Indian Institute of Technology Guwahati

Department of Computer Science and Engineering

Mid Sem Examination Course: CS343 (Operating System) 20th Sept 2023, 2PM-4PM Full Marks: 50 (scaled to 35)

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

==========

1. **[10 (= 2+5+3) Marks] [OS Services and Modes]**

a) [2] What characteristic is the same in trap, interrupt, and system calls but different in subroutine calls?

*ANS: trap, interrupt, and system calls **cause machine to shift to kernel mode** but subroutine calls **does not change the execution mode**.*

b) [2+3] What is a **system call**? How and what types of services does OS provide to users using system calls?

ANS: (Part A:2 Marks) Any one of the bellow:

- *A* **system call** is a programmatic way in which a computer program requests a service from the kernel of the operating system it is executed on.
- A system call is a way for programs to **interact with the operating system**.
- A computer program makes a system call when it makes a request to the operating system's kernel.
- System call **provides** the services of the operating system to the user programs via Application Program Interface(API)

Part (B: 3 Marks)

System call **provides** the services of the operating system to the user programs via Application Program Interface(API), System calls are essential for the proper functioning of an operating system, as they provide a standardized way for programs to access system resources.

Service Provides by OS for Process creation and management (end, abort, create, terminate), Main memory management (allocate, and free memory), File Access, Directory, and File system management (create, open, close, delete, read files,s, etc.) : Device handling(I/O), Protection, Networking, etc.

c) [3] What are the differences between a monolithic kernel and a microkernel? How keeping option of **loadable kernel module** help in designing OS in a better way?

Ans: In a monolithic kernel, all system services run in kernel space, whereas in a microkernel, only the most basic services (such as memory management and process scheduling) run in kernel space, with other services running in user space). Any two/three of these following:

Parameters	Microkernel	Monolithic kernel
Address Space	In microkernel, user services and kernel services are kept in separate address space.	In monolithic kernel, both user services and kernel services are kept in the same address space.
Design and Implementation	OS is complex to design.	OS is easy to design and implement.
Size	Microkernel are smaller in size.	Monolithic kernel is larger than microkernel.

Functionality	Easier to add new functionalities.	Difficult to add new functionalities.
Coding	To design a microkernel, more code is required.	Less code when compared to microkernel
Failure	Failure of one component does not effect the working of micro kernel.	Failure of one component in a monolithic kernel leads to the failure of the entire system.
Processing Speed	Execution speed is low.	Execution speed is high.
Extend	It is easy to extend Microkernel.	It is not easy to extend monolithic kernel.
Communication	To implement IPC messaging queues are used by the communication microkernels.	Signals and Sockets are utilized to implement IPC in monolithic kernels.
Debugging	Debugging is simple.	Debugging is difficult.
Maintain	It is simple to maintain.	Extra time and resources are needed for maintenance.
Message passing and Context switching	Message forwarding and context switching are required by the microkernel.	Message passing and context switching are not required while the kernel is working.
Services	The kernel only offers IPC and low-level device management services.	The Kernel contains all of the operating system's services.
Example	**Example :** Mac OS X.	**Example :** Microsoft Windows 95.

Ans: Loadable kernel modules **optimize system resources by loading modules on demand**. This reduces memory usage and system footprint, enhancing performance by unloading unused modules. Easy Debugging and Testing. Kernel modules facilitate debugging and testing of specific features or drivers.

They extend the functionality of the kernel without the need to reboot the system. A module can be configured as built-in or loadable. To dynamically load or remove a module, it has to be configured as a loadable module in the kernel configuration.

- The kernel does not have to rebuild your kernel as often
- t is easier to diagnose system problems: A bug in a device driver which is bound into the kernel can stop the system from booting at all. But LKM is easy to track and fix the bug
- Using modules can save memory, because they are loaded only when the system is actually using them
- Modules are much faster to maintain and debug.

2. **[10 (= 6+4) Marks] [Process Management]**

a) [2+2+2] What is a process control block (**PCB**)? What information gets stored in a **PCB**? When is the information on PCB used for process scheduling?

Ans: A Process Control Block in OS (PCB) is **a data structure used by the operating system to manage information about a process**.

It contains information about the process state, program counter, CPU register, memory allocation, CPU usage, I/O devices, open files, and CPU scheduling information and other resources used by the process. Basically, it store the context of theprocessor.

The PCB is used by the operating system to keep track of each process in the system. When a process is created, the operating system creates a new PCB for the process and stores it in the kernel memory. When the process is terminated, the PCB is removed from the memory. ***Also, the PCB store the context information used in context switching during the scheduling process and scheduling attribute of PCB used on process scheduling.***

b) [2+2] What are the differences between long-term schedulers and short-term schedulers? Explain the scenarios in which a process can enter the ready state.

ANS: Long-Term Scheduler is also known as **Job Scheduler.** Long-term scheduler regulates the programs which are selected to system for processing. In this the programs are setup in the queue and as per the requirement the best one job is selected and it takes the processes from job pool. **LTS/JS bring job to memory.** It regulates the Degree of Multi-programming (DOM). Number of activation of LTS is very low. Long-Term Scheduler changes the process state from **New** to **Ready**.

Short-Term Scheduler is also known as **CPU Scheduler.** Short-Term Scheduler ensures which program is suitable or important for processing. Number of activation of STS is very high. Short-Term Schedule **control Multitasking**. Short-Term Scheduler changes the process state from **Ready** to **Running**.

ANS: There many way a job can enter to ready state (a) New job created to LTS put into Ready queue, (b) time quatum elapsed (c) waitting child process finished, (d) waiting I/O finished for the process, (e) high pririty job came, (f) interrupt came

3. **[10 (= 2+4+4) Marks] [Scheduling]** ***[Hint: Problem of 3 (b) and 3 (c) are Polynomially Solvable]***

a) Given five processes with execution times 10, 20, 5, 3, and 18. Assume all arrived at time 0, and calculate the **average completion time** of tasks if scheduled using (1) SJF, and (2) RR with time quantum of 2 time unit.

ANS: SJF Gantts Chart 3+5+10+18+20

SJF Agv(Ci)= (3+8+18+36+56)/5=24.2

RR2= A B C D E A B C **D1** E A B **C1** E A B E A

B E B E B E B E B **E B**

RR2 Agv(Ci)= (17+24+34+54+56)/5=37.0

b) Describe the scheduling problem $\boldsymbol{P \mid p_j, pmtn, d_j=D, a_j=0 \mid \sum U_j}$. Solve the problem efficiently. Assume $\mathbf{p_j \leq D}$.

ANS: N tasks with **arbirary execution** and **common deadlines D** and all task arriaves at time 0, needs to executed on **M identical processors** with **allowing pre-emptions** such that **number of late task to be minimized.**

One Approach: Sort the job based on execution time (with SJF order). Consider the job in the SJF order and map to P processor if it can fit before deadlines. Suppose we consider first k jobs, then (as pre-emption is allowed) the condition can be checked by $\sum_{j=1}^{j=k} p_j <= M.D$, where M is number of processor and D is deadlines. This is give optimal result and one can use exchage argument approach to prove.

c) Describe the scheduling problem $\boldsymbol{R \mid p_{j,i}, a_j=0 \mid \sum C_j}$. Solve the problem efficiently.

ANS: N tasks with **arbirary execution** and arrives at time 0 need to executed on M **unrelated processors, without allowing pre-emptions** such that **sum of completion time of tasks to be minimized.** For unrelated processor, execution time of a task on different processor is different is given by p_{ji}, which is execution time of jth task on ith processor.

[[Ref: Section 5.13 of Scheduling Algorithm Book]]

We reduce this problem to an assignment problem. Again, if $i_1, i_2, \ldots, i_r$ is the sequence of jobs processed at machine M_j, then the contribution of these jobs in the objective function is $\mathbf{r.p_{i1\,j} + (r-1)p_{i2\,j} + \ldots + 1p_{ir\,j}}$.

We define a position of a job on a machine by considering the job processed last on the first position, the job processed second from last on the second position, etc. **If a job is scheduled first, then all the r jobs need to wait for the first job competion time including the first job.**

we have to assign the jobs i to positions k on machines j. The cost of assigning job i to (k, j) is kpij . Note that an optimal solution of this assignment problem has the following property: if some job i is assigned to position k > 1 on machine j, then there is also a job assigned to position k − 1 on machine j. Otherwise, scheduling job i in position k − 1 would improve the total assignment cost (provided that pij > 0). Thus, a solution of the assignment problem always yields an optimal solution of our scheduling problem.

4. **[10 (= 5+5) Marks] [Threading and Scheduling]**

a) [3+2] Suppose the OS scheduler uses the EWMA predictor $\tau_{n+1} = \alpha\, t_n + (1-\alpha)\tau_n$ with **α=0.5** to predict the burst time of the next interval of the process. If the actual burst time behaves randomly and the burst time ranges between time 0 and 100. Estimate the average error of the EWMA predictor in the long run, assuming the initial prediction is 0. How you will change the prediction model to minimize the average prediction error?

ANS: Observed value of is randomly distributed and Estimated value suppose be averaged at 50, suppose whole range is divided into 4 parts 0-24, 25-49, 50-74 and 74-99. Average prediction error

=abs(50-Avg(0:24))*1/4+abs(50-Avg(25-49))*1/4+abs(50-Avg(50:74))*1/4+abs(50-Avg(75:99))*1/4

= abs(50-12)*1/4+abs(50-37)*1/4+abs(50-62)*1/4+abs(50-87)*1/4 = 25.5 (which slightly higher than 25) if we consider 4 parts, but if we increase this to 8 parts, 16 parts, 64 parts then the **error converge to 27.25 //C Code Snipped to test the concept**

```
float E=0, O, Err, TE=0;
for(int i=0;i<10000;i++){
O=rand()%100; Err=O-E;
   printf("i=%d E=%f O=%f Err=%f \n", i, E, O, abs(Err));
TE+= abs(Err); E=(E+O)/2;
 }
printf("Avg Err=%f", TE/10000);
```

Part (b): Set value of τ_{n+1} =50. No way we can reduce less than 27% error.

b) [2+3] What is a thread control block (**TCB**)? What are the differences between the kernel thread and user-level thread.

ANS: Very similar to Process Control Blocks (PCBs) which represents processes, **Thread Control Blocks (TCBs)** represents threads generated in the system. It contains information about the threads, such as it's ID and states.

The components have been defined below:

- **Thread ID, Thread states: CPU information:** It includes everything that the OS needs to know about, such as how far the thread has progressed and what data is being used.
- **Thread Priority:** It indicates the weight (or priority) of the thread over other threads which helps the thread scheduler to determine which thread should be selected next from the READY queue.

ANS : User-Level Thread- The User-level Threads are implemented by the user-level software. the user-level threads are basically created and implemented by the thread library which OS provides as an API for creating the managing synchronizing threads. it is faster than the kernel-level threads, it is basically represented by the program counter, stack, register, and PCB. Example Pthread.

Kernel-Level Thread –So, in terms of the Operating systems basically, the threads are the unit of execution within a process. and the kernel level threads are also kinds of threads which is directly handled via kernel threads management. The Kernel-level threads are directly handled by the OS directly whereas the thread's management is done by the kernel.

Uthread managed by user library, fast, less context switch time, cannot take advantage of multiprocessor (hardware multithread) -cannot take advantage of resource, , blocking operation (If one uthread performs a blocking operation then the entire process will be blocked), any os can support, portable, shared address spaces.

Kthread is supported by OS, takes times (not fast), can take advantage of multicore, better in case of blocking operation, address space is wel separate, fault tolerent (if one thread faii other executes), visible to OS, kernel can be multithread.

5. **[10 (= 3+3+4) Marks] [Threading and Synchronization]**

a) [3] Write three essential properties that need to be achieved by any solution of the critial section problem. Describe those three properties.

ANS: Any solution to the critical section problem must satisfy the following requirements:

Mutual exclusion: When one process is executing in its critical section, no other process is allowed to execute in its critical section.

Progress/No deadlock: When no process is executing in its critical section, and there exists a process that wishes to enter its critical section, it should not have to wait indefinitely to enter it.

Bounded waiting/No Starvation: There must be a bound on the number of times a process is allowed to execute in its critical section, after another process has requested to enter its critical section and before that request is accepted.

b) [1+1+1] Write the assumptions behind **Peterson's lock** for mutual exclusion. What are the properties of **TAS (Test And Set) instruction**, and how does it help in achieving mutual exclusion?

ANS: Load and Store are atomic instructions

ANS: TAS is atomic,

```
boolean TestAndSet (boolean &target){
boolean rv = target;     target = true;     return rv;
}
```

Prperties: 1. Executed atomically 2. Returns the original value of passed parameter
3. Set the new value of passed parameter to “TRUE”.

```
while(1){
        while (TestAndSet(lock));    critical_section()
        lock = false;    remainder section()
}
```

c) [2+2] What is the main disadvantage of **spin lock**? How performance of locking can be improved?

ANS: Lock variable accesses with atomic TAS or CAS very frequently in a **tight loop** and it beacme the bottle neck.

ANS: We can use read lock if it is free then try lock other answer is backup lock or yeild others.