Process Management - Exam Notes

1. Definition of Process Management

- **Process**: A process is a program in execution. It includes the program code, current activity (program counter), process stack, data section, and other OS resources.
- **Process Management**: It's the activity of managing the lifecycle of processes, including their creation, execution, and termination, and ensuring efficient CPU utilization.

2. Types of Processes

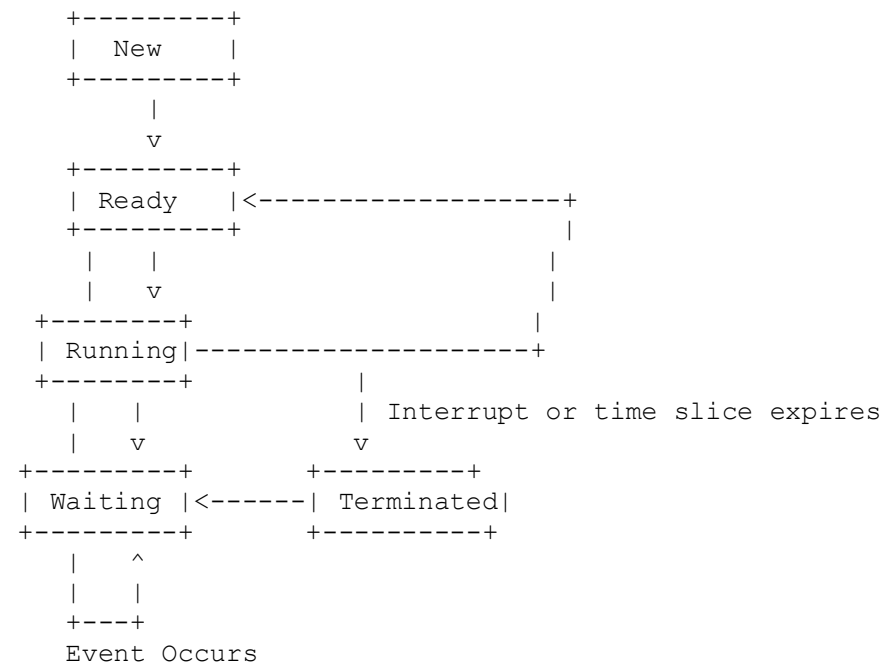
- **I/O-bound Process**: Spends more time performing I/O operations than computations.
- **CPU-bound Process**: Spends more time performing computations than I/O.
- **Foreground Process**: Interacts directly with the user.
- **Background Process**: Runs without user interaction.
- **System Process**: Executes OS functions.

3. Process States and Transitions

Process States

- **New**: Process is being created.
- **Ready**: Process is loaded into main memory, waiting to be assigned to CPU.
- **Running**: Process is currently executing on CPU.
- **Waiting (Blocked)**: Process is waiting for some event (e.g., I/O completion).
- **Terminated (Exit)**: Process has finished execution.

State Transition Diagram



- New → Ready: Process creation
- Ready → Running: Scheduler dispatch
- Running → Waiting: Waiting for I/O or event
- Waiting → Ready: Event completed
- Running → Ready: Interrupt or preemption
- Running → Terminated: Process finished or killed

4. Process Control Block (PCB)

- **Definition**: Data structure used by OS to store all information about a process.
- **Contents of PCB**:
  - Process ID (PID)
  - Process State
  - Program Counter (PC)
  - CPU Registers
  - CPU Scheduling Information (priority, pointers)
  - Memory Management Info (page tables, base/limit registers)
  - Accounting Info (CPU usage, time limits)
  - I/O Status (list of open files, devices)

5. Context Switching

- **Definition**: The process of storing the state of a running process and loading the state of another process so that the CPU can switch between processes.

- **Steps**:
  - Save the context of the current process into its PCB.
  - Load the context of the next scheduled process from its PCB.
- **Impact on Performance**:
  - Context switches are overhead and do not do useful work.
  - Frequent context switching reduces CPU efficiency.
  - Time taken for context switching depends on hardware and OS.

6. Process Scheduling

- The OS uses a **scheduler** to select which process runs next.
- **Goal**: Maximize CPU utilization, throughput, minimize waiting time, response time, and ensure fairness.

7. Types of Schedulers

- **Long-Term Scheduler (Job Scheduler)**
  - Decides which processes are admitted into the system for processing.
  - Controls the degree of multiprogramming.
  - Invoked infrequently.
- **Short-Term Scheduler (CPU Scheduler)**
  - Decides which ready process will get the CPU next.
  - Invoked very frequently (milliseconds).
  - Responsible for context switching.
- **Medium-Term Scheduler**
  - Temporarily removes processes from memory (swapping) to reduce degree of multiprogramming.
  - Balances CPU and memory resources.

8. Threads

- **Concept of Thread**: A thread is the smallest unit of CPU execution within a process.
- **Multithreading**: Running multiple threads within a single process to achieve parallelism and better resource utilization.
- **Benefits**: Improved responsiveness, resource sharing, and efficient CPU usage.

User-level vs Kernel-level Threads

Feature	User-level Threads	Kernel-level Threads
Management	Managed by user-level thread library	Managed by OS kernel
Context Switch	Fast, no kernel mode switch	Slower, involves kernel mode switch
Scheduling	Thread library schedules threads	Kernel schedules threads individually
Blocking	All threads block if one blocks	Only the blocked thread blocks
Portability	High (runs on any OS)	Dependent on OS

9. Scheduling Algorithms

Preemptive vs Non-preemptive Scheduling

- **Preemptive Scheduling**: CPU can be taken away from a process before it finishes (e.g., RR, Priority with preemption).
- **Non-preemptive Scheduling**: Once CPU is allocated, process runs to completion or waits for I/O (e.g., FCFS, SJF without preemption).

Common Scheduling Algorithms

Algorithm	Type	Description	Advantages	Disadvantages
FCFS (First Come First Serve)	Non-preemptive	Processes are scheduled in order of arrival.	Simple, easy to implement	Long waiting time, poor for short jobs
SJF (Shortest Job First)	Non-preemptive or Preemptive (SRTF)	Process with shortest next CPU burst runs first.	Optimal average waiting time	Difficult to predict burst time, may cause starvation
Round Robin (RR)	Preemptive	Each process gets a fixed time slice (quantum).	Fair, good for time-sharing systems	High context switch overhead, quantum choice critical
Priority Scheduling	Both	Each process assigned priority; highest priority runs first	Can be preemptive or non-preemptive	Risk of starvation of low priority processes

10. Process Scheduling in UNIX and Windows

- **UNIX Scheduling**
  - Uses priority-based preemptive scheduling.
  - Priorities can be dynamically adjusted based on process behavior (nice value).
  - Mixes time-sharing and batch processes.

- **Windows Scheduling**
  - Uses a multilevel feedback queue.
  - Threads have priority levels from 0 to 31.
  - Scheduler boosts priorities of threads that are interactive.
  - Preemptive with time slices varying based on priority.

Summary

Topic	Key Points
Process	Program in execution; needs resources
Process States	New, Ready, Running, Waiting, Terminated
PCB	Stores process info for context switching
Context Switching	Saving/restoring state, overhead involved
Scheduling	Long-term, medium-term, short-term schedulers
Threads	Lightweight processes, user vs kernel managed
Scheduling Algorithms	FCFS, SJF, RR, Priority, preemptive/non-preemptive
UNIX & Windows Scheduling	Priority-based and multilevel feedback queue respectively