**OS Questions Answered**

1. **What are differences between a process and a thread? Describe some**

**benefits of using threads?**

|  |  |
| --- | --- |
| **Process** | **Threads** |
| Program in execution is known as process | Threads are light-weighted process |
| Process has its PCB, which stores all information about a process | Threads within the same process share same resources |
| Communication in process is challenging | Threads within the same process communicate with each other |
| Processes are created and terminated independently | Terminating a process result in the termination of all its threads because of sharing resources |

1. **What is a Race Condition and how it can be avoided?**

A situation in which multiple threads or processes read or write the same shared resource and the final output depend on the relative timing of their execution.

We can solve the problem of race condition by semaphore where we ensure that only process can access or use the shared resource or critical section

1. **What is Critical Section, and what is Critical Section Problem?**

A section of code within a process that requires access to shared resources and no two parts of the program use this section of code at the same time because that can cause problems

Problem with the critical section is that when two or more processes use the critical section, we want to achieve mutual exclusion to prevent this problem

1. **Define Semaphore and discuss usages of some of its types.**

A semaphore is a synchronization tool used in computer science that helps manage access to shared resources among multiple processes or threads

**Counting semaphore:**

Counting semaphores have an integer value that can be any non-negative number. allowing multiple processes or threads to access the resources as long as there are enough resources available.

**Binary Semaphore:**

Binary semaphores, also known as mutex semaphores, have a value that can only be 0 or 1, ensuring that only one process or thread can access a shared resource at a time.

**Mutex (Mutual Exclusion Semaphore):**

A special case of a binary semaphore used for mutual exclusion, ensuring that only one process can access a critical section at a time.

1. **Differentiate between Logical and Physical Address Space**

|  |  |
| --- | --- |
| **Logical Address** | **Physical Address** |
| When a program is complied, it’s not known where the program will be stored in main memory for this, we assign logical address | The physical address space is the actual locations in the main memory |
| It is the virtual address | It is the real address |
| It can be larger than the main memory | It cannot be larger than the main memory |
| Later translate logical address into physical address at the time of execution | We don’t need to translate the physical address |

1. **Discuss Swapping and discuss in detail the issues related with it.**

The operating system use swapping technique and decides which pages to swap in and out of physical memory based on algorithms like least recently used (LRU) or least frequently used (LFU) for the maximum utilization of the memory

Issues:

* Thrashing: Swapping out a process just before that piece is needed
* Principle of locality
* Fragmentation

1. **Explain the difference between internal and external fragmentation, by**

**what means they can be avoided?**

Internal fragmentation occurs when actual size of the process is smaller than the allocated block whereas external fragmentation is referred to as memory space is enough to store the process but it is not contiguous

Internal fragmentation problem can be solved by dynamic partitioning whereas external fragmentation problem can be solved by paging or compaction

1. **Write a note on Virtual Memory.**

virtual memory allows a process to be executed even if its size is larger than the available main memory (RAM).

* Process is divided into small pieces called pages
* Main memory is divided into small portion called frames
* In virtual memory, we do not need to load all pages to the main memory, we just need to load some of the most important pages which help initially in executing the process.
* In virtual memory, we fit page table into frame in main memory
* Swapping: The operating system decides which pages to swap in and out of physical memory based on algorithms like least recently used (LRU) or least frequently used (LFU).
* If a particular page is not in the main memory which is necessary to execute a process. This condition is called page fault
* The operating system handles page faults by swapping the required page from secondary storage (e.g., a hard disk) into main memory.
* Thrashing: Swapping out a process just before that piece is needed
* Principle of locality: (Studied already, it’s just to make prediction which portion we need in future
* DMA 🡪 allows the peripherals devices e.g., hard disk to access the main memory.
  + In virtual memory, when a page fault occurs and the operating system needs to bring a page from secondary storage (e.g., a hard disk) into main memory, DMA can be involved in the actual data transfer part of this operation, ensuring that the process is executed efficiently

1. **Write a note on Segmentation**

* A program can be subdivided into segments
* It is not required that all segments of all programs would be of same length
* Address consists of two parts i.e segment number and an offset

Logical Address:

* The logical address is 16 bits.
* Let's assume that the first 4 bits are used for the segment number, and the remaining 12 bits are for the offset.

Logical Address Example:

* Logical address in binary form: 0001001011110000.

Segment number: 0001 (Segment 1).

Offset: 001011110000.

Segment Table Lookup:

* We look up the segment table using the segment number (0001). Suppose the segment table indicates that Segment 1 starts at physical address 0010000000100000.

Physical Address Construction:

* We have the base address for Segment 1, which is 0010000000100000.
* The offset part of the logical address is 001011110000.
* To construct the physical address, we add the base address to the offset:
* Base address: 0010000000100000
* Offset: 001011110000
* Adding these together, we get the physical address: 0010001100010000.

1. **Write a note on Paging.**

* Processes divided into small parts called ‘pages’
* Main memory divided into small parts called ‘frames’
* We divide in a way that page size must be equal to the frame size
* **How paging eliminates the external fragmentation?**
  + Page size is equals to frame size because we put that pages into frames, so there won’t be any kind of external fragmentation exist.
* Now, we need of a data structure known as ‘Page table’, it is used for address translation (virtual address to physical address)
* The page table shows the frame location for each page of the process
* The page table is used *by the memory management hardware* to convert the logical address to an absolute (or physical) address during execution.

Conversion of logical address into physical address using paging:

Logical address further divided into two parts:

* Page number
* Offset

1. **﻿﻿﻿Define Deadlock.**

A situation in which two or more processes are unable to proceed because each is waiting for one of the others to do something.

**Section C**

**C1) Provide definitions of any four of the following terms: (4 Marks)**

1. **starvation**

Starvation is when a computer program or process doesn't get the resources it needs to do its job because other programs keep using those resources, leaving it waiting and not able to make progress.

1. **convoy effect**

when a long process arrives and shorter processes are queued behind it. The shorter processes have to wait for the long process to complete, leading to increased waiting times and inefficiency.

1. **process**

A program in execution is called process or When a Program’s executable file loaded into memory is referred to as a process.

1. **process state**

The process state refers to the current condition or status of a process in an operating system

* Ready
* Running
* Blocked

**t) time sharing**

In a time-sharing system, the CPU time is shared among the processes. For example, instructions of each process may be executed before moving to the next. The advantage of this approach is that in multiprogramming, we may have to wait for one task to complete, but in time-sharing, each task is executed for a minimum time before switching to another, ensuring that all tasks get their fair share of CPU time.

**d) multiprogramming**

In multiple programming, multiple processes or programs are loaded into main memory. The main goal for multi-programming is to maximize the utilization of the CPU.

**C2) Can a process make a transition from the ready state to the blocked state? If Yes Why, and if No Why?**

Yes, a process can transmit from ready to block state, if a process waiting for an event to be completed. For example, a process is waiting for user input, for time being this process will go to the blocked state.

**C3) What is the function of the ready queue?**

In operating system, ready queue is a data structure and this holds all the processes or threads that are ready to be executed by the CPU, but are currently waiting for their turn to do so

**C4) In round robin scheduling, new processes are placed at the end of the queue, rather than at the beginning, Suggest a reason for this.**

In Round Robin scheduling, each process is allocated a time quantum. For example, if the time quantum for a process expires but its CPU burst is not completed, the process should be placed at the end of the queue. This allows for the reassignment of a new time quantum, enabling the process to fully execute its CPU burst in later turns

**C5) Describe the two general roles of an operating system, and elaborate why these roles are important.**

**Resource Management:**

* The OS allocates CPU time to different processes, ensuring that each gets a fair share of the CPU's resources. It also handles scheduling, prioritizing tasks, and managing interruptions.

**User Interface:**

* An operating system provides a user interface (UI) that allows users to interact with the computer and its various applications
* When a user uses computer, user worry about what computers can do rather than the internal working of the computer.

**C6) Explain what interrupts are and briefly describe how the operating system handles them?**

An interrupt is a signal that tells the computer to stop the execution of the current tasks and pay attention to something really important.

For handling:

* When an interrupt occurs, the processor stops the execution of the process and saves it PCB. OS use interrupt handler routine to resolve interrupt, if there are multiple interrupts, we can use priority scheme

**C7) Describe how system calls work?**

System calls work by allowing a user-space program to request a service from the kernel of the operating system. This is done by executing a special instruction that traps to the kernel, which then switches to kernel mode and executes the requested service. When the service is complete, the kernel returns control back to the user-space program.

**C8) What is a process? What are attributes of a process?**

Program in execution is called process.

Attributes of process:

* identifier
* State
* Priority
* Program counter
* Memory pointers
* Context data
* I/O status information
* Accounting information

**C9) Give two examples of when a process might go directly from the state 'running' to the state "ready.**

**Interrupt Handling**:

When an interrupt occurs, such as a hardware interrupt (e.g., a timer interrupt or an I/O completion interrupt), the currently running process is interrupted and moves from the "running" state to the "ready" state

**Preemptive Scheduling:**

In pre-emptive scheduling, each running process gets a turn to use the CPU for a specific time. This time is like a "turn" that the process can use. If a process doesn't finish its work in its turn and doesn't willingly give up the CPU, the operating system steps in and moves the process to a "waiting for another turn" state, which is called the "ready" state.

**C10) Both Windows and UNIX boost the priority of I/O bound processes. Explain why this is a good Idea. Can this lead to starvation of CPU-bound processes?**

* Improved responsiveness: I/O-bound processes spend a significant amount of time waiting for I/O operations to complete
* Reduced latency: I/O-bound processes often handle user interactions, such as keyboard or mouse input, network requests, or graphical updates
* Efficient use of resources: CPU-bound processes are typically more resource-intensive, continuously consuming CPU cycles

**C11) Multi-programming (or multi-tasking) enables more than a single process to apparently execute simultaneously. How is this achieved on a uniprocessor?**

Multiprogramming on a uniprocessor, also known as multitasking, is achieved through a technique called time-sharing. The operating system (OS) divides the processor time into small slices called time quanta and allocates each quantum to a different process These can be implemented using algorithms such as:

1)First come - First serve

2)Round Robin

3)Priority etc

**Q1. Advantages of Multi threading**

1. **Responsiveness** – may allow continued execution if part of process is blocked, especially important for user interfaces
2. **Resource Sharing** – threads share resources of process, easier than shared memory or message passing
3. **Economy** – cheaper than process creation, thread switching lower overhead than context switching
4. **Scalability** – process can take advantage of multicore architectures

**Q2. Advantages of interrupt**

* **Improves the efficiency of CPU**.
* **Reduced latency:** Interrupts allow the CPU to respond to important events immediately, without having to wait for the next polling cycle. This is especially important for real-time systems, where even a small delay can have serious consequences.
* **Improved concurrency:** Interrupts allow the CPU to switch between multiple tasks quickly and efficiently. This is essential for multi-user and multitasking operating systems.
* **Simplified I/O programming:** Interrupts allow I/O devices to communicate directly with the CPU, without the need for complex polling loops. This simplifies I/O programming and makes it more efficient.

**Q3.** **Can a process transmit from ready to block if yes why and if not why?**

In an operating system, a process can transition from the ready state to the blocked state if it is waiting for an event to occur, such as I/O completion or a signal from another process. When a process is blocked, it is not eligible for CPU time and is removed from the ready queue.

On the other hand, a process cannot transition from the ready state to the blocked state if it is already executing on the CPU. This is because the running state is the only state that can transition to the blocked state.

**Q4.** **What is process and what are the attributes of PCB**

**Process** – a program in execution; process execution must progress in sequential fashion

* Program is passive entity stored on disk (executable file), process is active
* Program becomes process when executable file loaded into memory

**Attributes of PCB**

1. **Process ID**: A unique identification number assigned to each process by the operating system.
2. **Process State**: The current state of the process, such as running, ready, or blocked.
3. **Program Counter**: The address of the next instruction to be executed by the process.
4. **CPU Registers**: The values of the CPU registers for the process.
5. **Memory Management Information**: Information about the memory allocated to the process, such as the base address and the size of the process’s address space.
6. **I/O Status Information**: Information about the I/O devices allocated to the process, such as open files and pending I/O requests.
7. **Priority**: The priority of the process, which determines its relative importance in the scheduling of processes.

**Q5.** **What are the basic unit of computer (describe)**

The basic units of a computer are the input unit, the central processing unit (CPU), the memory unit, and the output unit.

* 1. **Input unit**

The input unit allows the user to enter data and instructions into the computer. Common input devices include the keyboard, mouse, scanner, and microphone.

* 1. **Central processing unit (CPU)**

The CPU is the brain of the computer. It is responsible for fetching, decoding, and executing instructions. The CPU also performs arithmetic and logical operations on data.

* 1. **Memory unit**

The memory unit stores data and instructions that the CPU needs to access. There are two main types of memory: primary memory and secondary memory. Primary memory, also known as RAM, is fast but volatile, meaning that data is lost when the computer is turned off. Secondary memory, such as hard drives and optical drives, is slower but non-volatile, meaning that data is retained even when the computer is turned off.

* 1. **Output unit**

The output unit displays the results of the computer's processing to the user. Common output devices include the monitor, printer, and speakers.

**Q6.** **What are the general rule of operating system and define two**

1. **Manage hardware resources:** The operating system is responsible for managing the computer's hardware resources, such as the CPU, memory, and storage devices. This includes allocating resources to processes, scheduling processes to run on the CPU, and handling interrupts.
2. **Provide a common interface for applications:**The operating system provides a common interface for applications to use, so that they do not have to worry about the details of the underlying hardware. This includes providing services such as file I/O, networking, and security.
3. **Protect resources:** The operating system is responsible for protecting resources from unauthorized access. This includes isolating processes from each other and protecting the system from malicious software.
4. **Ability to Evolve:** The operating system should be extensible, so that new features can be added without having to modify the core of the operating system.
5. **Efficiency:** The operating system should use the system's resources efficiently. This means minimizing the amount of CPU time and memory used by the operating system itself.

**Q7.** **What is interrupt. Explain.**

An interrupt is a signal to the processor that an event has occurred and requires immediate attention. Interrupts can be generated by hardware devices, software programs, or the processor itself.

When an interrupt occurs, the processor saves its current state and transfers control to an interrupt handler. The interrupt handler is a piece of code that is responsible for handling the specific event that caused the interrupt. Once the interrupt handler has completed its task, the processor restores its previous state and resumes executing the program that was interrupted.

There are two types of interrupts:

* **Hardware interrupts**: These are generated by external devices such as keyboards, mice, and disk drives. Hardware interrupts are further classified into two types:
  + **Maskable interrupts**: These can be delayed if a higher priority interrupt is being serviced.
  + **Non-maskable interrupts**: These cannot be delayed and must be serviced immediately.
* **Software interrupts**: These are generated by software programs to request services from the operating system, such as input/output operations or memory allocation.

**Classes of Interrupts**

1. A **program interrupt** is a signal that is generated by either hardware or software to temporarily halt the execution of a program and transfer control to another program, called an interrupt handler or interrupt service routine (ISR).
2. **Timer:** Generated by a timer within the processor. This allows the operating system to perform certain functions on a regular basis.
3. **I/O:** Generated by an I/O controller, to signal normal completion of an operation or to signal a variety of error conditions.
4. **Hardware failure:** Generated by a failure, such as power failure or memory parity error.

**Q9.** **Disadvantage of ULT**

**Disadvantages of ULTs**

1. when a ULT executes a system call, not only is that thread blocked, but all of the threads within the process are blocked
2. In a pure ULT strategy, a multithreaded application cannot take advantage of multiprocessing.

**Q10.** **2 reasons/examples when a process transmit from running to ready**

1. **Interrupts**: When a process is running, it can be interrupted by an interrupt request from a device or a software interrupt.
2. **Preemption**: When a process is running, it can be preempted by the operating system if a higher-priority process becomes available.

**Q11.** **What are system calls and how do they work?**

A system call is a software mechanism that allows a user-mode program to request a service from the operating system kernel. System calls are the only way for a user-mode program to access the resources and services provided by the kernel.

To make a system call, a user-mode program executes a special instruction that causes the processor to switch to kernel mode. The processor then saves the state of the user-mode program and jumps to the kernel code that implements the system call.

Once the kernel code has completed the system call, it restores the state of the user-mode program and returns to user mode.

System calls are used for a variety of purposes, including:

* File I/O
* Process management
* Memory management
* Device management
* Networking

Here are some examples of system calls:

* open() - Opens a file
* read() - Reads data from a file
* write() - Writes data to a file
* close() - Closes a file
* fork() - Creates a new process
* execve() - Executes a program
* exit() - Terminates a process
* malloc() - Allocates memory from the heap
* free() - Frees memory back to the heap
* bind() - Binds a socket to a port
* listen() - Listens for incoming connections on a socket
* accept() - Accepts an incoming connection on a socket
* send() - Sends data on a socket
* recv() - Receives data on a socket

This is how a system call works

1. The user-mode program executes a special instruction that causes the processor to switch to kernel mode. This instruction is typically called the "syscall" instruction.
2. The processor saves the state of the user-mode program, including the program counter, registers, and stack pointer.
3. The processor jumps to the kernel code that implements the system call. The kernel code is typically located in a special area of memory called the "kernel space."
4. The kernel code executes the system call and performs the requested operation. This may involve accessing system resources, such as files, devices, or memory.
5. Once the kernel code has completed the system call, it restores the state of the user-mode program and returns to user mode.

**Q12. what is Context switching?**

Context switching is the process of storing the state of a process or thread, so that it can be restored and resume execution at a later point, and then restoring a different, previously saved, state. This allows multiple processes or threads to share a single central processing unit (CPU), and is an essential feature of a multiprogramming or multitasking operating system.

When a context switch occurs, the operating system saves the following information for the current process or thread:

* The program counter, which contains the address of the next instruction to be executed
* The processor registers, which contain temporary data used by the CPU
* The stack, which contains the current call stack and local variables

The operating system then loads the corresponding information for the new process or thread.

Context switching can be triggered by a number of events, including:

* A timer interrupt, which signals that the current process or thread has used up its allotted CPU time
* An I/O interrupt, which signals that an I/O operation has completed
* A system call, which is a request from the user-level application to the operating system kernel
* A user-level thread switch, which is initiated by the user-level application itself

Context switching is an essential part of modern operating systems. It allows multiple processes or threads to run concurrently, without interfering with each other. This makes it possible to run multiple applications at the same time, and to respond to events quickly and efficiently.

**Q14. what are reason for process termination?**

* normal completion
* Time limit exceeded
* arithmetic error ⇒ The process tries prohibited computation. for eg division by zero
* I/O failure

**Q15.** **how OS handles a process?**

The operating system handles processes in the following way:

1. **Process creation:** The operating system creates a new process when a user starts a program or when another process creates a child process.
2. **Process scheduling:** The operating system decides which process should run next on the CPU. This is done using a scheduling algorithm.
3. **Process execution:** The operating system loads the process's code and data into memory and gives the process control of the CPU.
4. **Process synchronization:** The operating system ensures that multiple processes can safely access shared resources.
5. **Process termination:** The operating system terminates a process when it has finished executing or when it is killed by the user or the operating system itself.

**Q16.****Difference between preemptive and nonpremptive?**

**Preemptive Non-preemptive**

|  |  |
| --- | --- |
| * **Interrupts running process** * **Provides better performance** * **More complex** * **More overhead** * **Used for real time systems, and interactive systems** | * **Doesn’t interrupt running process** * **Doesn’t provide better performance** * **Less complex** * **Less overhead** * **Used for batch processing systems** |

**Q17.** **Relationships between threads and processes?**

|  |  |  |
| --- | --- | --- |
| Feature | Thread | Process |
| Memory space | Shares memory space with other threads | Has its own memory space |
| Resources | Shares resources with other threads (e.g., open files, file descriptors) | Has its own resources |
| Code | Shares code with other threads | Has its own code |
| Execution | Can run concurrently with other threads | Runs independently of other processes |

**Q19.** **Give two disadvantages of ULTs over KLTs?**

There are two main disadvantages of ULTs over KLTs:

* System calls are blocking: When a ULT executes a system call, not only is that thread blocked, but also all of the threads within the process. This can lead to performance problems and deadlocks.
* Multiprocessing cannot be taken advantage of: ULTs are scheduled by the application, not the kernel. This means that ULTs cannot be scheduled across multiple processors, which can limit performance.

**Q20.** **What are the similarlies between threads and processes**

* Both threads and processes can execute code independently. This means that they can both be running different code at the same time.
* Both threads and processes have their own stack and program counter. This allows threads and processes to switch between each other quickly and efficiently.
* Both threads and processes can be created and destroyed dynamically. This means that they can be created and destroyed at any time during the execution of the program.
* Both threads and processes can be used to improve the performance of an application. For example, a web browser might use multiple threads to load different parts of a web page at the same time. This can improve the performance of the web browser by allowing it to load the web page more quickly.