Scheduling Algorithms

FCFS

- Process added first in ready queue should be scheduled first.
- Non-preemptive scheduling
- Scheduler is invoked when process is terminated, blocked or gives up CPU is ready for execution.
- Convoy Effect: Larger processes slow down execution of other processes.

SJF

- Process with lowest burst time is scheduled first.
- Non-preemptive scheduling
- Minimum waiting time

SRTF - Shortest Remaining Time First

- Similar to SJF - but Preemptive scheduling
- Minimum waiting time

Priority

- Each process is associated with some priority level. Usually lower the number, higher is the priority.
- Preemptive scheduling or Non Preemptive scheduling
- Starvation
    - Problem may arise in priority scheduling.
    - Process not getting CPU time due to other high priority processes.
    - Process is in ready state (ready queue).
    - May be handled with aging -- dynamically increasing priority of the process.

Round-Robin

- Preemptive scheduling
- Process is assigned a time quantum/slice.
- Once time slice is completed/expired, then process is forcibly preempted and other process is scheduled.
- Min response time.

Fair-share

- CPU time is divided into epoch times.
- Each ready process gets some time share in each epoch time.
- Process is assigned a time share in proportion with its priority.
- In Linux, processes with time-sharing (TS) class have nice value. Range of nice value is -20 (highest priority) to +19 (lowest priority).

IPC overview

- A process cannot access of memory of another process directly. OS provides IPC mechanisms so that processes can communicate with each other.
- IPC models
    - Shared memory model
        - Processes write/read from the memory region accessible to both the processes.
        - OS only provides access to the shared memory region.
    - Message passing model
        - Process send message to the OS and the other process receives message from the OS.
        - This is slower compared to shared memory model.
- Unix/Linux IPC mechanisms
    - Signals
    - Shared memory
    - Message queue
    - Pipe
    - Socket

Synchronization

- Multiple processes accessing same resource at the same time, is known as "race condition".
- When race condition occurs, resource may get corrupted (unexpected results).
- Peterson's problem, if two processes are trying to modify same variable at the same time, it can produce unexpected results.
- Code block to be executed by only one process at a time is referred as Critical section. If multiple processes execute the same code concurrently it may produce undesired results.
- To resolve race condition problem, one process can access resource at a time. This can be done using sync objects/primitives given by OS.
- OS Synchronization objects are:
    - Semaphore, Mutex

Semaphore

- Semaphore is a sync primitive given by OS.
- Internally semaphore is a counter. On semaphore two operations are supported:
    - wait operation: dec op: P operation:
        - semaphore count is decremented by 1.
        - if cnt < 0, then calling process is blocked.
        - typically wait operation is performed before accessing the resource.
    - signal operation: inc op: V operation:
        - semaphore count is incremented by 1.
        - if one or more processes are blocked on the semaphore, then one of the process will be resumed.
        - typically signal operation is performed after releasing the resource.
- Semaphore types
    - Counting Semaphore
        - Allow "n" number of processes to access resource at a time.
        - Or allow "n" resources to be allocated to the process.
    - Binary Semaphore

        - Allows only 1 process to access resource at a time or used as a flag/condition.

Mutex

- Mutex is used to ensure that only one process can access the resource at a time.
- Functionally it is same as "binary semaphore".
- Mutex can be unlocked by the same process/thread, which had locked it.

Semaphore vs Mutex

- S: Semaphore can be decremented by one process and incremented by same or another process.
- M: The process locking the mutex is owner of it. Only owner can unlock that mutex.
- S: Semaphore can be counting or binary.
- M: Mutex is like binary semaphore. Only two states: locked and unlocked.
- S: Semaphore can be used for counting, mututal exclusion or as a flag.
- M: Mutex can be used only for mutual exclusion.

Deadlock

- Deadlock occurs when four conditions/characteristics hold true at the same time.
    - No preemption: A resource should not be released until task is completed.
    - Mutual exclusion: Resources is not sharable.
    - Hold & Wait: Process holds a resource and wait for another resource.
    - Circular wait: Process P1 holds a resource needed for P2, P2 holds a resource needed for P3 and P3 holds a resource needed for P1.

Deadlock Prevention

- OS syscalls are designed so that at least one deadlock condition does not hold true.
- In UNIX multiple semaphore operations can be done at the same time.

Deadlock Avoidance

- Processes declare the required resources in advanced, based on which OS decides whether resource should be given to the process or not.
- Algorithms used for this are:
    - Resource allocation graph: OS maintains graph of resources and processes. A cycle in graph indicate circular wait will occur. In this case OS can deny a resource to a process.
    - Banker's algorithm: A bank always manage its cash so that they can satisfy all customers.
    - Safe state algorithm: OS maintains statistics of number of resources and number processes. Based on stats it decides whether giving resource to a process is safe or not (using a formula):
        - Max num of resources required < Num of resources + Num of processes
            - If condition is true, deadlock will never occur.
            - If condition is false, deadlock may occur

Deadlock recovery

- it is done by one of following method
    - Resource pre emption

    - process termination

Computer structure

- CPU: Genral purpose processor for program/OS execution
- Memory: RAM
- Storage: Disk
- IO: Keyboard, Monitor
- Connected by "bus".
- Each IO device has a "dedicated" "internal" processing unit -- IO device controller.

Computer IO (Input Output)

- Synchronous IO: CPU is waiting for IO to complete.
    - Hw technique: Polling
- Asynchronous IO: CPU is not waiting for IO to complete (doing some other task)
    - Hw technique: Interrupts
    - OS maintains a device status table to keep track of IO devices (busy/idle) and processes waiting for those IO devices.

Interrupt Processing

- IO event is sensed by IO device controllers.
- It will be conveyed to CPU as a special signal - Interrupt.
- CPU pause current execution and execute interrupt handler.
- "Interrupt handler" will get address of "ISR" (from IVT) and execute ISR.
- When ISR is completed, execution resumes where it was paused.

Hardware vs Software interrupt

- Hardware -- interrupts from hardware peripherals.
- Software interrupt
    - Special instructions (Assembly/Machine level) when executed, current execution is paused, interrupt handler is executed and then the paused execution resumes.
    - Arch specific:
        - 8085/86: INT
        - ARM 7: SWI
        - ARM Cortex: SVC
    - Also called as "Trap" in few architecture.

Interrupt Controller

- Convey the interrupts from various peripherals to the CPU.
- Also manage priority of the interrupt (when multiple interrupts arrives at same time).
- e.g. 8085/86 <-- 8259, Modern x86 processors (apic), ARM-7 (VIC), ARM-CM3 (NVIC), ...

System Calls

- Software interrupt is used to implement OS/Kernel services.

- Functions exposed by the kernel so that user programs can access kernel functionalities, are called as "System calls".
  - e.g. Process Mgmt: create process, exit process, communication, synchronization, etc.
  - e.g. File Mgmt: create file, write file, read file, close file, etc.
  - e.g. Memory Mgmt: alloc memory, release memory, etc.
  - e.g. CPU Scheduling: Change process priroty, change process CPU affinity, etc.
- System calls are specific to the OS:
  - UNIX: 64 syscalls e.g. fork(), ..
  - Linux: 300+ syscalls e.g. fork(), clone(), ...
  - Windows: 3000+ syscalls e.g. CreateProcess(), ...

Redirection

- By default every process opens 3 file descriptors
  - fd = 0 -> standard input to a program
  - fd = 1 -> standard output to a program
  - fd = 2 -> standard error to a program
- You can redirect each of these independently.
- According to direction of redirection, there are three types
- Input redirection
  - “<” is used for input redirection
- Output redirection
  - “>” is used for input redirection
- Error redirection
  - “2>” is used for input redirection

Pipe

- Using pipe, we can redirect output of any command to the input of any other command.
- Two processes are connected using pipe operator (|).
- Two processes runs simultaneously and are automatically rescheduled as data flows between them.
- If you don’t use pipes, you must use several steps to do single task.
- E.g.
  - who | wc

Regular Expressions

- Find a pattern in text file(s).
- Regular expressions are patterns used to match character combinations in strings.
- A regular expression pattern is composed of simple characters, or a combination of simple and special characters e.g. /abc/, /ab*c/

grep

- Pattern is given using regex wild-card characters.
    - Basic wild-card characters
        - $ - find at the end of line.
        - ^ - find at the start of line.
        - [ ] - any single char in give range or set of chars
        - [^] - any single char not in give range or set of chars
        - . - any single character
        - - - zero or more occurrences of previous character
    - Extended wild-card characters
        - ? - zero or one occurrence of previous character
        - - - one or more occurrences of previous character
        - {n} - n occurrences of previous character
        - {,n} - max n occurrences of previous character
        - {m,} - min m occurrences of previous character
        - {m,n} - min m and max n occurrences of previous character
        - () - grouping (chars)
        - (|) - find one of the group of characters
- Regex commands
    - grep - GNU Regular Expression Parser - Basic wild-card
    - egrep - Extended Grep - Basic + Extended wild-card
    - fgrep - Fixed Grep - No wild-card
- Command syntax
    - grep "pattern" filepath
    - grep [options] "pattern" filepath
        - -c : count number of occurrences
        - -v : invert the find output
        - -i : case insensitive search
        - -w : search whole words only
        - -R : search recursively in a directory
        - -n : show line number.