OS QB Solution

Tnmy H.

1. Describe the action taken by a kernel to context switch processes.

Context Switching by the Kernel:
Context switching is a fundamental operation performed by the operating system's kernel to manage multiple processes running concurrently on a system with a single CPU core or multiple cores. When a context switch occurs, the kernel switches the CPU from executing the instructions of one process to another. Here's a step-by-step description of the action taken by the kernel during a context switch:

- **Saving the Current Process State**: The kernel saves the current state of the running process into its Process Control Block (PCB). This includes the contents of CPU registers, the program counter (PC) indicating the next instruction to execute, and other relevant information such as process-specific data.
- **Selecting a New Process:** The kernel selects a new process to run from the pool of ready processes. This selection can be based on scheduling algorithms like Round Robin, Priority Scheduling, or others.
- **Loading the New Process State:** Once a new process is chosen, the kernel loads its saved state from its PCB into the CPU and memory. This involves restoring the contents of CPU registers and the program counter with the values stored in the PCB of the selected process.
- **Updating Data Structures**: The kernel updates various data structures, such as the process scheduler's data, to reflect the change in the currently running process and any changes in the ready queue.
- **Handing Control to the New Process**: Finally, the kernel transfers control of the CPU to the newly selected process, allowing it to resume execution from where it left off.

2. What is inter-process communication? Are the function callback and inter-process communication the same?

Inter-process communication (IPC) refers to the mechanisms and techniques used by processes to exchange data and information with each other. In a multitasking operating system environment where multiple processes are running concurrently, IPC allows processes to collaborate, synchronize, and share resources. IPC enables processes to communicate and coordinate their activities, facilitating various tasks such as synchronization, data sharing, and coordination of computation.

While both IPC and function callbacks involve communication between different parts of a system, they operate at different levels of abstraction and serve different purposes. IPC facilitates communication between separate processes running concurrently on a system, enabling collaboration and resource sharing across process boundaries. Function callbacks, on the other hand, facilitate modular and event-driven programming within a single process, allowing functions to be invoked dynamically in response to specific events or conditions.

IPC enables communication between processes, while function callbacks facilitate dynamic function invocation within a single process. While they both involve communication, they operate at different levels of granularity and serve distinct purposes in system and application design.

3. What is a process? Describe process state and its transition.

A. A process is an instance of a program in execution on a computer system.
B. It comprises the program code, associated data, and resources (such as memory, CPU time, and I/O devices) allocated to it.
C. Process State:
    a. New: The process is being created.
    b. Ready: The process is waiting to be assigned to a processor for execution.
    c. Running: The process is currently being executed by a processor.
    d. Blocked (or Waiting): The process is waiting for some event (such as I/O completion) to occur.
    e. Terminated: The process has finished its execution and has been terminated.
D. Process Transition:
    a. New to Ready: Occurs when the process is created and ready to execute but waiting for CPU time.
    b. Ready to Running: Happens when the scheduler selects the process for execution.
    c. Running to Blocked: Occurs when the process initiates an I/O operation or another event that requires waiting.
    d. Blocked to Ready: Happens when the event the process was waiting for occurs, and it is ready to execute again.
    e. Running to Terminated: Occurs when the process completes its execution.
    f. Ready to Terminated: Can occur if the process is removed from the system without being executed.
E. Processes transition between states based on events, scheduler decisions, and resource availability.

4. Explain the layout of system. How do we save the context of the process?

The layout of a system refers to how its components are organized and interconnected to perform various functions and provide services to users and applications.

1. In the context of an operating system, the layout typically involves the organization of memory, processes, devices, and other system resources.
2. Saving the context of a process is crucial for allowing the system to switch between processes efficiently and resume execution accurately.
3. Context refers to the state of a process, including the values of its CPU registers, program counter, stack pointer, and other relevant data.
4. When a context switch occurs, the current state of the running process is saved, and the state of the next process to be executed is restored.
5. This process is usually managed by the kernel, which stores the context of each process in its Process Control Block (PCB) or a similar data structure.
6. The context switch involves saving the current context to the PCB of the currently running process and loading the context from the PCB of the next process to be executed.
7. The kernel typically performs context switches in response to events such as a timer interrupt, I/O completion, or a scheduling decision.
8. Context switching ensures that processes can share the CPU efficiently, allowing for multitasking and concurrent execution of multiple processes on a single processor system.

5. What is system program describe is categories with example

A system program is a type of software that provides essential functions and services to the operating system and users. These programs facilitate the management, operation, and interaction with the underlying hardware and software components of a computer system. System programs typically run in privileged mode, allowing them to access system resources and perform tasks that regular user programs cannot.

System Utilities:

- Disk Management Utilities: Programs for managing disk storage, including partitioning, formatting, and file system maintenance. Examples include:
  - Disk Management in Windows.
  - fdisk and mkfs commands in Unix/Linux.
- File Management Utilities: Tools for organizing, manipulating, and accessing files and directories. Examples include:
  - File Explorer (Windows).
  - ls, cp, mv commands in Unix/Linux.
- System Configuration Utilities: Programs for configuring system settings and parameters. Examples include:
  - Control Panel (Windows).
  - System Preferences (macOS).
  - System Configuration Tool (sysconfig) in Unix/Linux.

System Service Programs:

- Printing Services: Programs responsible for managing printing operations and interacting with printers. Examples include:
  - Print Spooler service in Windows.
  - CUPS (Common Unix Printing System) in Unix/Linux.
- Network Services: Programs providing network-related services such as web serving, file sharing, and remote access. Examples include:
  - Apache HTTP Server for web serving.
  - OpenSSH for secure remote access.
- Security Services: Programs for implementing security policies and mechanisms, such as firewalls and antivirus software.

Device Drivers:

- Peripheral Drivers: Software components that allow the operating system to communicate with hardware devices such as printers, disk drives, and network adapters. Examples include:
  - Printer drivers for various printer models.
  - Disk drivers for hard disk and solid-state drives.
  - Network interface drivers for Ethernet and Wi-Fi adapters