# Indian Institute of Technology Guwahati

Department of Computer Science and Engineering

End Sem Examination Course: CS343 (Operating System) 21st Nov 2023, 2PM-5PM **Full Marks: 50 (scaled to 40)**

(***Write assumption clearly if you assume anything for answering the questions***)

==========

1. **[10 (= 5+5) Marks] [Scheduling]**

a) [1+4] Describe the scheduling problem $P \mid p_j, pmtn, tree, d_j=D, a_j=0 \mid L_{max}$ . Assume execution time of task is $p_j$ and $p_j$ is a positive integer and the task can be pre-empted at most $(p_j-1)$ times (we can say it is ***limited pmtn)***. Solve the problem efficiently.

ANS: Given m identical processor and n tasks with arbitrary execution. All tasks arrives at time 0, all tasks have common deadline D and have tree dependency among them. Pre-emption is allowed but a task can be pre-empted on $p_j$-1 times. $L_{max}$ need to be minimized, $L_{max}$ is max of ($f_i$-D) of all tasks where $f_i$ is finishing time of task.

As the all tasks have common deadline if max $\{f_i\}$ or $C_{max}$ is greater than D, we need to minimize max $\{f_i\}$ which is same as minimization of $C_{max}$. So problem is same as $P \mid p_j, pmtn, tree, d_j=D, a_j=0 \mid C_{max}$.

As pmtn is allowed can each task can be pre-empted $p_{j-1}$ times, we can easily break/split each task with execution time $p_j$ to $p_j$ number serial sub tasks with unit execution time. After splitting all the tasks the resultant constructed one will be still a tree with each sub-tasks of unit execution time. Number of node of the resultant tree will be order of $S=\Sigma p_j$. As $P \mid p_j=1, tree \mid C_{max}$ can be solved in polynomial time using HU's level based approach (**which is Critical Path Algorithm: it is not simple List Scheduling**), we can solve the original problem in polynomial time with number of unit time tasks which is **Pseudo-polynomial with n = $\Sigma p_j$**. HU's algorithm complexity is O(V+E) and here V is $\Sigma p_j$ and E is same as earlier/original tree along with $\Sigma p_j$ number of extra induced edges by the splitting.

b) [1+4] Describe the scheduling problem $P \mid p_j=\{x,2*x\}, a_j=0 \mid C_{max}$ . The value of $p_j$ can have two possibilities $x$ and $2*x$ for all the tasks, where $x$ is a positive number. Solve the problem efficiently.

ANS: Given m identical processor and n tasks with execution time either x or 2x, all tasks arrived a time 0, pre-emption is not allowed. Tasks need to be scheduled on m processors such that overall execution time is minimized. As the tasks are of only two types and the longer task is exactly two times the length of the shorter tasks. Use LPT (longest processing time first) Scheduling, where longer tasks get first scheduled on m processor, then scheduling shorter tasks get scheduled, this approach will give minimum $C_{max}$. Suppose there are n1 type1 tasks with x execution time and n2 type2 tasks with 2x execution time, scheduling first type2 tasks and after that type1 task will minimize the $C_{max}$. It can be solved mathematically based on number of n, m, n1 and n2 using simple calculation;

2. **[10 (= 5+5) Marks] [Synchronization and Deadlock]**

a) [2+3] What are the properties of ***wait ()*** and ***signal ()*** procedures of semaphore? Give an example of incorrect use of ***wait()*** and ***signal()***, where the system may enter to a deadlock.

ANS: wait() and signal() need to be atomic (or in java term it synchronized only one thread should execute at time). **Atomicity:** wait(s) and signal(s) are atomic also in the sense that no two processes can be inside wait(s) and signal(s) at the same time.

Deadlock–two or more processes are waiting indefinitely for an event that can be caused by only one of the waiting processes. Let semaphore S and Q initialized to 1 and the following condition occurs

P0 : P1:

wait(S); wait(Q); wait(Q); wait(S);

signal(S); signal(Q); signal(Q);signal(S)

Holding one and waiting for others may happen in both the process and go to deadlock. The sequence can be different.

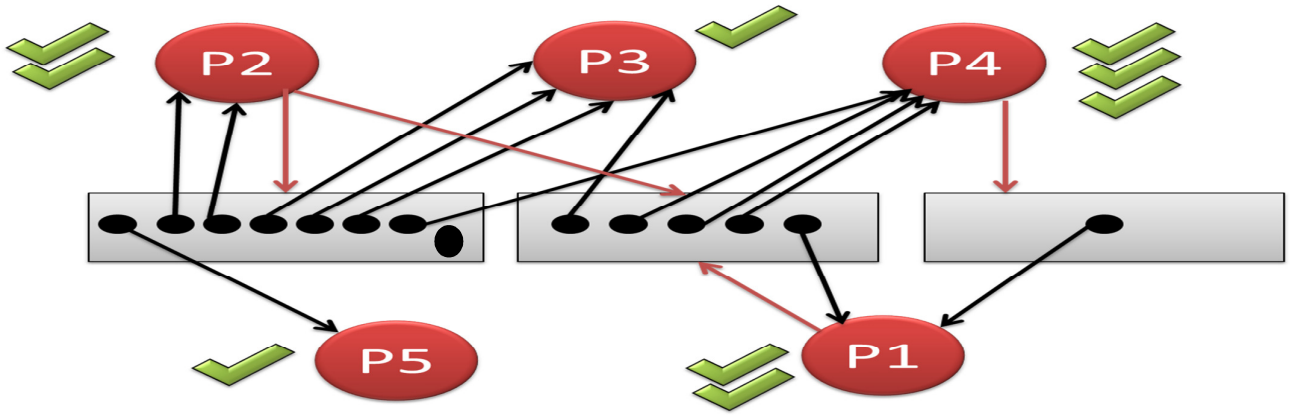
b) [2+3] Given the process resource usage and availability in Table, (a) Draw resource allocation graph (RAG), (b) Draw all the reduced **RAGs** and check if the system is in the deadlock state or not.

| Process | Current Resource Allocation | | | Outstanding Requests | | | Resource Available | | |
|---|---|---|---|---|---|---|---|---|---|
| | R1 | R2 | R3 | R1 | R2 | R3 | R1 | R2 | R3 |
| P1 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 0 |
| P2 | 2 | 0 | 0 | 1 | 1 | 0 | | | |
| P3 | 3 | 1 | 0 | 0 | 0 | 0 | | | |
| P4 | 1 | 3 | 0 | 0 | 0 | 1 | | | |
| P5 | 1 | 0 | 0 | 0 | 0 | 0 | | | |

ANS: RAG is shown. As P5 and P3 no pending/outstanding requests both can execute in first round. After execution of P3, we can execute/satisfy requirement of either P1 or P2. After execution of either P1 we can execute P4.

Safe Execution Sequence: P5|| P3, P2, P1, P4 or P3||P5, P1, P4, P2

**There is no deadlock in the system.**



3. **[10 (= 5+5) Marks] [Memory Management]**

a) [3+2] Explain the techniques to reduce page fault penalty? Given memory access time of 200ns and average page fault service time of 10 mili-seconds, **estimate the bound on page fault rate** such that expected memory access time is 220ns.

ANS: Techniques to reduce page fault penalty: (a) page size reduction, (b) through away unwritten page or write back only the dirty page, (c) introducing page buffer to store the new incoming page and avoiding demand time page replacement, (d) improved page replacement and reduced number of frames, (e) high speed disks

Target effective access time EAT is 220ns

**=(1-p)*200ns+p*10ms** = 220ns => (1-p)*200+p*($10^7$)ns=220ns

=> 200ns-p*200ns+p*$10^7$ns=220ns = p($10^7$-200) =20 => p = 20/($10^7$ -200) = **0.00000200004**

b) [1.5+1.5+2] What is Belady's Anomaly in page replacement? Construct an example which shows Belady's Anomaly? What are the major demerits of ***move to front (MTF)*** data structure implementation of LRU page replacement policy?

ANS: Belady's Anomaly is the phenomenon where increasing the number of page frames results in an increase in the number of page faults for a given memory access pattern. This phenomenon occurs for FCFS/Random/Second Chance replacement algorithm but not for the stack based LRU/Optimal replacement algorithms.

Construction of Example for Belady Anamoly: ***This question is supposed to be for pure algorithmic topic and not for proper system course but we asked to test your thinking skill.***

Ref String "1, 2, 3, 4, 5, 1, 2, 5, 1, 2, 3, 4, 5" ==> with skipped 3 and 4

will have anomaly in FCFS replacement with 3 and 4 frames. 3 Frames will have 9 PF and 4 frames have 10 PF.

Further ref: Belady, Nelson and Shedler constructed reference strings for which FIFO page replacement algorithm produced nearly twice as many page faults in a larger memory than in a smaller one..

Major demerits of MTF implementation of LRU is require 6 pointer adjustment and searching probabilistically 50% number of allocated frame on an average and in worst case total number frames for a page access. As the number of allocated frames may goes above 10^6, it takes huge amount of time to search the accesses page and for every page access this search needed to be performed. This increases the overall page access time.

4. **[10 (= 6+4) Marks] [Demand Paging]**

a) [2+2+2] How to solve these issues of simple paging? (i) Large page table (ii) Loading of all the pages of the process to the memory and (iii) Frequent accesses to page table memory.
ANS: Large page table can be handled by paging the page table/multi-level page table, hashing and inverted page table.

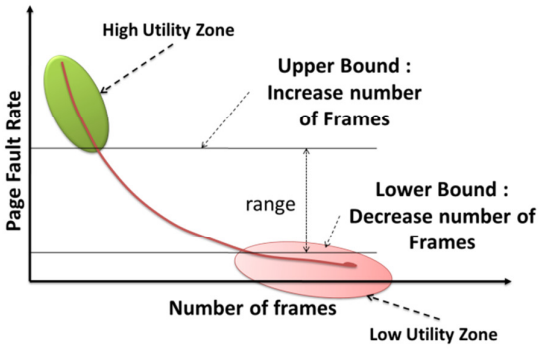
Loading of all the pages of process to the memory can be handled by demand paging and dynamic page loading,

Frequent access to the page table memory can solve by using TLB (translation look-ahead buffer) to reduce the page table access.

b) [2+2] What is utility based frame allocation? Calculate working set ***WS (t, Δ)*** with ***t*** between for 8th and 9th reference and ***Δ=6*** for the given reference to the following pages in a program.

**Ref: 0,2,2,1,5, 1,8,7,8,5, 1,2,8,2,7, 8,2,3,8,3**

ANS: Estimate utilization by allocating extra frames and decide based on that. If page fault is getting lower by allocating an extra frame then beneficial . Establishing "acceptable" **page-fault frequency (PFF)** rate and use in local replacement policy. If actual rate too low, process loses frame and If actual rate too high, process gains frame

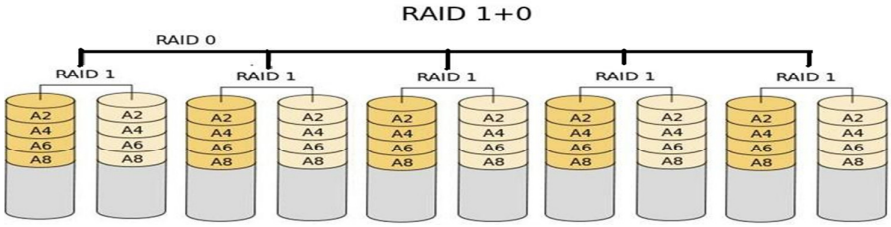



Ref: 0,2, *2 ,1,5, 1 ,8,7*, *t is here*, 8 ,5, 1,2,8,2,7, 8,2,3,8,3
Last six references shown in large italic blue: working set **WS (t, Δ) = {2, 1, 5, 8, 7}** , cardinality/WS size is 5. Duplicate item in the set get removed.

5. **[10 (= 3+3+4) Marks] [Disk and File Management]**

a) [3] Calculate the effective capacity and bandwidth of RAID (1+0) system with a total of 10 disks, with each disk of capacity 1TB and speed 10MBPS.

ANS: RAID 0 uses striping to increase capacity and bandwidth and RAID 1 uses mirroring to introduce redundancy. RAID 1+0 uses mirroring and striping, which say at lower level use mirroring (so need to create 5 groups of disk with each group happing two disks: one for data other for mirror) and these 5 groups will form RAID 0 for stripping. Logically picture looks like this:




So Effective capacity will be: 5*1TB=5TB and Speed will be 5*10MBPS=50MBPS.

b) [3] Compare the performance of FCFS, Greedy/Shortest Seek First, and Elevator approaches for disk arm scheduling for the following cylinder requests (assume range of cylinder 0-199 and head pointer at **100**th cylinder) :

**98, 183, 37, 122, 14, 124, 65, 67**

ANS**: Performance can be measure by Total seeks time or Average seeks time**. One can show graphical arm movement but we are showing textually.

Total seek time for FCFS service order 98, 183, 37, 122, 14, 124, 65, 67

= |100-98|+|98-183|+|183-37|+|37-122|+|122-14|+|14-124|+|124-65|+|65-67|

= 2+ 85+146+85+108+110+59+2= 597

Total seek time for SSF for service order 98, 122, 124, 67, 65, 37, 14, 183

= |100-98|+|98-122|+|122-124|+|124-67|+|67-65|+|65-37|+|37-14|+|14-183|

= 2+ 24+2+57+2+28+23+169= 307

Total seek time for elevator approach: Head is at 100, to serve 98 it will move downward direction and serve all the requests, after that it will move towards upwards to serve the remaining request. Serving Order: 98, 67, 65, 37, 14, (Choose not to go to zero), 122, 124, 183.

Total Seek time: |100-14|+|14-183|=86+169=255 **Elevator algorithm perform the best among FCFS, SSF and Elevator.**

c) [4] How files are identified uniquely in disk of a typical UNIX file system? What is superblock of a disk partition?

ANS: Unix file system uses **Inode number** to identify uniqueness of file in a disk in UFS.

The superblock essentially records a file system's characteristics – block size, other block properties, sizes of block groups and location of **inode tables**. The superblock is especially useful in UNIX and similar operating systems where a root directory contains a variety of subdirectories. The superblock holds metadata about the file system, like which inode is the top-level directory and the type of file system used. If you are unable to mount your device, this can be due to a Corrupted superblock.