UNIT-3

1. Process Concept -
   A process is a program in execution.
   Process state: New, Running, Waiting, Ready, Terminated

2. Process Control Block (PCB) - Task control block, contains piece of information.
   - Process state: new, ready, waiting, running, terminated
   - Process number
   - Program counter: address of the next instruction
   - Registers
   - Scheduling info: which scheduling is using
   - Memory info: page tables or segment tables
   - Accounting info: amount of CPU and real time used.
   - I/O status: allocated I/O devices, list of open files

3. Scheduling Criteria -
   - CPU utilization (range 0 to 100%) (MAXIMUM)
   - Throughput: No. of processes completed per unit time. (MAXIMUM)
   - Turnaround time: finish time - arrival time. (MINIMUM)
   - Waiting time = Turnaround time - Burst time. (MINIMUM)
   - Response time: First response time - arrival time. (MINIMUM)

4. Preemptive Scheduling -
   1. Running state to waiting state
   2. Running state to ready state
   3. Waiting state to ready state
   4. When a process terminates
Non-preemptive scheduling:
(1) Running state to waiting state
(2) When a process terminates

Scheduling algorithms:
(1) FCFS scheduling → non-preemptive
(2) Shortest job first scheduling (SJF) → preemptive or non-preemptive
(3) Priority scheduling → preemptive or non-preemptive
(4) Round Robin scheduling (RR) → preemptive

If time quantum is extremely small, the RR approach is called process sharing.

Multilevel queue scheduling:
- system process
- interactive process
- interactive editing process
- batch process
- student process

Multilevel feedback queue scheduling:
It gives highest priority to any process with a CPU burst of 8 milliseconds or less. It is defined by the following parameters:
(1) The number of queues
(2) Scheduling algorithm of each queue
(3) Method used to determine when to upgrade a process to a higher-priority queue
(4) Method used to determine when to demote a process to a lower-priority queue
(5) Method used to determine which queue a process will enter when that process needs service.
6. Algorithm Evaluation
   1. Maximizing CPU utilization under the constraint that the minimum response time is 1 second.
   2. Minimizing throughput such that turnaround time is (on average) linearly proportional to total execution time.

5. Deterministic Modeling
   - Queuing Models & Little's formula \( n = \lambda \times W \)

6. Implementation
   - code up the algorithm & put it in OS

7. Multiple Processor Scheduling
   - Asymmetry multiprogramming: scheduling decisions, I/O processing, and other system activities handled by a single processor, maintenance other processors execute only user code.
   - Symmetry multiprogramming: each processor is self-scheduling.
     - Processor affinity: a process has an affinity for the processor on which it is currently running.
     - Soft affinity: running on same processor, but not guaranteeing it will do.
     - Hard affinity: specify not to migrate to other processors.
   - Load balancing: evenly distributed across all processors.
     - Push migration: task periodically checks the load on each processor.
     - Pull migration: occurs when an idle processor pulls a waiting task from a busy processor.

8. Process Scheduling
   - ready queue
   - I/O
   - I/O queue
   - I/O request
   - time slice expired
   - child executes
   - fork a child
   - interrupt occurs
   - wait for an interrupt
Scheduling:
- Long-term scheduler or job scheduler - selects processes from the pool and loads them into memory for execution.
- Short-term scheduler or CPU scheduler - selects from among the processes that are ready to execute and allocates the CPU to one of them.
- Degree of multiprogramming - No. of processes in memory.
- Medium-term scheduler -
  - Ready queue
  - Partially executed and ready processes
  - Wait

Content switch - Switching the CPU to another process to perform a state save of the current process and a state set up of a different process.

IO Bound process - Spends more of its time doing IO.
CPU Bound process - Spends more of its time doing computation.

Real-time scheduling:
- Static priority scheduling and dynamic priority scheduling
  - Scheduling decision based on fixed parameters
  - Assigned before their execution
    - Example: highest priority scheduling
  - Change during system execution
    - Example: Earliest deadline scheduling

Operation on processes:
- Process creation - fork() 
- Child execution - exec() 
- Process termination - termination starts with child.

Threads - A process is a program that performs a single/multiple threads of execution. That means single/multiple tasks at a time.
12. Interprocess communication:
   - Processes: Information sharing, computation speedup, modularity, communication.
   - Two models: Shared memory and message passing.

   ![Diagram of shared memory and message passing models]

   - Message passing systems: E.g., chat programs.
   - Shared memory systems: E.g., producer-consumer.

13. Precedence Graphs:
   1. Preamble
   2. Fork and join constructions
   3. Semaphores

14. Critical Section Problem:
   - A segment of code in which the process may be changing common variables, updating a table, writing a file, and so on.
   - No two processes are executing in their critical sections at the same time.
   - Solution to critical section problem:
     1. Mutual Exclusion
     2. Progress
     3. Bounded Waiting

   ```
   do {
   entry_section
   critical_section
   exit_section
   remainder_section
   } while (true);
   ```

   General structure of a typical process.
Semaphore - (synchronization tool)

A semaphore $S$ is an integer variable is accessed only through two standard atomic functions: \texttt{wait()} & \texttt{signal()}. 

\begin{align*}
\text{wait}(S) \; \mathbf{?} \; \text{signal}(S) \; \mathbf{?} \\
\text{while} \; S \leq 0 \\
\; \text{// no-operation} \\
S \; \mathbf{- -} \\
\end{align*}

Mutual exclusion implementation with semaphore -

\texttt{do}

waiting (mutex);

\text{// critical section}

\texttt{signal (mutex)};

\text{// remainder section}

\texttt{while (TRUE)};

\text{Two types of semaphores - counting semaphore & binary semaphore}

also known as mutex lock.

Implementation problem - deadlock and starvation.

\textbf{Classical Problems of synchronization -}
(1) \textbf{Producer - Consumer Problem} - semaphore mutex, full, empty:

\textbf{Producer Process} -

\texttt{do}

\text{// produce an item in heap}

\texttt{wait (empty)};

\texttt{wait (mutex)};

\text{// add mutex to buffer}

\texttt{signal (mutex)};

\texttt{signal (full)};

\texttt{? while (TRUE)};

\textbf{Consumer Process} -

\texttt{do}

\text{// consume the item in heap}

\texttt{wait (full)};

\texttt{wait (mutex)};

\texttt{// remove an item from buffer}

\texttt{signal (mutex)};

\texttt{signal (empty)};

\texttt{? while (TRUE)};
(2) Reader-Writer Problem -

Reader Process -

```
semaphore muter, wrt; int readcount;
do {
    wait (muter);
    readcount++;
    if (readcount == 1)
        wait (wrt);
    signal (muter);
    if (reading is performed)
        wait (muter);
    readcount--;
    if (readcount == 0)
        signal (wrt);
    signal (muter);
} while (TRUE);
```

Writer Process -

```
do {
    wait (wrt);
} while (TRUE);
```

(3) Dining - Philosophers Problem -

```
Philosopher i -

```
```
semaphore chopstick [5];
do {
    wait (chopstick [i]);
    wait (chopstick [(i+1)%5]);
    // eat
    signal (chopstick [i]);
    signal (chopstick [(i+1)%5]);
    // think
    s while (TRUE);
```
```
Deadlock -

In a deadlock, processes never finish executing, and system resources are tied up, preventing other jobs for starting.

1. **Deadlock Characterization** -

   → Necessary conditions - four conditions hold simultaneously

   (1) Mutual exclusion
   (2) Hold and wait
   (3) No preemption
   (4) Circular wait

   → Resource allocation graph - directed graph

   ![Resource allocation graph](image)

   - Requested an owned resource (currently waiting, request edge)
   - Resource is allocated to process (assignment edge)

2. **Methods for handling deadlocks** -

   (1) Use a protocol to prevent or avoid deadlocks.
   (2) Allow the system to enter a deadlock state, detect it, and recover.
   (3) Ignore the problem altogether.

3. **Deadlock prevention** - By ensuring that at least one of these four necessary conditions for deadlock cannot hold, we can prevent occurrence of a deadlock.

4. **Deadlock Avoidance** -

   → Safe state - A system is in a safe state only if there exists a sequence.

   ![Safe state](image)

   - Safe
   - Unsafe
   - Deadlock

   - Safe state - A system is in a safe state only if there exists a sequence...
Resource allocation graph Algorithm:

\[ R_2 \]

claim edge (it may request resource in future)

Banker's Algorithm:

\[
\begin{align*}
\text{Need}[i,j] &= \text{Max}[i,j] - \text{Allocation}[i,j] \\
\text{Available}[i] &= \text{Total instance}[i] - \text{Need}[j].
\end{align*}
\]

Satisfy Algorithm:

1. Work = Available and Finish[i] = false
2. If (Finish[i] = false AND need[i] ≤ work)
   then
   \[ \text{Work} = \text{Work} + \text{Allocation} \]
   \[ \text{Finish}[i] = \text{true} \]
   \[ \text{repeat step 2}; \]
   else if no such i exists goto step 3.
3. If finish[i] = true for all i, then determine a safe state

Resource request algorithm:

If request[i] ≤ Need[i]
then if request[i] ≤ Available
then available = available - request;
Allocation = Allocation + request;
Need = Need - request
otherwise request not granted

Deadlock detection:

\[ n^2 \] operations needed to detect

- Single instance of each resource type: Make a wait for graph. If there is a cycle in waitfor graph, then deadlock occurs.
- Several instances of a resource type: Use Banker's algorithm. \[ m \times n^2 \] operations needed to detect.
Recovery from deadlock -
   → Process termination -
     → abort all deadlocked process
     → abort one process at a time until the deadlock cycle is eliminated

Resource Preemption -
   Vice-versa selecting a victim, Rollback, Stimulation