Note Read chapter 12 of textbook!

Lecture #10 part 1

I Task Scheduling

- In a multitasking system, the main objective is to have the CPU utilized at all times (in practical systems 40% ÷ 90%)
- At each opportunity, OS asks: If N tasks are Ready, which one should I run?
- If we can show that the scheduler always can achieve deadlines, the system is "deterministically schedulable"!
- In real-time design, absolutely essential are:
  - reprotability
  - predictability
- **Scheduling types:**

1) - non-preemptive ≡ a new task that is Ready is scheduled only after the currently Running one switches to Waiting state (initiated by an I/O request) or after it terminates.

2) - preemptive ≡ otherwise.

- **Goals**

1) CPU utilization $U_{cpu} = 1 - \frac{idle}{period}$

2) Throughput: work units completed per unit of time. (# of processes)

3) Turnaround time: interval of time from submission of a task and its completion (waiting + execution + I/O time)

4) Waiting time: time spent waiting in the queues. Recall that when a task enters the system initially, it is put into an "entry queue". Then it is placed in "ready queue", when conditions are met.

5) Response Time: the time from submission to first response.

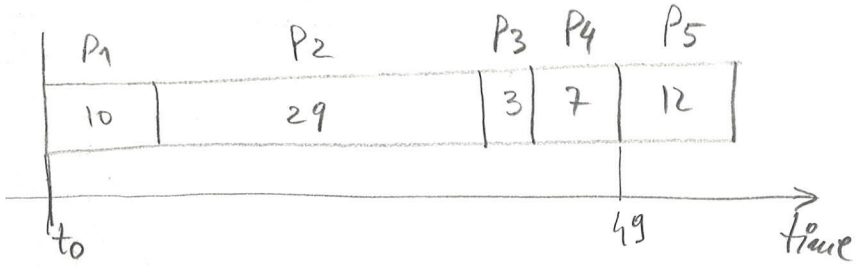
Scheduling Algorithms in Time-Shared Systems

(1) First-come First-Served

- when process enters the system, it is added to a (FIFO) queue.
- when the CPU becomes free, it is allocated to the process at the head of the queue.
- its non-preemptive

Example:

process	(Burst) Time
P1	10
P2	29
P3	3
P4	7
P5	12



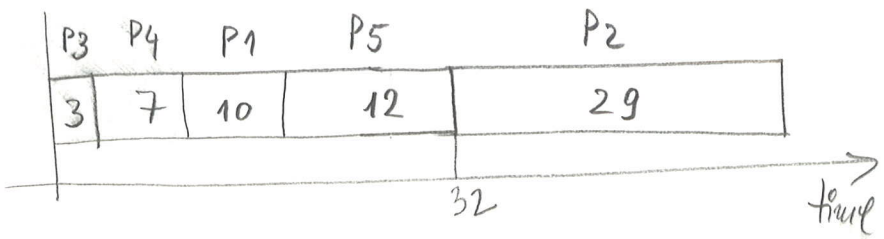
Average wait time:

P1	0
P2	10
P3	32
P4	42
P5	49

28

(2) Shortest Job first

- each task has associated with it an estimate of how much time will need to complete its execution once given the CPU
- this algorithm can be { – preemptive (Running task may be interrupted by shorter task)
  – non-preemptive



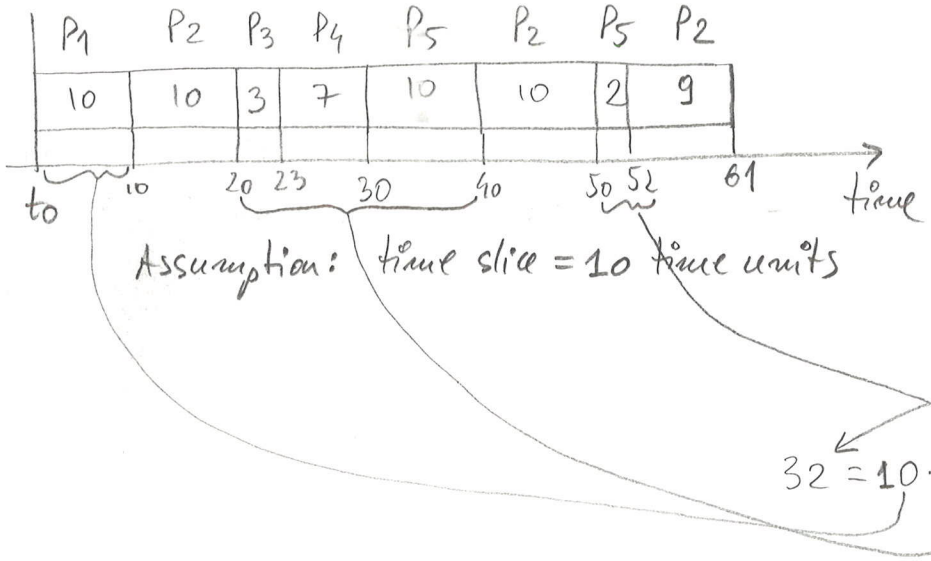
Average Wait Time

P3	0
P4	3
P1	10
P5	30
P2	32

13

(3) Round Robin

- designed especially for time shared systems.
- similar to first-come first-served, with preemption added to switch between processes.
- time quantum (or slice) ≡ time "unit" for which CPU is allocated to tasks or processes.
- Ready queue is treated as a circular queue. Scheduler walks the queue, allocating CPU to each task for one time slice!
- If process-completes in less than a slice time ⇒ it releases CPU
  - does not complete within a slice time ⇒ it's interrupted and placed at the end of queue!



Average Waiting Time

P1	0
P2	32
P3	20
P4	23
P5	40
	23

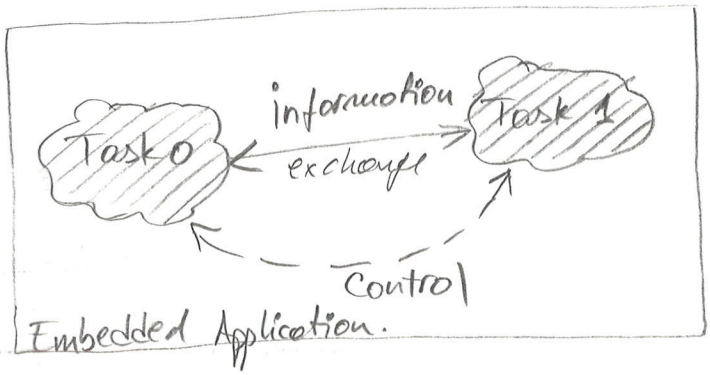
32 = 10 + 20 + 2

Note: if the evaluation criterion is "average waiting time", then shortest-job-first should be selected as the scheduling algorithm in the example above!

II Tasks and Threads Communication

- Resource sharing and intertask communication and synchronization must take place in a robust, safe, and reliable manner!

- The model for interprocess communication and synchronization:
  1. The information = data or signals being moved ← moved via shared variables or messages!
  2. Place (or places) from which (data) information is moved to/from (variables or pointer variables w/ memory addresses)
  3. Control and synchronization of actions and the movement of information (techniques based on flags or status bits)

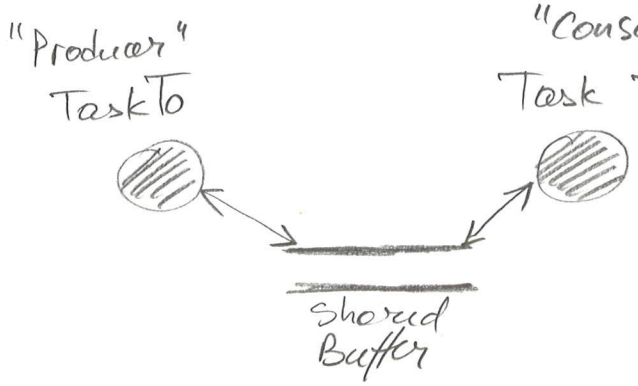


: data flow and control diagram

1.a Shared Variables

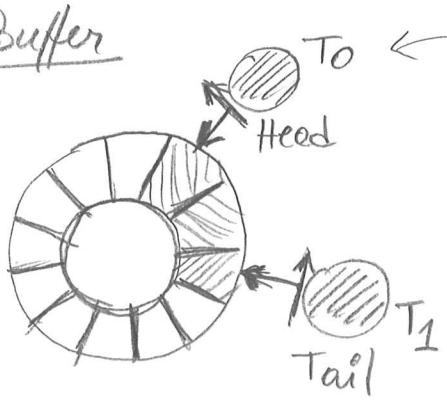
→ Global Variables
- advantage: do not need to be copied on the stack during a context switch!

→ Shared Buffer = an exchange technique in which two processes share a common set of memory locations.



issue: if tasks are running at different speeds, there is the potential for overrun or underrun!
=> need appropriate buffer sizing!

→ Ring Buffer



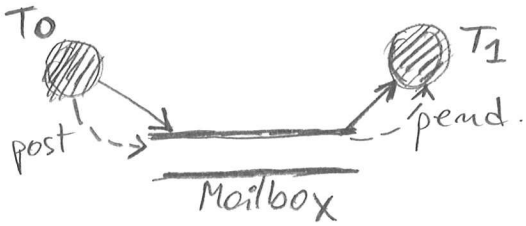
To ← Adds data to the Buffer

T1 ← Removes data from Buffer.

issue: underflow, overflow.

→ Mailbox
(similar to queue)
- supports two operations { 1) write (or post) 2) read (or pend) : during which a flag is raised to indicate data is available.

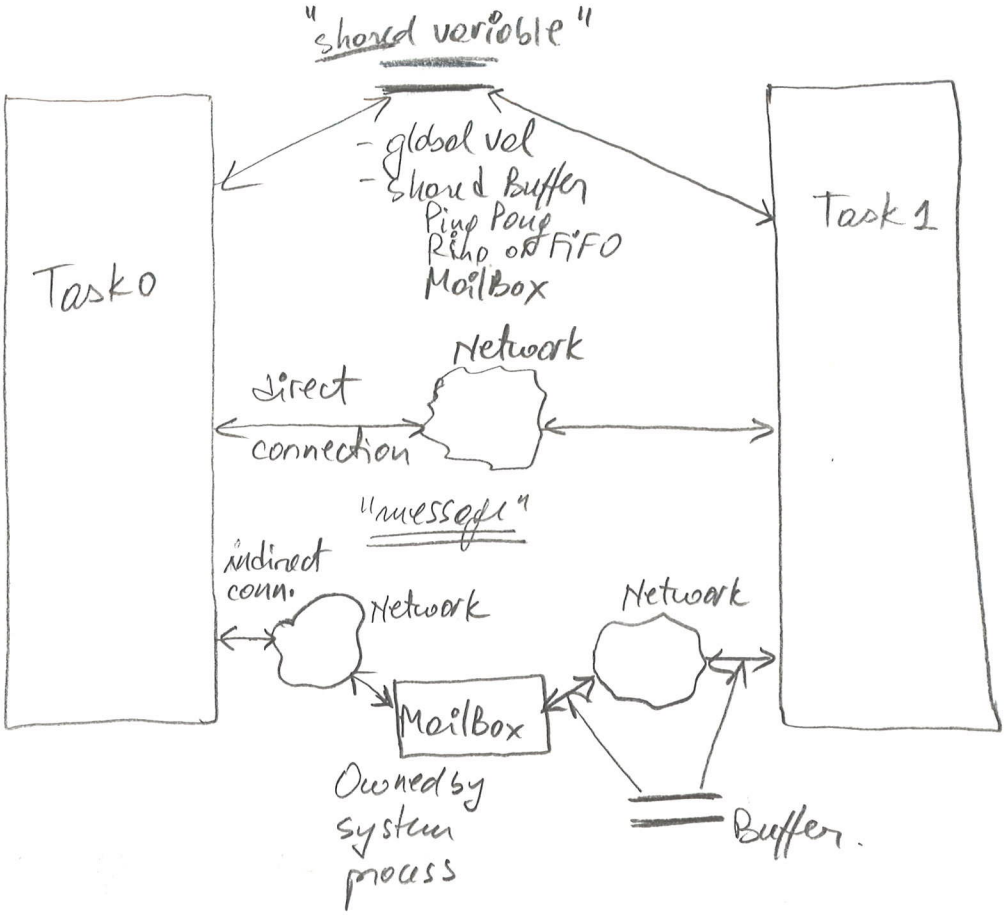
- Data flow and control diagram:



1.6 Messages

- utilized especially in distributed applications.
- supports 2 operations: { 1) send 2) receive
- messages can have fixed or variable sizes!
- messages can be moved from one place to another directly or indirectly.

Summary of intertask communication:



III Task Cooperation, Synchronization, and Sharing

- Many times, in a multitasking system, processes need to cooperate/synchronize as they execute the application!

- A "critical section" is a resource that several tasks may share such as an I/O port or segment of memory in which they are reading and writing common variables.

- We must ensure mutually exclusive access to these resources!

- Any solution to the critical section problem must satisfy the requirements:
  1) ensure mutual exclusion.
  2) prevent deadlock
  3) ensure progress
  4) ensure bounded waiting