**Blocking receive** has the receiver block until a message is available

**Non-blocking** is considered **asynchronous**

**Non-blocking** send has the sender send the message and continue

**Non-blocking** receive has the receiver receive a valid message or null

**Long-term scheduler (or job scheduler) – selects which processes should be brought into the ready queue**

**Short-term scheduler (or CPU scheduler) – selects which process should be executed next and allocates CPU**

To solve critical section problem each process must ask permission to enter critical section in entry section, follow critical section with exit section and then execute the remainder section

**Semaphore**: Synchronization tool that does not require busy waiting. Standard operations: wait () and signal () ← these are the only operations that can access semaphore

**Semaphore Types:** Counting (unrestricted integer) & Binary (0 or 1)

**Acquire semaphore**

**CRITICAL Section**

**Release semaphore**

**Monitor** is a high-level abstraction that provides a convenient and effective mechanism for process synchronization. Java adds inbuilt monitors using **synchronized** blocks

**Only one process may be active within the monitor at a time. Contains wait and signal functions to notify control process synchronization**

CPU scheduling decisions take place when a process:

Switches from running to waiting (no preemptive)

Switches from running to ready (preemptive)

Switches from waiting to ready (preemptive)

Terminates (no preemptive)

**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

**PROCESS SCHEDULING**

FCFS, SJF, Shortest-Remaining-Time-First (preemptive SJF), Round Robin, Priority

**Determining length of next CPU burst: Exponential Averaging:**

tn = actual length of nth CPU burst

τn+1 = predicted value for the next CPU burst

α, 0 ≤ α ≤ 1 (commonly α set to 1/2)

Define: τn+1 = α\*tn + (1-α)τn

**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 quantum’s 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

**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 as they wait in the queue

**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)

**CPU utilization** – keep the CPU as busy as possible

**Throughput** – # of processes that complete their execution per time unit

**Turnaround time** – amount of time to execute a process

**Waiting time** – amount of time a process has been waiting in the ready queue

**Response time** – amount of time it takes from when a request was submitted until the first response is produced, not output (for time-sharing environment)

**Preemptive**: CPU is emptied/ stopped from executing current process

**POSIX:** **is a family of standards specified by the IEEE Computer Society for maintaining compatibility between operating systems.**

**MULTIPROCESS: C**

#include <stdio.h>

#include <sys/types.h>

#include <unistd.h>

int main (int args, char \*argv[])

{

pid\_t fork\_return;

pid\_t pid;

pid=getpid();

fork\_return = fork ();

// When fork () returns -1, an error happened.

If (fork\_return==0)

// When fork () returns 0, we are in the child process.

{

printf(“\n\nThe values are Child ID = %d, Parent ID=%d\n”, getpid(), getppid());

execl(“/bin/cat”, “cat”, “-b”, “-v”, “-t”, argv[1], 0);

}

else

// When fork () returns a positive number, we are in the parent process

// and the return value is the PID of the newly created child process.

{

wait(NULL);

printf(“\nChild Completes “ );

printf(“\nIn the Parent Process\n”);

printf(“Child Id = %d, Parent ID = %d\n”, getpid(), getppid());

}

return 0;

}

**Processor affinity** – process has affinity for processor on which it is currently running

**Soft affinity: Process can go to any processor**

**Hard affinity: Process can only go to a specific processor**

Embedded Computers run on a real time system

Windows XP masks all interrupts that may change a global variable

**An Operating System (OS)** is a large and complex piece of system software that manages all major computer resources (memory, input/output, processor, processes, files, system security) and some network communication functions

The operating system is divided into several layers (levels), each built on top of lower layers. The bottom layer (layer 0), is the hardware; the highest (layer N) is the user interface such that each uses functions (operations) and services of only lower-level layers

**Job queue** – set of all processes in the system

**Ready queue** – set of all processes residing in main memory, ready and waiting to execute

**Device queues** – set of processes waiting for an I/O device