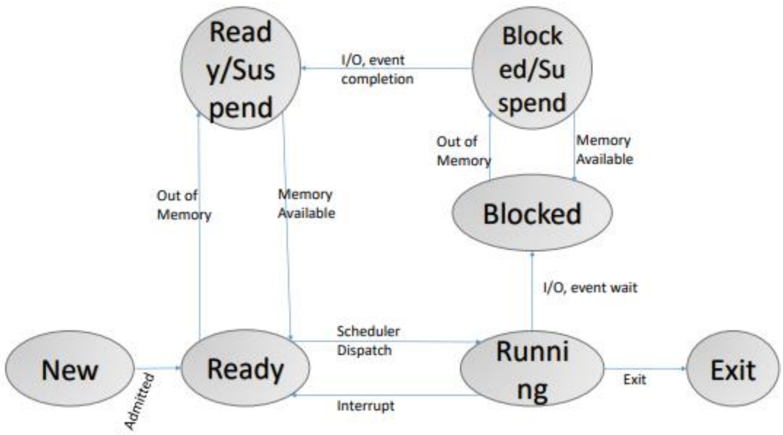
Scheduling

- cpu burst is followed by an io burst
- cpu scheduling decisions may take place when a process:
    - switches from running to waiting state
    - switches from running to ready state
    - wait to ready
    - when a process exits



- cpu schedular : the program which selects which process in the ready queue to load to CPU
- Dispatcher : Program which loads the selected program to the CPU
- Dispatch latency : time between stopping one process and loading the next.

Pre-emption

- If the current running process can be removed by another process it's pre-emptive
    - If it can be removed forcefully, it is non-cooperative
- Race condition can occur in a preemptive system.
    - When data is shared across several processes.

Dispatcher

- Switching context
- Switching to user mode
- Jumping to the proper location in the user program to restart that program.

Scheduling criteria

Scheduling Criteria

- **CPU utilization** – keep the CPU as busy as possible (40% - 90%) real system
- **Throughput** – # of processes that complete their execution per time unit
- **Turnaround time** – amount of time to execute a particular process final output
  (time to get into mem + time in queues + time execution + time I/O)
- **Waiting time** – amount of time a process has been waiting in the ready queue As algorithm only affects the waiting time
- **Response time** – amount of time it takes from when a request was submitted until the first response is produced. Not output. Just response

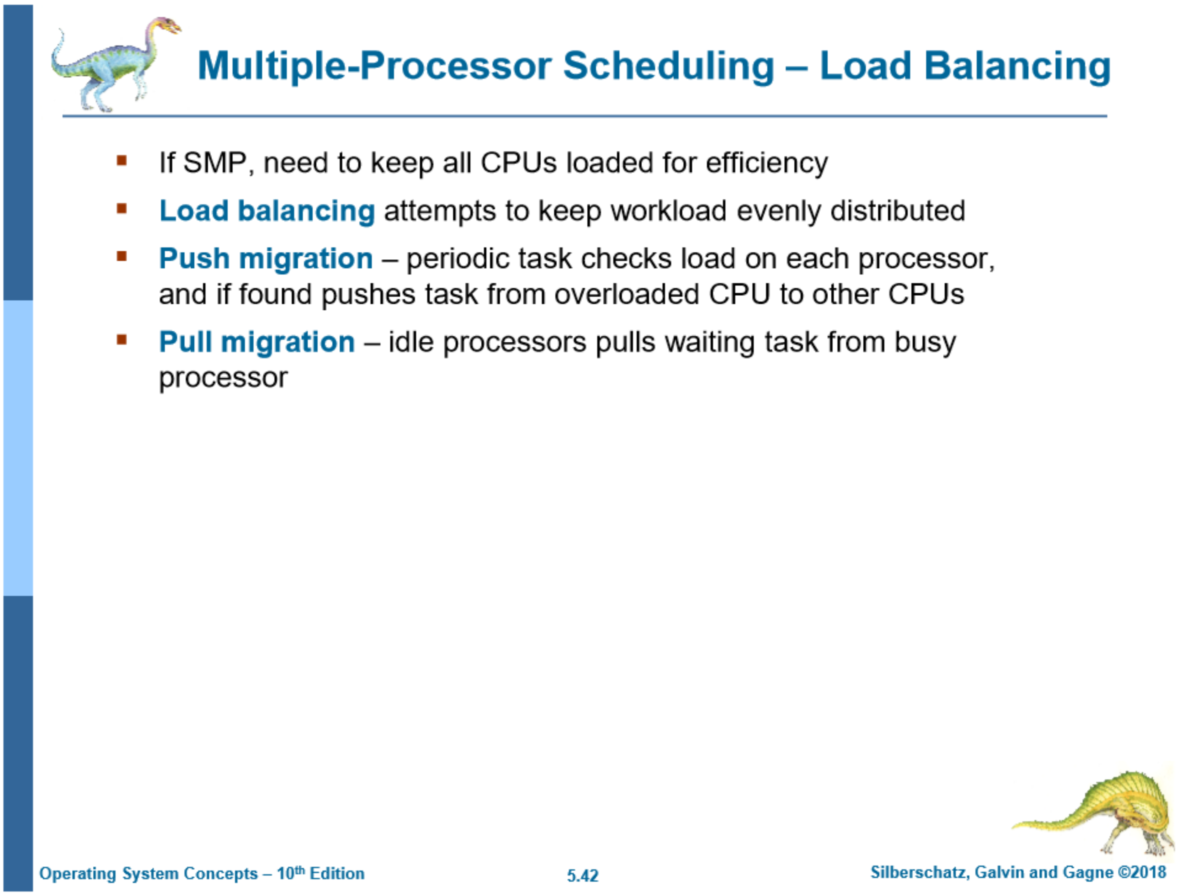
(better alternative to turnaround time to use in interactive systems)
as TA time is dependent on the I/O speed
TA needs full completion
Response does not need that

Scheduling algorithms

- FCFS - Picture on page "Chapter 05 - CPU Scheduling" (Web view)
  - If two processors arrive at the same time, the one with smaller pid gets the cpu first
  - Convoy – Short processes behind long processes
  - Not good for time sharing systems
- Shortest job first (sjf) - Picture on page "Chapter 05 - CPU Scheduling" (Web view)
  - Can only estimate the length of the burst time
- Shortest remaining job first
  - Preemptive version of the above algorithm
  - Behave same as sjf when all the process are arrived to the queue
- Round robin
  - Time sharing system
  - A time quantum is assigned to each thread
  - if q is large -> FCFS
  - if q is small -> overhead
- priority scheduling
  - equal priority -> FCFS
  - every algorithm is a priority scheduling algorithm
  - starvation can occur – solution : aging
    - multilevel queue
      - prioritization based upon process type
    - multilevel feedback queue
      - process can move between multiple queues
      - queues can have different scheduling algorithms
      - aging is implemented in this manner

Operating Systems
Thread Scheduling

- typically done via priority set by programmer



Real-time scheduling

- soft real time – no enforcement / no deadline
- hard real-time – task must be serviced by its deadline
- latency
  - interrupt latency – time from arrival of interrupt to start of routine that service interrupt
  - dispatch latency – time for schedule to take current process off CPU and switch to another

The major differences between a process and a thread are given as follows –

Comparison Basis	Process	Thread
Definition	A process is a program under execution i.e an active program.	A thread is a lightweight process that can be managed independently by a scheduler.
Context switching time	Processes require more time for context switching as they are more heavy.	Threads require less time for context switching as they are lighter than processes.
Memory Sharing	Processes are totally independent and don't share memory.	A thread may share some memory with its peer threads.
Communication	Communication between processes requires more time than between threads.	Communication between threads requires less time than between processes .
Blocked	If a process gets blocked, remaining processes can continue execution.	If a user level thread gets blocked, all of its peer threads also get blocked.
Resource Consumption	Processes require more resources than threads.	Threads generally need less resources than processes.
Dependency	Individual processes are independent of each other.	Threads are parts of a process and so are dependent.
Data and Code sharing	Processes have independent data and code segments.	A thread shares the data segment, code segment, files etc. with its peer threads.
Treatment by OS	All the different processes are treated separately by the operating system.	All user level peer threads are treated as a single task by the operating system.
Time for creation	Processes require more time for creation.	Threads require less time for creation.
Time for termination	Processes require more time for termination.	Threads require less time for termination.

File System

- Directory Structure
    - Nodes containing information about all files.
- File operation
    - Create
    - Write
    - Read
    - Reposition within file – seek
    - Delete
    - Truncate
    - Open
    - Close
- Access methods
    - Sequential – simplest
        - Going step by step
        - Read next, write next and reset
    - Direct
        - Can go to the specific location in the memory
        - read n, write n,

    - position to n
        - read next
        - write next
        - rewrite n; n – relative block number
    - Sequential access in also possible
  - Other access methods

Disk structure

- Two types
  - General file-system
  - Special purpose file system

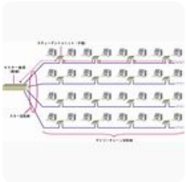
Operations Performed on Directory

- Search for a file
- Create a file
- Delete a file
- List a directory
- Rename a file
- Traverse the file system

- Directory organization goals
  - Efficiency
  - Naming
  - Grouping
- Types of directories
  - Single level – single directory for all users
    - Limitations when files and users increase
    - Naming problem/ grouping problem
  - Two level
    - Separate directory for each user
    - Path name available
    - Search efficiency
    - No grouping
  - Tree structured
    - Most common
    - Path name available for files
  - Acyclic graph
    - Shared subdirectories and files are available
    - One copy, accessed by all users
    - Limitations
      - Dangling point when one user deletes a shared file and the other is not updated about the file
      - Solution – back pointers using a daisy chain

Daisy Chain

Electrical Engineering



In electrical and electronic engineering, a daisy chain is a wiring scheme in which multiple devices are wired together in sequence or in a ring, similar to a garland of daisy flowers. Daisy chains may be used for power, analog signals, digital data, or a combination of them.

-
- Link – a pointer to an existing file – ignores traversing through the tree.
- Resolve the link - path

- General graph directory
  - Possible defects
    - Self-referencing
    - Cycles –

- How do we guarantee no cycles?
  - Allow only links to files not subdirectories
  - **Garbage collection**
  - Every time a new link is added use a cycle detection algorithm to determine whether it is OK

very time consuming for a large directory

- File-system Structure
  - Resides on secondary storage
  - File control block – Storage structure consisting of information about a file
- File System Layers
  - Devices
  - I/O control – Device drivers manage devices
  - Basic File system –
    - Translates commands to I/O control
    - Manages memory buffers (Hold data in transit) and caches (Hold frequently used)
  - File organization module –
    - translates logical block to physical block
    - Manages free space, disk allocation
  - Logical file system –
    - Manages metadata information
    - Directory management
    - Protection
  - Application programs
- Boot control block – contains info needed to boot OS from a volume
- Volume control block – contains information about the volume
- File control block –
  - One FCB per file
  - Contains information about the file
- In-memory file system structures
  - Mount tables – Store file system mounts, mount points, file system types
  - System-wide open-file table – contains a copy of FCB of open files
  - Per-process open-file table – contains a pointer to appropriate entries in System wide table
- Directory Implementation

- Linear list
  - Simple to program
  - Not efficient
- Hash table
  - Decrease search time
  - Collisions – two names can hash to same location
  - Good only if entries are fixed sized
  - Else use chained overflow

- Memory allocation method
  - Contiguous
    - Best performance in most cases
    - Simple
    - Problems
      - Finding space in disk
      - Knowing file size
      - External fragmentation
  - Extent based systems – modified contiguous
    - Allocate disk blocks in extents
    - Extent is a contiguous block of disks
    - A file consists of one or more extents
  - Linked
    - Each file is a linked list of blocks.
    - File ends at nil pointer
    - No external fragmentation
    - Reliability is less
  - File Allocation Table
    - Indexed allocation method
      - Each file has an index table with pointers to its data blocks.
- Performance
  - Contiguous – good for sequential and random
  - Linked – good for sequential
  - Indexed – more complex
    - Clustering can improve performance
- Free-space management
  - Maintain a free-space list
  - A bit vector.
  - Easy to get contiguous files
  - Linked list (Free list)
    - No waste
    - Grouping
    - Counting
      - Keep address of first free block and next number of free blocks
      - Free space list has entries containing addresses and counts
  - Space maps
    - Used in ZFS
    - Divides device space into **metaslab** units
    - Each metaslab has associated space map
    - Records log to a file instead of filesystem
    - Combines contiguous free blocks into single entry
- TRIMming unused blocks
  - Never mechanism to inform NVM devices that a page is free

    - Then can be garbage collected or erased
- Efficiency and Performance
    - Efficiency Depends on
        - Disk allocation
        - Directory algorithms

- Disk allocation and directory algorithms
- Types of data kept in file's directory entry
- Pre-allocation or as-needed allocation of metadata structures
- Fixed-size or varying-size data structures

- Performance
    - Keeping data and metadata close

- Page cache
    - Caches pages rather than disk blocks
    - Memory mapped I/O uses page cache
    - Routine I/O uses buffer (Disk) cache
- Unified buffer cache
    - Use the same page cache in Memory-mapped I/O and Ordinary I/O
    - Avoid double caching
- Recovery
    - Consistency checking compares directory structure with data blocks
        - Slow
        - Can fail
- Log structured file systems
    - Record each metadata update as a transaction
    - All transactions are written to a log
    - Faster recovery from crashes

File system Internals

- Partitions and mounting
    - Root partition – contains the OS, mounted at boot time
    - Other - can be mounted automatically or manually on mount points
    - File system consistency checked at mount time
- Virtual File systems
    - Object oriented way of file systems
    - Can use the same API for different file systems
    - Linux has 4 object types in VFS
        - Inode
        - File
        - Superblock
        - Dentry
- Remote file systems
    - Sharing files across a network
    - Method 1 – FTP (Manually sharing each file)

  - Method 2 – Distributed File System (DFS)
    - Remote directories are visible from the local machine
  - Method 3 – WWW
    - Use browsers to locate files
    - Has anonymous access
- Client – server model
- Distributed Information systems (DNS, NIS, CIFS)

- Consistency semantics
  - For evaluating file-sharing systems
  - File sessions

Operating Systems

Synchronization Tools

Background

- Concurrent access may result in data inconsistency
- Processes can either
    - Share a logical address space (Code + data)
    - Be allowed to share data through only files or messages

Critical Section Problem

- Changing common data
- When one process is in critical section, no other process should be in critical section
- Permission requested at entry section
- Requirements
    - Mutual Exclusion – No two process
    - Progress – No process should be starved
    - Bounded waiting – When one process requests access to enter critical section, there should be a limit that other processes are allowed to enter their critical section before this one

Peterson's Solution

- Two process solution
- Assume load and store are atomic
- Two processes share two variables turn, flag[2]
- Humble algorithm
- All three requirements are met
- Drawbacks
    - Cannot be used for more than two processes.
    - Cannot be used in multithreaded operations.
- Use a memory barrier to optimize Peterson's
    - Memory models – Guarantees that computer architecture provides to application programs,
    - Strongly ordered – Memory modification of one processor is immediately visible to all other processes.
    - Weakly ordered – may not be immediately visible.
    - Memory barrier – Enforces those changes should be visible to other processes.

Hardware support for synchronization

- Uniprocessors – Could disable interrupts
- Others – Three forms of support
    - Hardware instructions
        - Test and set – Does not solve critical section
            - Returns the original value of the parameter
            - Set the new value to true
            - No bounded wait
            - Can cause starvations
        - Compare and swap
            - Returns the original value of the parameter
            - Set new value if the current is expected
    - Atomic variables
        - Gives atomic updates on basic data types

Mutex Locks

- Simplest

- Boolean variable – locked or not
- Requires busy waiting
- Acquire and release locks using hardware atomic instructions

Semaphores

- Integer variable
- Wait and signal operations
- Types
  - Counting – To use over a resource having multiple instances. Initialized to the number of instances.
  - Binary – same as mutex
- Implementation with no busy wait
  - Have a waiting queue with each semaphore.
  - Block and wait operations
  - Can have deadlocks and starvations.

Monitors

- A higher-level abstraction. (Itself is an abstract data type)
- Only one process may be active within the monitor at a time. (Mutual exclusion within the monitor)
- Shared variables are defined within the monitor
- Procedures within a monitor can access only local variables and vice versa.

Liveness

Evaluation