[**Spring (April) 2019**](#_bvnbq0mznr4u) **2**

[Problem #1](#_w6nffn4vyrtk) 2

[Problem #2](#_4uatz9u9trag) 4

[Problem #3](#_j8vbzab1j5e8) 5

[**Fall (November 2018**](#_gq5piv1czosq) **8**

[Problem #1](#_pjy2mxxyrn8y) 8

[Problem #2](#_yqf5c49qh27k) 8

[Problem #3](#_ju61asq4x2qh) 10

[**Spring (April) 2018**](#_yi9jz9m17w0o) **12**

[Problem #1](#_4hy7wlowtlbq) 12

[Problem #2](#_yudesb94h8sx) 14

[Problem #3](#_gct8l9fcrwxz) 14

[**Fall (November) 2017**](#_72qf3gwmbcvr) **16**

[Problem #1](#_sylqg9eaklf) 16

[Problem #2](#_8kugb6no4u1z) 17

[Problem #3](#_kqtcajbvrnoz) 17

[**Spring (April) 2017**](#_5hkusg2b0w5n) **21**

[Problem #1](#_i77o3xwf6vta) 21

[Problem #2](#_8q9f8pvyttlp) 21

[Problem #3](#_fv7ghrksmene) 22

[**Fall (November) 2016**](#_d6qblc6w2omx) **24**

[Problem #1](#_i0pso1ifsn6x) 24

[Problem #2](#_f6sp06hcog4j) 24

[Problem #3](#_v7w53tc9sdsz) 26

[**Spring (April) 2016**](#_2hui7gy8q74u) **28**

[Problem #1](#_ptisukmppibc) 28

[Problem #2](#_4gb577ybgzt3) 30

[Problem #3](#_xl7gb7xwv312) 30

[**Fall (November) 2012**](#_gn8ria5hqjc) **31**

[Problem #1](#_7d1ojxndqanv) 31

[Problem #2](#_h0nr2rhmjvp7) 32

[Problem #3](#_xfac1tiokgq3) 32

[**Notes**](#_vw43tpilekj1) **33**

# Spring (April) 2019

## Problem #1

**Explain what process CPU burst and what is CPU-I/O burst cycle. Describe the basic scheduling disciplines: Shortest Job First (SJF), Shortest Remaining Time (SRT), Round-Robin (RR), First Come First Served (FCFS).**

Processes alternate between two states: (1) CPU bursts, and I/O bursts

**CPU burst** is the amount of time the process uses the processor, a burst of performing calculations.

**I/O burst** refers to the state of waiting for data transfer in or out of the system. The process is in the running state, it request for I/O, goes into the blocking or wait state. Then when completed the process is put back into the ready state.

**CPU - I/O burst cycle** refers to the overall alternation between a CPU burst, and an I/O burst - when the process is waiting.

<https://www.quora.com/What-is-meant-by-CPU-Burst-and-I-O-Burst>

**Shortest Job First (SJF)**:

* Selects from the queue, a process with the shortest execution time
* We can think of this as a Greedy Algorithm
* It is not preemptive
* May cause starvation if shorter processes keep coming. This can be solved using the concept of “aging”
* It is practically infeasible as the OS may not know the burst time ahead of time and therefore may not be able to sort them appropriately.

**Shortest Remaining Time (SRTF)**:

* Preemptive version of SJF
* Select process with the shortest amount of time remaining until completion
* Because the currently running process is the one with the shortest amount of time remaining by definition, and since that time should only reduce as execution progresses
  + Processes will always run until they complete or until a new process is added that requires a smaller amount of time to complete
* Like SJF it has the potential to cause starvation. Long processes may be held off immediately if short processes are continuously added

**Round-Robin (RR)**:

* Each process is assigned a fixed time slot in a cyclic way
* Simple to implement, and starvation-free as all processes get fair share of CPU
* One of the most commonly used techniques in CPU scheduling
* It is preemptive as processes are assigned CPU only for a fixed slice of time at most
* It is more overhead because of context switching

**First Come First Served (FCFS)**:

* Queues processes in the order that they arrive in the ready queue
* In this algorithm, the process that comes first will be executed first and the next process only starts after the previous process get fully executed
* We consider that the arrival time for all processes is 0

**Assume the processes become runnable as follows:**

|  |  |  |
| --- | --- | --- |
|  | Arrival time, ms | Burst time, ms |
| P1 | 0 | 10 |
| P2 | 1 | 5 |
| P3 | 3 | 3 |

**Assume the time slice for RR is 1 ms. Drawn Gantt charts of scheduling the processes by the four scheduling disciplines. Explain your answer.**

**RR Gant**

Note that in Round Robin, when a process hasn’t finished within the given time slot, the process’ arrival time is reset to the time it was preempted.

Correct Answer

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |  |  |  | 17-18 |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

<https://www.gatevidyalay.com/round-robin-round-robin-scheduling-examples/>

Wrong Answer

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| P1 | P2 | P3 | P1 | P2 | P3 | P1 | P2 | P3 | P1 | P2 | P1 | P2 | P1 | | | | |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |  |  |  | 17 |

**SJF Gantt**

|  |  |  |
| --- | --- | --- |
| P1 | P2 | P3 |
| 0 - 10 | 11 - 15 | 16 - 19 |

**SRTF (Preemptive SJF) Gantt**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| P1 | P2 | P3 | P2 | P1 |
| 0 - 1 | 1 - 3 | 3 - 6 | 6 - 9 | 9 - 18 |

**FCFS Gantt**

|  |  |  |
| --- | --- | --- |
| P1 | P2 | P3 |
| 0 - 10 | 10 - 15 | 15 - 18 |

## Problem #2

**- Briefly explain page-based virtual memory organization.**

Page-based virtual memory organization is a memory management scheme that utilizes the concept of Pages - a unit of logical memory of a program, which are indexed on a Page Table - a mapping between the virtual addresses and the physical addresses.

* All pages are of the same size
* All frames - a unit of physical memory

Paged Memory Management gives rise to the notion of demand paging using virtual memory. While not in use, the pages are stored on disk - **secondary storage** (disk).

* The page table indicates whether the page is in **main memory** (RAM or on the secondary storage (disk)

If a process requests a page that is not in memory, a page fault trap is generated and control is passed to the OS. The faulted process is suspended (another process may be started while it waits) and a request to fetch the page is generated. When the page is in memory, the page table is updated and the instruction that caused the page fault is re-executed.

<http://faculty.salina.k-state.edu/tim/ossg/Memory/virt_mem/virt_mem.html>

<http://faculty.salina.k-state.edu/tim/ossg/Memory/paged_mem.html#paged-mem>

**- Introduce the concept of page replacement algorithm and explain why it is necessary.**

Whenever a process refers to a page that is not present in main memory (RAM), a page fault occurs. Subsequently, the OS replaces one of the existing pages with the referred page. Page replacement algorithms are an important part of virtual memory management because it helps the OS decide which memory page can be moved out to make space for the requested page.

<https://study.com/academy/lesson/page-replacement-definition-algorithms.html>

**- Define thrashing and explain it with respect to page replacement.**

Thrashing to the critical point when the number of page faults goes up dramatically, and the time spent resolving them overwhelms the time sent on the program’s computation.

**- Explain the concept of working set and how keeping track of it may prevent or mitigate thrashing.**

The pages most frequently accessed are called the **working set**; It is a conceptual model aimed at preventing thrashing. The idea behind it is that we want to figure out how much memory each process needs, which must be present in main memory, to prevent thrashing.

Its implementation is as follows:

1. Choose time *T*
2. Pages that were accessed during time *T* constitute a working set, the rest can be disregarded
3. Scan periodically to update working set

<https://web.stanford.edu/~ouster/cgi-bin/cs140-spring14/lecture.php?topic=thrashing>

## Problem #3

**Explain why operating system schedules disk read-write head movement. Consider a 1000-cylinder disk and the following queue of requests:**

**40, 180, 400, 900, 950**

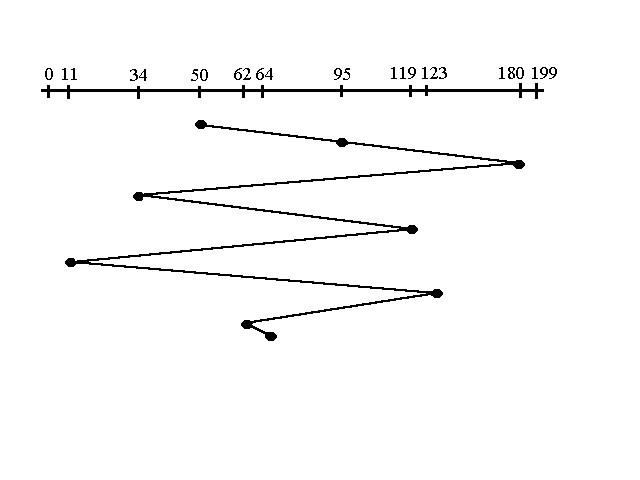
**Assume that the read-write head is presently at the cylinder number 500 going from inner (lower numbered) to outer (higher numbered) cylinders. Describe shortest-seek-time-first (SSTF), LOOK, and C-LOOK would schedule the read-write head movement. Explain your answer. Out of these three scheduling algorithms, which one would most likely be implemented in practice? Explain.**

The operating systems disk read-write head movement to attempt to most efficiently access the proper disk position / optimize seek time. Since moving read-write heads is slow we can use selection algorithms such as FCFS, SSTF, SCAN, C-SCAN, LOOK or C-LOOK.

**FCFS**: All incoming requests are put in a queue. Whichever number is next in the queue will be the next number served. Not optimal algorithm as this algorithm implies that in the worsts case we can go from the lower-end number of the cylinder to the highest.

**40, 180, 400, 900, 950**

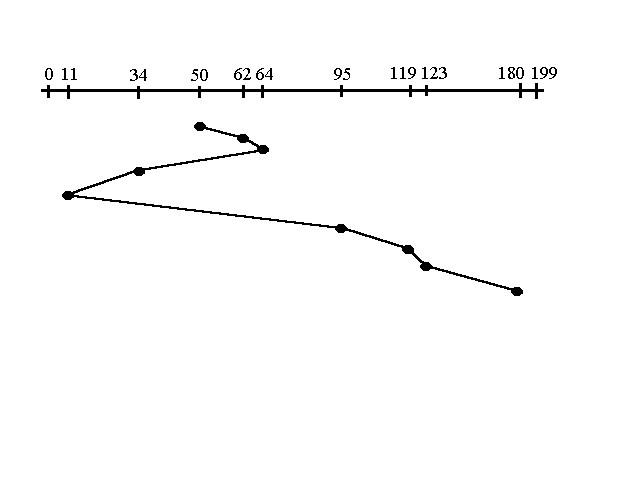
Example chart: <http://www.cs.iit.edu/~cs561/cs450/disksched/disksched.html>



**SSTF**: Requests are serviced according to the next shortest distance from the current cylinder number. May cause starvation for requests with far away distance, if we consider the scenario where incoming requests are always the shortest distance away from the current cylinder number.

**400, 180, 40, 900, 950**

Example chart: <http://www.cs.iit.edu/~cs561/cs450/disksched/disksched.html>

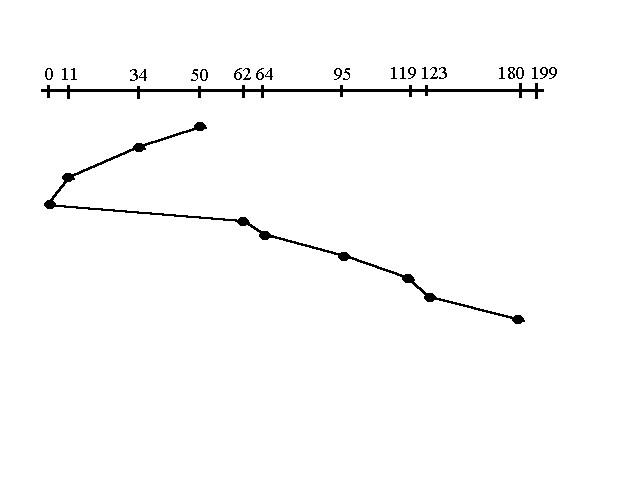


**SCAN**: Elevator scan. Starts from current disk position scans down towards the nearest end. When it hits bottom it scans ups servicing requests that it didn’t hit on the way down. As a result, requests arriving at midrange will be serviced more than those arriving behind the disk arm.

**400, 180, …, 0, …, 40, 900, 950, …, 1000**

<https://www.geeksforgeeks.org/scan-elevator-disk-scheduling-algorithms/>

Example chart: <http://www.cs.iit.edu/~cs561/cs450/disksched/disksched.html>

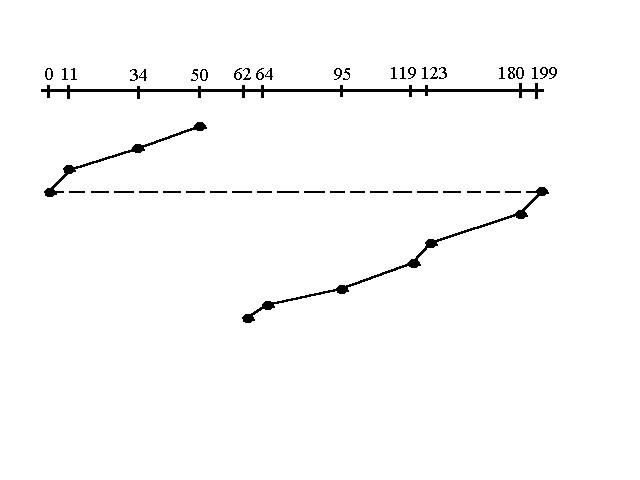


**C-SCAN**: Also known as circular elevator. It is similar to C-SCAN except that it treats the cylinders as a circular list that wraps around from the last cylinder to the first one. Aimed at servicing requests more uniformly. We scan down (or up) from current cylinder, when we hit the bottom (or top) we scan from the down from other end. Keep in mind that the huge jump doesn’t count as head movement.

**400, 180, 40, …, 0, 1000, …, 950, 900**

<https://www.geeksforgeeks.org/c-scan-disk-scheduling-algorithm/>

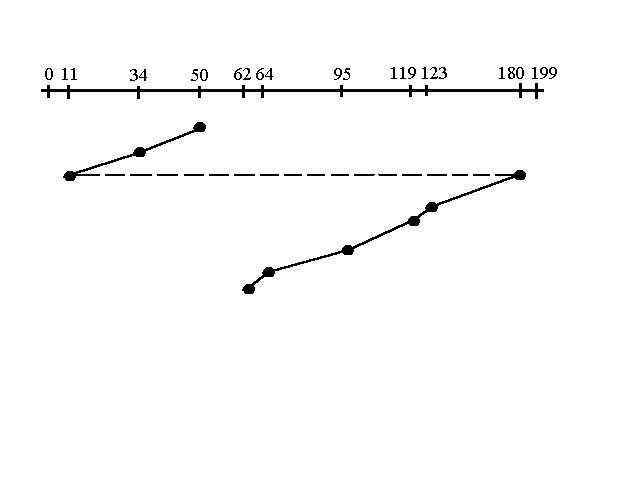
Example chart: <http://www.cs.iit.edu/~cs561/cs450/disksched/disksched.html>



**C-LOOK**: This is just an enhanced version of C-SCAN except that it doesn’t go past the last request in the direction that the head is moving. It too jumps to the other end except not all the way to the end, it just jumts to the farthest request.

**400, 180, 40, 950, 900**

Example chart: <http://www.cs.iit.edu/~cs561/cs450/disksched/disksched.html>



<http://www.cs.iit.edu/~cs561/cs450/disksched/disksched.html>

<https://cs.gmu.edu/~huangyih/471/disk.pdf>

# 

# Fall (November 2018

## Problem #1

**Define page-based memory allocation.**

Page-based memory allocation is a memory management scheme which uses the concept of Pages - a fixed unit of logical memory, which maps to physical memory through the use of a Page Table.

**Differentiate between logical and physical memory address.**

Physical memory refers to the actual device memory.

Logical memory refers to the translation to a physical device memory. Logical addresses are virtual addresses as it does not exist physically.

Virtual memory refers to a simulation of a physical device.

<https://techdifferences.com/difference-between-logical-and-physical-address.html>

**Explain translation look-aside buffer (TLB) and how it is used for memory access.**

Translation lookaside buffer (TLB) is a memory cache that is used to reduce the time taken to access a user memory location. It is part of the chip’s memory-management unit (MMU)

The TLB stores the recent translation of virtual memory to physical memory and can be called an address-translation cache.

**Differentiate TLB hit from TLB miss.**

TLB hit refers to a TLB search that results in the requested address being present in the TLB.

TLB miss refers to a TLB search that results in a requested address not being present in the TLB.

## Problem #2

**Describe a process control block (PCB) and how it is used.**

As the OS supports multiprogramming, it needs to keep track of all the processes.

For this, the process control block (PCB) is used to track a process’s execution status and all the information needed to manage the scheduling of a particular process.

A process control block (PCB) contains information about the process, i.e. registers, quantum, priority, etc.

* The process table is an array of PCB’s, that means logically contains a PCB for all of the current processes in the system.

**List typical contents of a PCB.**

* **Process state:** The state may be (a) new, (b) running, (c) waiting, (d) halted, and so one
* **Program counter:** The counter indicates the address of the next instruction to be executed for this process
* **CPU register:** The registers may vary in number and type, depending on the computer architecture.
  + They include accumulators, index, registers, stack pointers, and general-purpose register, plus any condition-code information. Along with the program counter, this state information must be saved when an interrupt occurs, to allow the process
* **CPU-scheduling information:** The information includes a process priority, pointers to scheduling queues, and any other scheduling parameters
* **Memory-management information:** This information may include information such as the value of the base and limit registers, the page tables, or the segment tables, depending on the memory used by the OS
* **Accounting information:** This information includes the amount of CPU an real-time used, time limits, account numbers, job or process numbers, and so on
* **I/O status information:** This information includes the list of I/O devices allocated to the process, a list of open files, and so on

<https://en.wikipedia.org/wiki/Process_control_block>

<https://www.tutorialspoint.com/what-is-process-control-block-pcb>

**Describe what process queue is and how PCBs are related to process queues.**

The OS maintains all Process Control Blocks in Process Scheduling Queues. The OS maintains a separate queue for each of the process states and PCBs of all processes in the same execution status are placed in the same queue.

<https://www.tutorialspoint.com/operating_system/os_process_scheduling.htm>

**Differentiate device (I/O) queue and ready queue. Explain how both types of queues are used by the process scheduler.**

* **Ready queue** - This queue keeps a set of all processes residing in main memory, ready and waiting to execute. A new process is always put in this queue. Generally stored as a linked list.
* **I/O queue** - I/O device query: processes in state waiting, blocked on particular device waiting for their request to be submitted to the device controller. An I/O device queue is one type of wait queue. Each device has its own I/O queue

<https://www.tutorialspoint.com/operating_system/os_process_scheduling.htm>

<https://www.quora.com/What-is-difference-between-Ready-queue-and-Job-queue-in-operating-system>

## Problem #3

**(a) Describe the need for synchronization in OS design.**

**(b) Define semaphores and two operations that are available on each semaphore.**

**(c) Define the bounded buffer (producer-consumer) problem. Given the following code for producer:**

|  |
| --- |
| #1 while (true){  #2 // produce item for the buffer  #3 wait(empty);  #4 wait(mutex)  #5 // add item to the buffer  #6 signal(mutex)  #7 signal(full);  #8 } |

**d) Reconstruct the data structures (semaphores and their initialization) and the code for consumer. Explain the operation of your code.**

a) The need for synchronization originates when processes need to execute concurrently. The main purpose of synchronization is the sharing of resources without interference using mutual exclusion.The other purpose is the coordination of the process interactions in an operating system.

<https://link.springer.com/chapter/10.1007/0-306-46976-6_7>

b) Semaphores are a type of synchronization primitive used to control access to a common shared resource by multiple processes. Apart from its initialization, can be accessed by its two atomic operations (1) P / **wait** and (2) V / **signal**

**P/Wait** decrement the semaphore’s counter by 1. If after the decrement the counter is negative, the process executing “wait” is blocked and placed in the semaphore’s wait queue. Otherwise, the process continues execution, having used a unit of the resource.

**V/Signal** increments the semaphore’s counter by 1. After the increment, if the value is still negative or equal to zero (meaning there are processes waiting for the resource), it wakes u a blocked process from the semaphore’s queue.

<https://sites.cs.ucsb.edu/~rich/class/cs170/notes/Semaphores/>

c) The producer-consumer problem (also known as the bounded-buffer problem), and the readers-writers problem are two classical case studies considered to describe and test synchronization mechanisms. These two problems are implemented in various simulation models, each with a different solution or different workload parameters to study the process synchronization mechanisms that can be used with interacting processes.

There is a fixed size buffer.

The producer produces items and adds them into the buffer.

The consumer removes items from the buffer and consumes them.

A producer should not produce items into the buffer when the consumer is consuming an item, and vice-versa. So in essence access to the buffer should be mutually exclusive.

<https://www.tutorialspoint.com/producer-consumer-problem-using-semaphores>

LOOK AT THIS ONE <https://www.youtube.com/watch?v=kPMrsQfE2_8>

d) **In the producer code** - mutex, empty and full are semaphores.

mutex is initialized to 1,

empty is initialized to n (maximum size of the buffer) - it denotes the “empty“ space available, and

full is initialized to 0 - denotes how “full“ the buffer is

**In the consumer code below**

The wait on full, indicates that the items in the buffer have decreased by 1.

The wait on mutex prevents the producer process from interfering

The item is removed from the buffer

Then we signal on mutex to let the producer know it can act on the buffer.

The signal on empty indicates that we have increased the number of empty slots in the buffer

Consumer code

|  |
| --- |
| #1 while (true) {  #2 wait(full);  #3 wait(mutex);  #4 // remove item from buffer  #5 signal(mutex);  #6 signal(empty);  #7 // consume item  #8 } |

<https://www.tutorialspoint.com/producer-consumer-problem-using-semaphores>

# 

# Spring (April) 2018

## Problem #1

**Describe the concept of a process.**

A process is basically a program in execution. The execution of a process must progress in a sequential fashion.

**Explain what process control block (PCB) is and how it is used.**

**List the contents of a PCB.**

It is a data structure that stores the state of a process and used during context switching.

* **Process state**: new, ready, waiting, etc
* Process number
* **Program counter**: indicates the address of the next instruction to be executed
* **CPU registers**: Depends on the architecture. Includes accumulators, index registers, stack pointers, general-purpose registers
* **CPU-scheduling information**: Includes process priority, pointer to scheduling queues, and other scheduling parameters
* **Memory-management information**: Include information such as the value of the base and limit registers
* **Accounting information**: This information includes amount of CPU and real-time used
* **I/O status information**: Includes the list of I/O devices allocated to the process, a list of open files and so on

**Explain what scheduling queues are, specifically ready queue and device I/O queue.**

Ready queue: processes in the ready state waiting to be ran by the CPU

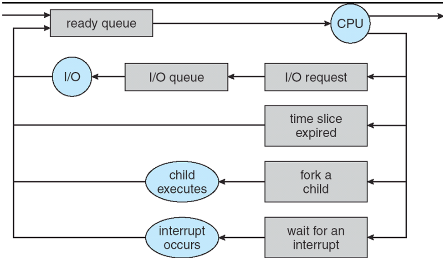
Device I/O queue: process in the waiting state, blocked on a particular device waiting for their request to be submitted to the device controller. The device I/O queue is one type of wait queue.

**Explain how and why a PCB of a process migrates from one of these queues to another.**

A process starts in the ready queue, then when it is scheduled to run the dispatcher allocates it to the cpu - in running state. When there’s an I/O request a process is placed in the wait in particular the Device I/O queue. Once I/O completes the process is placed back in the ready queue to be scheduled to run.

https://cseweb.ucsd.edu/classes/fa06/cse120/lectures/120-fa06-l3.pdf

<https://www.cs.rutgers.edu/~pxk/416/notes/07-scheduling.html>



**Describe the process creation (either Unix of Windows).**

Standard steps

* A process is created by another process, through a “process creation system call”.
  + In the case of Unix this is through a call to fork( )
* Child - the new process
  + An almost identical coy of the parent (same code, same data, etc)
* Parent - the original process
  + Can either wait for the child to complete or continue executing in parallel (concurrently) with the child
* Fork returns twice
  + Returns the child’’s PID to the parent,
  + Return “0” to the child
* The child often use exec( ) to start another completely different program

**In case of Unix, explain the operation of fork() system call. In case of Windows, explain the operation of CreateProcess() system call.**

Unix fork( ) creates a process

* Creates and initializes a new PCB
* Creates a new address space
* Initializes the address space with copy of the contents of the address space of the parent (text, data, & stack into new address space of the parent)
* Initialize the Kernel resources to point the resource used by the parent. (Provides child with access to open files)
* Places the PCB on the ready queue

## Problem #2

**Describe the operation of a CPU scheduler (dispatcher).**

When various processes are residing in the READY Queue, waiting to be executed by the CPU, it is the Short-term scheduler’s job to select a particular process on the basis of a given scheduling algorithm.

Once this process is selected, it is the Dispatcher’s job to take the process from the READY state to the RUNNING state. It is the dispatcher that gives a process control over the CPU after it has been selected by the short-term scheduler. Therefore, it is the CPU Scheduler that gives the Dispatcher an ordered list of processed to move to the CPU over time.

<https://www.geeksforgeeks.org/difference-between-dispatcher-and-scheduler/>

**Define CPU burst time of a process.**

It is the total time spent a process spends running on the CPU after it is dispatched from the ready queue. It’s the period of computation between I/O requests that is called the CPU burst.

**Explain how operating system determines the CPU burst time of a process.**

The burst time is an estimate based on an initial starting default burst value and actual historical run values. The last estimated burst value and the last actual burst value are averaged to provide the process’ next burst.

<https://cs.stackexchange.com/questions/50438/how-does-the-os-determine-the-cpu-burst-time-of-a-process>

**Describe Shortest Job First (SJF) and Shortest Remaining Time (SRT) scheduling algorithms and explain how these two algorithms use process burst time.**

<https://www.geeksforgeeks.org/shortest-job-first-cpu-scheduling-with-predicted-burst-time/>

**Give an example of SRT and SJF scheduling.**

## Problem #3

**Explain why the OS schedules disk head movement.**

To optimize seek time.

**Explain why First Come First Served (FCFS) disk head scheduling algorithm is inadequate and suggest an alternative that is more suitable.**

FCFS disk head scheduling algorithm is inadequate because it causes the disk head to oscillate across a large number of cylinders to serve requests that are far apart. We can use an algorithm like C-SCAN or C-LOOK that provides better performance

**Give an example of head scheduling that would demonstrate the advantage of your suggested algorithm over FCFS.**

Example of using C-LOOK vs FCFS is shown below.

Consider the following:

Current head: 500

Request Queue: 40, 180, 400, 900, 950

FCFS

(500 - 40) = 460

(40 - 180) = 140

(180 - 400) = 220

(400 - 900) = 500

(900 - 950) = 400

Total = 1720 tracks gone through

C-LOOK Algorithm

Similar to C-SCAN except we only scan down up to the lowest request, than circle around only to the largest requests. The wrap around does not cost a track movement.

400, 180, 40, 950, 900

(500 - 400) = 100 tracks gone through

(400 - 180) = 220 tracks gone through

(180 - 40) = 140 tracks gone through

(950 - 900) = 50 tracks gone through

Total = 510 tracks gone through

# 

# Fall (November) 2017

## Problem #1

**Define a process, explain what “process state” is.**

A process, also called a task or job, is a program in execution.

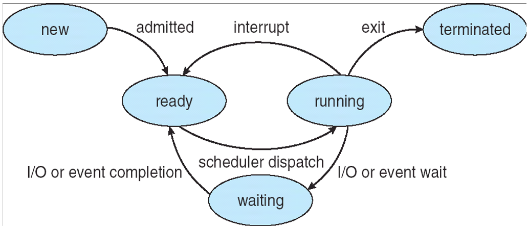
It contains two parts: (1) program code, (2) sequence of instructions to be executed, another of which is context (execution state).

The process state indicates the status or condition of a process.

**A process may be in one of the 5 states. Name and describe them.**

1. New: The process is being created
2. Running: Instructions are being executed
3. Waiting: The process is waiting for some event to occur, such as I/O completion or receiving a signal
4. Ready: The process is waiting to be assigned to a processor
5. Terminated: The processor has finished execution

**Describe transitions between these states.**



**Explain what context switch is.**

It is the process of storing the state of a computer process, so that it can be restored and its execution resumed from the same point later.

Stopping one process and restarting another

Save the state of that process, update the Process Control Block of both processes, restore state of the next process.

**Describe what state transition involves context switch and why.**

Running -> Ready // Kernel executes (system call, exception, interrupt)

Running -> Waiting // I/O or event

Ready -> Running // Which process to schedule

// Look at slide 56

<https://www.cs.swarthmore.edu/~kwebb/cs45/s18/03-Process_Context_Switching_and_Scheduling.pdf>

## Problem #2

**Explain why operating system design needs the concept of synchronization primitives, such as semaphores.**

The need for synchronization arises when we need processes to run concurrently. The main purpose of synchronization is to allow resources to be shared without interference using the concept of mutual exclusion - only one thread is allowed to access the shared resources or allowed to enter the critical section. Critical section refers to the part of the program that must be protected.

A semaphore is a variable or ~~abstract data type~~ (integer type) used to control access to a common resource by multiple processes in a concurrent system such as a multitasking operating system.

In Multithreading or multiprocessing resources are shared. If multiple threads are allowed to access such resources, this may lead to a race condition. To prevent such race conditions, a thread’s access to a shared resource should be mutually exclusive.

In essence, synchronization primitives allows for concurrency in operating systems.

**Explain the operation of the two semaphore operations: wait() and signal().**

Wait (P) decrements the value of semaphore variable by 1. If the new value of the semaphore is negative, the process executing wait is blocked (i.e. added to the semaphore’’s queue). Otherwise, the process continues execution, having used a unit of the resource.

**V/Signal** increments the semaphore’s counter by 1. After the increment, if the value is still negative or equal to zero (meaning there are processes waiting for the resource), it wakes up a blocked process from the semaphore’s queue. If it is 1, it will signal the critical section is free and leave.

**Define the critical section (mutual exclusion) problem.**

A critical section is a code segment that accesses shared variables and has to be executed as an atomic action. The critical section (mutual exclusion) problem refers to how to ensure that only one process must be executing this critical section at a time.

If any other process also wants to execute the critical section, it must wait until the first one finishes.

**Provide a semaphore-based solution (write code) to the critical section problem.**

**--------------------------------------------------------------------**

**t1 () {**

**while (true) {**

**wait(s);**

**/\* CS \*/**

**signal(s);**

**/\* non-CS \*/**

**}**

**}**

**t2 () {**

**while (true) {**

**wait(s);**

**/\* CS \*/**

**signal(s);**

**/\* non-CS \*/**

**}**

**}**

**--------------------------------------------------------------------------**

## Problem #3

**Introduce the concept of RAID as a mechanism to use hard disk space redundancy to counteract failure.**

RAID - redundant array of independent disks, is a way of storing the same data across different places on multiple hard disks to protect data in case of a drive failure.k

<https://searchstorage.techtarget.com/definition/RAID>

**Describe striping and mirroring as techniques.**

**Striping** is a way of writing data to member disks in which the data flow is split onto the blocks of a certain size and then written to the disks in turn.

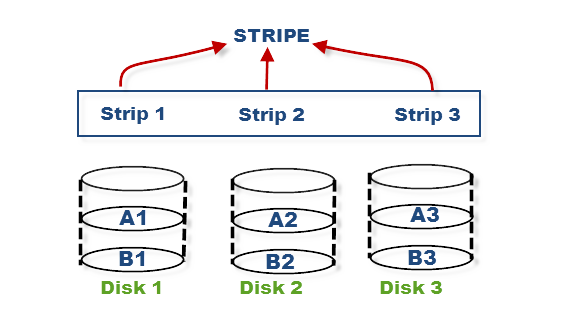
**Mirroring** - stores identical data copies on the array member disks.

**Parity** is a type of data organization where splitting data onto the blocks (striping) is used along with the calculation of a certain checksum which then is written to the member disks.

<http://www.raid-recovery-guide.com/raid-methods.aspx>

Striping and mirroring are two of the basic of RAID levels.

For me, Striping is the most confusing RAID level as a beginner and needs a good understanding and explanation. We all know that, RAID is a collection of multiple disk’s and in these disk predefined number of contiguously addressable disk blocks are defined which are called as strips and collection of such strips in aligned in multiple disk is called stripe.



Suppose you have hard disk, which is a collection of multiple addressable block and these blocks are stacked together and called strip and you have multiple such hard disk, which are place parallel or serially. Then such combination of disk is called stripe.

<https://www.storagetutorials.com/understanding-concept-striping-mirroring-parity/>

**Explain what parity is and how it is used.**

As explained above, mirroring involves high cost, so to protect the data new technique used with striping is called parity. This is reliable and low cost solution for data protection. In this method and additional HDD or disk is added to the stripe width to hold parity bit.

**Differentiate RAID levels 4 and 5. Explain their relative advantages and disadvantages.**

RAID allows you to have a set of physical disks provide itself as one single disk through an abstraction.

RAID 0 -- Uses a form of striping. There’s different forms of striping (bit, byte and block striping. What it means is that we are splitting the data across the disks. We get much faster writes.

But, if we lose a disk we are fucked.

RAID 1 -- Uses mirroring technique to replicate the data across other disks. Increases fault tolerance and reliability.

RAID 2 -- Requires a minimum of 3 disks. Data is striped at the bit level (not block. Each bit is written on a different drive/stripe. It requires the use of Hamming Code for error correction stored in extra disks. But, when it needs to read it will need to read the corresponding Hamming Code.

RAID 3 -- Works as RAID 0 does. It uses byte-level striping - but it also uses on additional disk in the array to store checksums - the parity disk. If disk 0 dies then it can reconstruct by looking at the parity. Good for sequential reads and writes. But, random read and writes are worse performing.

RAID 4 -- Striped at a block level and has a dedicated parity disk. Good random reads, bad random writes because it writes that parity disk. If you destroy more than 1 disk everything is fucked.

RAID 5 -- Striped at block level and distributed parity. You lose more space in each disk, but can distribute the parity across all disk. Good for databases. If you destroy more than 2 disks.

RAID 6 -- Extends RAID 5, but adds another parity block. Essentially you’ll have two parity disks distributed across the disks. Configuration is complex.

https://www.youtube.com/watch?v=wTcxRObq738

======================

* **RAID 0** - based on striping.
  + This RAID level doesn't provide fault tolerance but increases the system performance (high read and write speed).
* **RAID 1**
  + utilizes mirroring technique, increases read speed in some cases, and provides fault tolerance in the loss of no more than one member disk.
* **RAID 4**
  + RAID 4 stripes the data across multiple disks just like RAID 0. In addition to that, it also stores parity information of all the disks in a separate dedicated disk to achieve redundancy. In the diagram below, Disk 4 serves as the parity disk having parity blocks Ap, Bp, Cp and Dp. So, if one of the disks fails, the data can be reconstructed using the parity information of that disk. Space is more efficiently used here when compared to RAID 1 since parity information uses way less space than mirroring the disk.
  + The write performance becomes slow because all the parity information is written on a single disk which is a bottleneck.
  + This problem is solved in RAID 5 as we will see next.
* **RAID 5**
  + RAID 5 is very similar to RAID 4, but here the parity information is distributed over all the disks instead of storing them in a dedicated disk. This has two benefits — First, there is no more a bottleneck as the parity stress evens out by using all the disks to store parity information and second, there is no possibility of losing data redundancy since one disk does not store all the parity information.

<https://www.booleanworld.com/raid-levels-explained/>

<http://www.freeraidrecovery.com/library/what-is-raid.aspx>

# Spring (April) 2017

## Problem #1

**Define multitasking. Define and differentiate cooperative and preemptive multitasking.**

Multitasking refers to an OS running multiple independent computer programs / processes on the same computer, giving the appearance that it is performing the tasks at the same time, while such tasks share common processing resources such as central processing units (CPUs) and main memory.

In cooperative multitasking, applications voluntarily yield CPU time to one another.

In preemptive multitasking, the operating system uses a criteria to decide how long to allocate a CPU to one task before giving another task a turn to use the CPU.

**Explain why preemptive multitasking requires hardware support.**

**Describe the kind of hardware support that is required.**

It requires a hardware timer that interrupts at periodic intervals.

**Outline the sequence of events that happen when one disk (processes) is preempted and another process is executed.**

(1) OS takes control through interrupt

(2) Saves old context in the Process Control Block (PCB)

(3) Reloads new context from the new process PCB

(4) Returns control to the application / program

## Problem #2

**Define process CPU-burst.**

CPU burst is the total time a process spends in the running state in the CPU, after it was dispatched from the ready queue.

**Explain why a CPU burst size is important for process scheduling. It seems that the OS needs to know this size before executing the process to decide when to schedule the process.**

Some scheduling algorithms like SJF, SRT are used to estimate CPU burst time to maximize the throughput of the concurrent operation and minimize the average waiting time for each process

CPU burst cycle refers to the alternating cycle between CPU burst and I/O burst.

**How would the OS determine the CPU-burst size in advance?**

The burst time cannot be determined beforehand.

The burst time is an estimate based on an initial starting default burst value and actual historical run values. The last estimated burst value and the last actual burst value are averaged to provide the process’ next burst.

Add function.

<https://cs.stackexchange.com/questions/50438/how-does-the-os-determine-the-cpu-burst-time-of-a-process>

**In the below listed scheduling disciplines, name the ones that use the CPU-burst and explain how it is used in each discipline.**

* First Come First Served (FCFS)
* Shortest Job First (SJF)
  + CPU-burst
* Round-Robin (RR)
* Shortest Remaining Time (SRT)
  + CPU-burst

## Problem #3

**Define the problem of disk read // write head scheduling.**

Multiple I/O requests may arrive by different processes and only one I/O request can be served at a time by the Disk Controller. Thus other I/O requests need to wait in the waiting queue and need to be scheduled. Two or more requests may be far from each other so can result in greater disk arm movement. Thus, we need efficient disk management to reduce the effects of such access operation.

------------------------------

Every file operation may require movements of the read/write head.

This is known as the access operation, is very time consuming.

Thus, we use disk management to reduce the effects of such access operation.

<http://www.cs.iit.edu/~cs561/cs450/disksched/disksched.html>

<https://www.geeksforgeeks.org/disk-scheduling-algorithms/>

**Describe first-come first served (FCFS) and shortest seek time first (SSTF) scheduling techniques.**

First Come First Served (FCFS): FCFS is the simplest of the scheduling algorithms. Requests are served in the order that they arrive in the disk queue.

Shortest Seek Time First (SSTF): Select the request that requires the shortest seek time from the current track.

**Explain the problems with these techniques. Name and explain at least one technique that eliminates these problems.**

First Come First Served is the worse disk scheduling algorithm, since the Disk Head could oscillate between between a large number of tracks.

Shortest Seek Time First (SSTF) might lead to starvation, where if many requests that are close to each other continue to arrive other requests will never be served since the distance will always be greater.

# 

# Fall (November) 2016

## Problem #1

**a) Define CPU burst cycle. Differentiate it from I/O burst cycle.**

**b) Differentiate I/O bound and CPU-bound processes.**

**c) Differentiate preemptive and non-preemptive (cooperative) CPU scheduling. Define process starvation.**

**d) Out of the four elementary process scheduling disciplines: First-come, first-served (FCFS), shortest job first (SJF), round robin (RR) and shortest remaining time (SRT), name the disciplines that may result in process starvation. Explain your answer.**

a)CPU burst cycle is the total time a process in the running state, uses a process CPU (performing calculations).

b) An I/O bound process spends more time doing I/O than computations. A CPU-bound process spends more time doing computations.

c) In preemptive CPU scheduling the Operating Systems preempts / forces the process to give up the CPU - usually done by timer interrupt. This prevents long processes from monopolizing the CPU.

In cooperative CPU scheduling processes voluntarily give up the processor to another process in the ready queue.

Process starvation is a process which could be dispatched to the CPU but stays indefinitely in the blockching state and never gets scheduled to run.

d) Shortest Job First (SJF), and Shortest Remaining Time (SRT) are the two elementary process scheduling disciplines that may result in process starvation. SJF will prevent long processes from being scheduled to run if shorter jobs continuously are placed in the queue. SRT will prevent long CPU burst processes from running if shorters ones show up.

## Problem #2

**a) Motivate the need for a page-replacement algorithm in demand-paging in virtual memory organization.**

**b) Explain how least recently used algorithm (LRU) operates. Explain why it is never used in practice.**

**c) Describe the operation second-change page replacement algorithm. Explain why it approximates LRU.**

a) Paged Memory Management gives rise to the notion of demand paging using virtual memory. While not in use, the pages are stored on disk - **secondary storage** (disk) to allow for more active pages to remain in main memory..

* The page table indicates whether the page is in **main memory** (RAM or on the secondary storage (disk)

Whenever a process refers to a page that is not present in main memory (RAM), a page fault occurs. Subsequently, the OS replaces one of the existing pages with the referred page. Page replacement algorithms are an important part of virtual memory management because it helps the OS decide which memory page can be moved out to make space for the requested page.

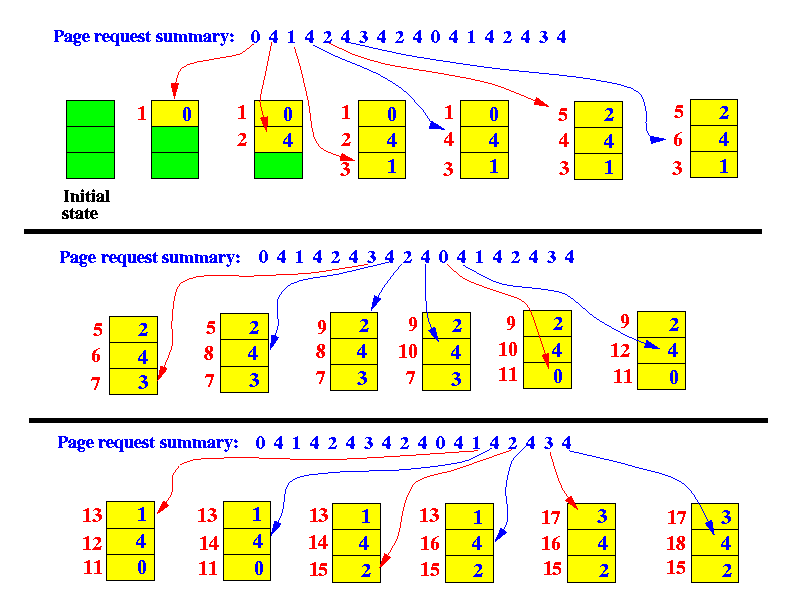
<https://www.geeksforgeeks.org/program-for-least-recently-used-lru-page-replacement-algorithm/>

b) Least Recently Used (LRU) is a greedy algorithm where the page to be replaced is the least recently used page. It can be implemented in the following way:

* Associate a clock register with every page frame - store the last time that the age in that frame was accessed
* Use a “logical clock” that advances by 1 tick each time a memory reference is made
* Each time a page is referenced, update its register
* During replacement, scan through all the pages and find the one with the lowest value in its clock register

It is never used in practice because it requires that we scan all pages to determine which page to replace.

c) Second-chance page replacement algorithm



<http://www.mathcs.emory.edu/~cheung/Courses/355/Syllabus/9-virtual-mem/LRU-replace.html>

## Problem #3

**(a) Define the concept of a block in disk storage.**

**(b) Motivate the need for free space disk management.**

**(c) Explain the bit-vector and linked list free disk space management techniques. Compare their advantages and disadvantages.**

(a) A block in disk storage, sometimes called a physical record, is a sequence of bytes or bits, usually containing some whole number of records having maximum length - block size.

<https://en.wikipedia.org/wiki/Block_(data_storage)>

(b) DIsk space is limited we must reuse the space from deleted files, for new files, if possible.

(c) Bit-vector - uses one bit to note whether the block is free (1) or allocated (0).

**Advantages**

- relatively simple to implement,

- efficient to find the first free block or n consecutive blocks on disk

- can improve performance by caching bit vector in memory

**Disadvantages**

- Only useful when bit vector can be kept in memory

- As the bit vector gets bigger this is difficult to do

LinkedList - Linking together all free disk blocks, keeping a pointer to the first free block in a special location on disk and caching it in memory. LinkedList structure, so first block contains a pointer to the next free disk block, and so on

Advantages

- Easy to find free block, just get head of linked list

-Disadvantages

- Traversing a list is not ideal

# Spring (April) 2016

## Problem #1

**(a) Differentiate between a process and a thread. Motivate the use of threads.**

**(b) Differentiate between user thread implementation and kernel thread implementation.**

**(c) Define many-to-one, one-to-one and many-to-many multithreading models and compare their relative advantages and disadvantages.**

**(d) Define and motivate the concept of a thread pool.**

**(a)** A process is a program under execution, i.e. an active program. It has its own program counter, process stack, registers and program code.

A thread is a lightweight process that can be independently managed by the scheduler. It improves the application performance by using parallelism. A thread shares data, code segments, and files with its peers.

Notable differences:

* 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 more lightweight.
* Memory Sharing: Processes do not share memory with other processes. They are independent. A thread may share some memory with other threads within the same process.
* Creation Time: Processes are more expensive to create, require more time for creation. Threads are cheaper to create, they only need a stack and storage for registers.
* Communication: Communication between processes requires more time Communication between threads is simpler and requires less time, as they share the same address space.

<https://www.tutorialspoint.com/difference-between-process-and-thread>

**(b)** A thread is a lightweight process that can be managed by the scheduler. It improves application performance by using parallelism. A thread shares information like data segments, code segments, files, etc. with its peers while each thread contains its own set of register, a stack, counter, etc.

**User threads** are implemented by users and the kernel is not aware of the existence of these threads. It handles them as if they were single-threaded processes. User-level threads are small and much faster than kernel level threads. They are represented by a Program Counter (PC), stack, registers and a small Thread Control Block (TCB). There is no kernel involvement in synchronization for user-level threads.

**Kernel thread** implements are handled directly by the operating system and thread management is done by the kernel. The context information for the process as well as the process threads are all managed by the kernel. Hence, kernel-level threads are slower than user-level threads.

<https://www.tutorialspoint.com/user-level-threads-and-kernel-level-threads>

<https://www.tutorialspoint.com/operating_system/os_multi_threading.htm>

**(c)** Thread Programming Model. Many operating systems support kernel thread and user thread in a combined way. The user threads can be mapped to kernel threads.

**One-to-One Relationship**:

This model creates a separate kernel thread to handle each an every user thread.

Advantages:

* This model provides more concurrency than the many-to-one model by allowing another kernel thread to run when a thread makes a blocking system call.
* It supports multiple threads to execute in parallel on microprocessors.

Disadvantages:

* Creating user threads requires the corresponding Kernel thread

<https://www.tutorialspoint.com/operating_system/os_multi_threading.htm>

<https://www.studytonight.com/operating-system/multithreading>

<https://www.geeksforgeeks.org/multi-threading-models-in-process-management/>

**Many-to-One Relationship**:

In this model, many user-level threads are all mapped onto one single kernel thread.

Advantages

* Efficiency - thread management is handled by the thread library in user space

Disadvantages

* When a thread makes a blocking system call, the entire process will be blocked
* No parallelism for multiprocessors - only one thread can access the Kernel at a time. So multiple threads are unable to run in parallel on multiprocessors.

<https://www.tutorialspoint.com/operating_system/os_multi_threading.htm>

<https://www.studytonight.com/operating-system/multithreading>

<https://www.geeksforgeeks.org/multi-threading-models-in-process-management/>

**May-to-Many Relationship**: This model multiplexes any number of user threads onto an equal or smaller number of kernel threads --- combining the best features of the one-to-one and many-to-one models.

Advantages

* Number of kernel level threads are specific to the machine
* Parallelism -- developers can create as many user threads as necessary and the corresponding Kernel threads can run in parallel on a multiprocessor machine
* Blocking system call does not block the entire process, the kernel can schedule another thread for execution

Disadvantages

<https://www.tutorialspoint.com/operating_system/os_multi_threading.htm>

<https://www.studytonight.com/operating-system/multithreading>

<https://www.geeksforgeeks.org/multi-threading-models-in-process-management/>

**(d)** A thread pool is a group of pre-instantiated idle threads waiting to be given work. This model is preferred over instantiating new threads for a large number of short tasks. This prevents from incurring the overhead of creating a thread a large number of times.

## Problem #2

**Explain why operating system design needs the concept of synchronization primitives, such as semaphores. Define semaphore. Explain the operation of the two semaphore operations: wait() and signal(). Define the critical section (mutual exclusion) problem. Provide a semaphore-based solution (write code) to the critical section problem.**

The need for synchronization arises when we need processes to run concurrently. The main purpose of synchronization is to allow resources to be shared without interference using the concept of mutual exclusion - only one thread is allowed to access the shared resources or allowed to enter the critical section. Critical section refers to the part of the program that must be protected.

A semaphore is a type of synchronization primitive (variable or abstract data type) used to control access to common shared resources.

The operation wait (P) **decrements** the value of semaphore variable by 1.

If the new value of the semaphore is negative, the process executing wait is blocked (i.e. added to the semaphore’s queue). Otherwise, the process continues execution, having used a unit of the resource.

The operation signal(V) **increments** the value of semaphore variable by 1.

After the increment, if the pre-increment value was negative - meaning that there are processes waiting for the resource - it transfer a blocked process from the semaphore’s waiting queue to the ready queue.

## Problem #3

Explain the concept of a page table. Motivate the need for a hierarchical page table. Explain the address lookup diagram if in a two-level hierarchical page table (draw a diagram if necessary).

# Fall (November) 2012

## Problem #1

**What is the purpose of a short-term CPU scheduler? What is the difference between a preemptive and non-preemptive scheduler? Which type of scheduling (preemptive or non-preemptive) is used in most modern operating systems and why is it used? Explain round-robin (RR) scheduling, and how its behavior and performance varies with the size of the time quantum? Are there any constraints on how the quantum should be chosen?**

Short-term CPU scheduler is used to select among the processes that are already in the ready queue. Its main goals is to increase system performance.

Long-term CPU scheduler determines which programs are admitted to the system for processing. It selects processes from the queue and loads them into memory for execution.

The goals of the long-term CPU is to provide a balance between different kinds of jobs such as I/O bound and processor bound.

Medium Term Scheduler is part of swapping.

It removes the processes from memory.

Preemptive: SRTF, RR

Non-preemptive: FCFS, SRT

RR - round robin algorithm.

* Define a fixed time slice, also called Time Quantum
* Choose process from the head of the Ready Queue
* Run that process for at most one time slice
* If process hasn’t finished, add it to the tail of the ready queue
* Choose another process from the ready queue and run it for at most one time slice

The performance of RR is sensitive to the time quantum selected.

If the quantum is large enough, the RR reduces to FCFS algorithm.

If the quantum is very small, then each process gets 1/nth of the processor time and share the CPU equally. But, in practice, if this happens then there would be an overhead for every context switch - the smaller the time quantum the more context switches there would be.

Most modern systems use a 10 - 100 millisecond time slice, and context switch times are on the order of 10 microseconds so the overhead is small relative to the time quantum.

<https://www.cs.uic.edu/~jbell/CourseNotes/OperatingSystems/5_CPU_Scheduling.html>

## Problem #2

Explain the difference between contiguous (continuous) allocation, linked (chained) allocation, and indexed allocation for file systems. Explain the combined scheme (multilevel indexed allocation) used by the Unix File System (UFS) and how it implements access for small and large files.

**Contiguous allocation** - each file occupies a continued set of blocks on disk. For example, if the file requires *n* blocks and is given a block *b* as the starting position then the blocks assigned to the file will be *b*, *b + 1*, *b + 2, b + 3, …, b + n-1*. This allows for direct access by referencing a specific block and is fast since the number of seeks is minimal since all blocks are contiguous. But, it is subject to fragmentation and it is difficult to increase the file size since all blocks have to be contiguous.

**Linked List Allocation** - each file is a linked list of disk blocks, where each block holds a pointer to the next block. The directory entry contains a pointer to the starting block. This makes it easy to increase the file size since we don’t need contiguous block, but also increases the number of seeks due to the random allocation of linked blocks.

**Index Allocation** - each file has an index block which contains pointers to all the blocks occupied by a file. The ith entry of the index block contains the address of the ith file block. This scheme support direct access to the blocks, therefore provides fast access time to the file. It also overcomes fragmentation. But, there’s an overhead for index allocation that is greater than linked allocation.

**In multilevel indexing** - a first level index block points to a second level index block which in turn points to the index blocks of the actual file.

<https://www.geeksforgeeks.org/file-allocation-methods/>

## Problem #3

**Explain the concept of a thread and of multithreaded programming. What are the benefits of multithreaded programming? Explain the difference between user and kernel threads and discuss the three models for establishing a relationship between the two.**

A thread is a lightweight process that represents a single sequential execution stream within a process.

Multithreading is an execution and programming model that allows multiple threads to concurrently perform work within the context of one process. These threads share the process' resources.

This allows for the benefits of parallelism on a multiprocessing system.

User-level threads are created by the user and the kernel is not aware of them. The kernel handles them as if they were a single-threaded process. They are much smaller and faster than kernel-level threads

Kernel-level threads are handled directly by the operating system and thread management is done by the kernel. The context information for the process as well as the process threads are all managed by the kernel. Hence, the kernel-level threads are slowerd.

3 Relationship Model between User-level and Kernel-level threads

One-to-one relationship. One user-level thread to one kernel-level thread. Allows for more concurrency than the many-to-one model by allowing another kernel thread to run when a thread makes a blocking system call. Disadvantages -- The overhead of requiring one kernel thread per user-level thread.

Many-to-one. Many user-level threads are mapped to a single kernel-level thread. Thread management is handled by the thread library in the user space. Disadvantages - if one thread performs a blocking system call then the entire process will be blocked.

Many-to-Many relationships. Multiplexes any number of user-level threads onto an equal or less number of kernel-level threads. The number of kernel threads depends on the specific machine. Advantages -- parallelism is achieved a developer can create any number of user-level threads and the corresponding kernel-level threads will run in parallel on a multiprocessor machine. Blocking system call do not block the entire process.

# Notes

<https://www.studytonight.com/operating-system/process-scheduling>

<https://www.tutorialspoint.com/operating_system/os_memory_management.htm>

<https://www.youtube.com/watch?v=JDsrcoLKKhg&list=PLIY8eNdw5tW_lHyageTADFKBt9weJXndE&index=6>

Raid:

<https://www.youtube.com/watch?v=wTcxRObq738>