Utilization

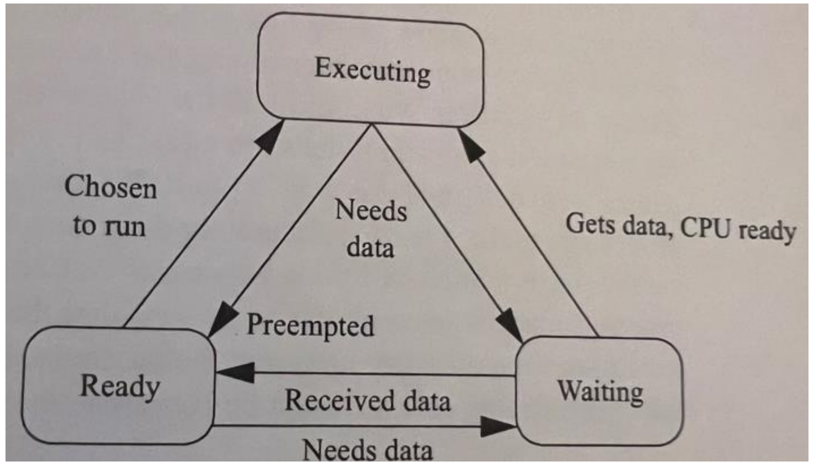
- CPU utilization:
  - Fraction of the CPU that is doing useful work (T).
  - Often calculated assuming no scheduling overhead.
- Utilization:
  - U = (CPU time for useful work)/ (total available CPU time)

$$= \frac{\sum_{t_1}^{t_2} T(t)}{t_2 - t_1}$$

= T/t

Process States

- A process can be in one of three states:
  - executing on the CPU;
  - ready to run;
  - waiting for data.



The scheduling problem

- Can we meet all deadlines?
  - Must be able to meet deadlines in all cases.
- Problem:
  - Multiple tasks in a task set running in parallel with different periods trying to complete at least one full cycle simultaneously
- Importance
  - Synchronization

Hyperperiod

- Hyperperiod: least common multiple (LCM) of the task periods.
- Must look at the hyperperiod schedule to find all task interactions.
- Hyperperiod can be very long if task periods are not chosen carefully.

Hyperperiod Example

- **Long** hyperperiod:
  - P1 7 ms.
  - P2 11 ms.
  - P3 15 ms.
  - **LCM = 1155 ms.**

- **Shorter** hyperperiod:
  - P1 8 ms.
  - P2 12 ms.
  - P3 16 ms.
  - **LCM = 48 ms.**

Hyperperiod Example

- Hyperperiod is the smallest period of time that encompasses all the individual periods of each task in the system. If the periods of the tasks are not integer multiples of each other, then the hyperperiod will be the least common multiple (LCM) of all the periods.

- To find the hyperperiod, we need to calculate the LCM of these three periods. The LCM is the smallest multiple that is common to all the periods. To do this, we can **factorize** each period into its **prime factors** and then find the **highest power** of each prime factor that appears in any of the periods. We then multiply these factors together to get the LCM.

- The prime factorization of each period is as follows:
- 7 ms = 7
- 11 ms = 11
- 15 ms = 3 x 5

Hyperperiod Example

- The **highest** power of each prime factor is:
    - 7 appears only once.
    - 11 appears only once.
    - 3 appears once in the period of task P3.
    - 5 appears once in the period of task P3.
- Therefore, the LCM of the periods is:
    - LCM = 7 x 11 x 3 x 5 = 1155 ms
- So the hyperperiod in this case is 1155 ms, which means that the pattern of execution of these tasks will repeat every 1155 ms

Hyperperiod Example

- we can factorize each period into its prime factors and then find the highest power of each prime factor that appears in any of the periods. We then multiply these factors together to get the LCM.
- The prime factorization of each period is as follows:
  - 8 ms = 2^3
  - 12 ms = 2^2 **x 3**
  - 16 ms = 2^4
- The highest power of each prime factor is:
  - 2 appears at least four times (3 for P1, 2 for P2, 4 for P3).
  - 3 appears once in the period of task P2.
- Therefore, the LCM of the periods is:
  - LCM = 2^4 x 3 = 48 ms
- So the hyperperiod in this case is 48 ms, which means that the pattern of execution of these tasks will repeat every 48 ms.

Simple processor feasibility example

- P1 period 1 ms, CPU time 0.1 ms.
- P2 period 1 ms, CPU time 0.2 ms.
- P3 period 5 ms, CPU time 0.3 ms.

$$\text{Task execution time} = \left\lceil \frac{LCM}{period} \right\rceil T_i$$

LCM=5ms	Period	CPU Time	Task execution time
P1	1 ms	0.1 ms	0.5 ms
P2	1 ms	0.2 ms	1 ms
P3	5 ms	0.3 ms	0.3 ms
Total tasks execution time: 1.8 ms			
CPU Utilization: 1.8/5=0.36			

Schedulability and overhead

- The scheduling process consumes CPU time.
    - Not all CPU time is available for processes.
- Scheduling overhead must be taken into account for exact schedule.
    - May be ignored if it is a small fraction of total execution time.

Running periodic processes

- Need code to control execution of processes.

- Simplest implementation: process = subroutine.

while loop implementation

- Simplest implementation has one loop.
    - No control over execution timing.

```
while (TRUE) {
  p1();
  p2();
}
```

Timed loop implementation

- Encapuslate set of all processes in a single function that implements the task set.
- Use timer to control execution of the task.
    - No control over timing of individual processes.

```
void pall(){
  p1();
  p2();
}
```

Multiple timers implementation

- Each task has its own function.
- Each task has its own timer.
  - May not have enough timers to implement all the rates.

```
void pA(){ /* rate A */
    p1();
    p3();
}
void B(){ /* rate B */
    p2();
    p4();
    p5();
}
```

Implementing processes

- All of these implementations are inadequate.
- Need better control over timing.
- Need a better mechanism than subroutines.

Operating systems

- The operating system controls resources:
  - who gets the CPU;
  - when I/O takes place;
  - how much memory is allocated.
- The most important resource is the CPU itself.
  - CPU access controlled by the scheduler.

OS Structure

- OS needs to keep track of:
  - process priorities;
  - scheduling state;
  - process activation record.
- Processes may be created:
  - statically before system starts;
  - dynamically during execution.

Preemptive real-time operating systems

- A preemptive real-time operating system solves the fundamental problems of a cooperative multitasking system.

- It executes processes based upon timing requirements provided by the system designer.

- The most reliable way to meet timing requirements accurately is to build a preemptive operating system and to use priorities to control what process runs at any given time.