|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Time | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 |
| FCFS | A | A | A | B | B | B | B | B | C | C | D | D | D | D | D | E | E | E | E | E |
| RR 1 | A | B | A | B | C | A | B | C | B | B | D | D | D | E | D | E | D | E | E | E |
| RR 4 | A | A | A | B | B | B | B | C | C | B | D | D | D | D | E | E | E | E | D | E |

RTX scheduling cont'd

Round Robin – all tasks have the same priority

Cooperative Multitasking (MT) – type of MT where the process currently controlling the CPU must offer control to other processes – i.e. they must ALL 'cooperate' in order for the application to work (i.e. pass control vs relying on continuous interrupts).  
- Typically uses os\_tsk\_pass() along with signals and waits

Preemptive MT – Scheduling that forces application processes to share CPU whether they want to or not.  
- Each task has a different priority, task runs until preempted by higher priority (or blocked by OS)  
- RTX kernel controls scheduling & saves states.

NON-Preemptive – tasks have different priority & run until completion. Next highest priority  
- Task then executes etc. Uses same protocol as preemptive  
- If task is time consuming, may use yield() instead to pass control to equal priority task.

TASK (&Thread) communication  
Model for interprocess communcation & synchro:

1. INFORMATION – data being moved (via shared vars or messages) to other tasks
   1. Types of shared vars in multitasking:
      1. Global Variables – don't need to be copied to tasks
      2. Shared Buffer – exchange technique where 2 processes share a common set of memory location. (PROBLEM – underrun & overrun)
      3. MAILBOX (queue) – supports 2 operations: A flag is raised to indicate available data
         1. WRITE (post)
         2. Read (pend)
   2. Messages – directly sent from 1 task to another using send() & recieve()
2. PLACE – where data is moved to/from (var and/or pointers)
3. CONTROL & SYNCHRO – movement & actions of data (ie using flags/status bits etc.)
   1. Mutexes & semaphores used to protect critical sections/resources shared amongst tasks. (i.e. I/O, mem segments, shared variables
   2. Implemented as boolean vars, accessed by 2 basic atomic operations [wait and signal flag)

Control

Task 1

Task 0

<= Data exchange =>