1 (A) What are the three main purposes of an operating system?

Answer: The three main puropses are:

• To provide an environment for a computer user to execute programs on computer hardware in a convenient and efficient manner.

• To allocate the separate resources of the computer as needed to solve the problem given. The allocation process should be as fair and efficient as possible.

• As a control program it serves two major functions: (1) supervision of the execution of user programs to prevent errors and improper use of the computer, and (2) management of the operation and control of I/O devices.

(b) **Difference between operating system for mainframe computers and PCs**

Mainframe computers :

- **Mainframes are typically big boxes containing a large number of processors and a large amount of storage, as well as high-bandwidth buses.**
- **A mainframe operating system is meant to handle a huge number of calculations in a sequential manner**, whereas a PC operating system is designed to process interactive transactions.

Personal computers (PCs) :

- **The hardware of a personal computer is typically designed to deliver speedy responses to the user.**
- These are usually graphical or console interfaces on top of an operating system's kernel and are designed for a lot of user interaction.

(c) 3What is the main difficulty that a programmer must overcome in writing an operating system for a real-time environment?

Answer: The main difficulty is keeping the operating system within the fixed time constraints of a real-time system. If the system does not complete a task in a certain time frame, it may cause a breakdown of the entire system it is running. Therefore when writing an operating system for a real-time system, the writer must be sure that his scheduling schemes don’t allow response time to exceed the time constraint

2) a) A process is essentially running software. The execution of any process must occur in a specific order. A process refers to an entity that helps in

representing the fundamental unit of work that must be implemented in any system.

In other words, we write the computer programs in the form of a text file, thus when we run them, these turn into processes that complete all of the duties specified in the program.

A program can be segregated into four pieces when put into memory to become a process: stack, heap, text, and data. The diagram below depicts a simplified representation of a process in the main memory.

b) NEW

When a process is started/created first, it is in this state.

Ready

Here, the process is waiting for a processor to be assigned to it. Ready processes are waiting for the operating system to assign them a processor so that they can run. The process may enter this state after starting or while running, but the scheduler may interrupt it to assign the CPU to another process.

Running

When the OS scheduler assigns a processor to a process, the process state gets set to running, and the processor executes the process instructions.

Waiting

If a process needs to wait for any resource, such as for user input or for a file to become available, it enters the waiting state.

Terminated or Exit

The process is relocated to the terminated state, where it waits for removal from the main memory once it has completed its execution or been terminated by the operating system.

c)

3) (a)

Parameter	**PREEMPTIVE SCHEDULING**	**NON-PREEMPTIVE SCHEDULING**
Basic	In this resources(CPU Cycle) are allocated to a process for a limited time.	Once resources(CPU Cycle) are allocated to a process, the process holds it till it completes

Parameter	**PREEMPTIVE SCHEDULING**	**NON-PREEMPTIVE SCHEDULING**
		its burst time or switches to waiting state.
Interrupt	Process can be interrupted in between.	Process can not be interrupted until it terminates itself or its time is up.
Starvation	If a process having high priority frequently arrives in the ready queue, a low priority process may starve.	If a process with a long burst time is running CPU, then later coming process with less CPU burst time may starve.
Overhead	It has overheads of scheduling the processes.	It does not have overheads.
Flexibility	flexible	rigid
Cost	cost associated	no cost associated
CPU Utilization	In preemptive scheduling, CPU utilization is high.	It is low in non preemptive scheduling.
Waiting Time	Preemptive scheduling waiting time is less.	Non-preemptive scheduling waiting time is high.
Response Time	Preemptive scheduling response time is less.	Non-preemptive scheduling response time is high.
Examples	Examples of preemptive scheduling are Round Robin and Shortest Remaining Time First.	Examples of non-preemptive scheduling are First Come First Serve and Shortest Job First.

b)

4) a) Various benefits of multithreading in the operating system are as follows:

1. Responsiveness

Multithreading in an interactive application enables a program to continue running even if a section is blocked or executing a lengthy process, increasing user responsiveness.

A server in a non-multithreading environment listens to a port for a request, processes the request, and then resumes listening for another request. Other users are made to wait unnecessarily because of the time it takes to execute a request. Instead, a better approach will be to pass the request to a worker thread while listening on a port.

For instance, a multithreaded web browser permits the user interaction in one thread while a video is loading in another thread. As a result, instead of waiting for the entire web page to load, the user can continue viewing a section of a web page.

2. Resource Sharing

Processes can only share the resources only via two techniques such as:

1. Message Passing
2. Shared Memory

The programmer must explicitly structure such strategies. On the other hand, by default, threads share the memory and resources of the process they belong to.

The advantage of sharing code and data is that it permits an app to execute multiple code threads in the same address space.

3. Economy

Allocating memory and resources for process creation is an expensive procedure because it is a time and space-consuming task.

Because threads share a memory with the process to which they belong, establishing and context switching threads is more cost-effective. In general, generating and managing processes takes far more time than threads.

4. Scalability

The advantages of multi-programming become much more apparent in the case of multiprocessor architecture, when threads may execute in parallel on many processors. When there is just one thread, it is impossible to break the processes into smaller jobs performed by different processors.

A single-threaded process could only run on one processor, despite the number of processors available. Multithreading on multiple CPU machines increases parallelism.

5. Better Communication

Thread synchronization functions could be used to improve inter-process communication. Moreover, sharing huge amounts of data across multiple threads of execution inside the same address space provides extremely high-bandwidth, low-latency communication across various tasks within an application.

6. Utilization of multiprocessor architecture

The advantages of multithreading might be considerably amplified in a multiprocessor architecture, where every thread could execute in parallel on a distinct processor.

A single-threaded task could only run on one of them, no matter how many CPUs are available. On a multi-CPU machine, multithreading enhances concurrency.

The CPU switches among threads so quickly in single-processor architecture that it creates the illusion of parallelism, but only one thread is running at a particular time.

7. Minimized system resource usage

Threads have a minimal influence on the system's resources. The overhead of creating, maintaining, and managing threads is lower than a general process.

b) **Similarities of thread and process:**

- Both can share CPU
- Both can create a child
- If one block then other can run

Advantages of threads:

- Reduce context switching.
- Increase processing speed.

- Don’t need for inter-process communication.

Features	User Level Threads	Kernel Level Threads
Implemented by	It is implemented by the users. It is implemented by the OS.	
Context switch time	Its time is less.	Its time is more.
Multithreading	Multithread applications are unable to employ multiprocessing in user-level threads.	It may be multithreaded.
Implementation	It is easy to implement.	It is complicated to implement.
Blocking Operation	If a thread in the kernel is blocked, it blocks all other threads in the same process.	If a thread in the kernel is blocked, it does not block all other threads in the same process.
Recognize	OS doesn't recognize it.	It is recognized by OS.
Thread Management	Its library includes the source code for thread creation, data transfer, thread destruction, message passing, and thread scheduling.	The application code on kernel-level threads does not include thread management code, and it is simply an API to the kernel mode.
Hardware Support	It doesn't need hardware support.	It requires hardware support.
Creation and Management	It may be created and managed much faster.	It takes much time to create and handle.
Examples	Some instances of user-level threads are Java threads and POSIX threads.	Some instances of Kernel-level threads are Windows and Solaris.
Operating System	Any OS may support it.	The specific OS may support it.

c)