Real-Time Operating Systems

Mehmet Orhun SEYMEN
MilSOFT Software Technologies

Topics

- What is RTOS?
- Structure
- Classifications
- Functions
- Debugging
- Which RTOS?

What is RTOS?

Real-Time Operating System (RTOS) is a special kind of operating system used to service ***real-time system*** requirements of software.

What is RTOS?

But... What is a Real-Time System?

A Real-time System is a system which has ***precise timing requirements*** to give output of a system.

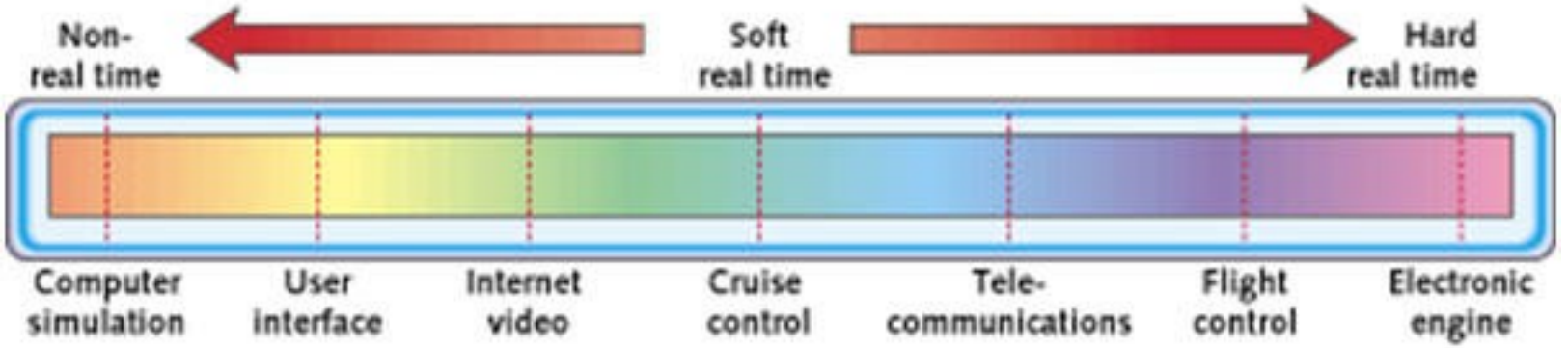
In other words, a system's correctness depends not only on the ***correctness of the logical result*** of the computation but also on the ***resulting delivery time***.

What is RTOS?

Hard vs. Soft Real-time

If a deadline miss leads to a catastrophic failure of a system, we can call this system as a **hard real-time** system.

If the system has some acceptable degraded performance caused by deadline misses, then we can classify this system as a **soft real-time** system.



What is RTOS?

When and Why we should use RTOS?

It may not be necessary to use an RTOS to meet real-time system / software requirements.

This should be assessed according to the hardware of your system, the complexity of the work to be performed, project time and financial resources of your project.

There must be a suitable hardware and software design with proper use of RTOS in order to achieve a real-time system.

RTOS Structure

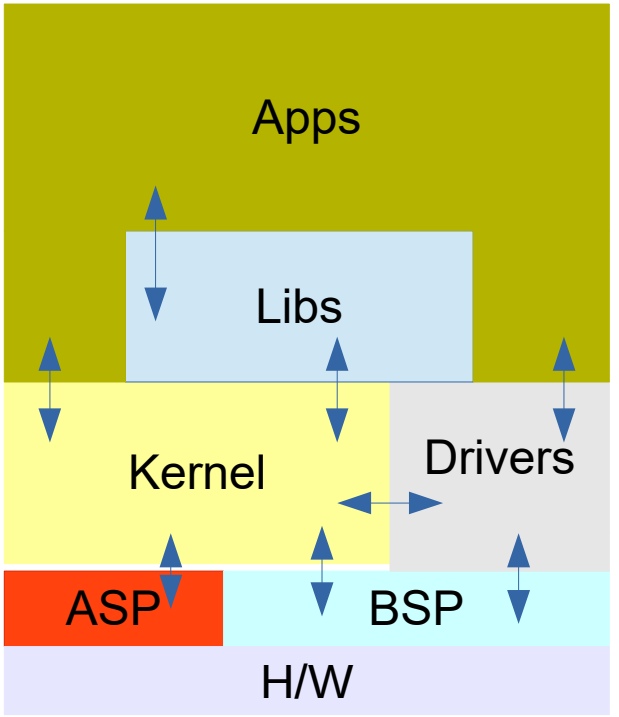
ASP (Architecture Support Package) : Processor specific support for kernel. ie. PowerPC, ARM, MIPS... A kind of HAL.

BSP (Board Support Package) BSP keeps board specific information such as the addresses and number of the devices in the hardware, external interrupt routing etc.

Kernel is the core of RTOS.

Drivers allow use of H/W.

Libraries are the collection of software routines that applications use.



RTOS Classifications

- Kernel Type
  - Nanokernel : RTOS with a nano kernel has primitive task management capabilities with interrupts and thin layer of HAL. User is responsible for device drivers and completing HAL. Very small footprint, fits for MCU with very small memories.
  - Microkernel : Microkernel provides essential kernel functions in kernel, other services are provided by processes ie. file system, network.
  - Monolitic kernel: All in one, every service is inside the kernel.
- Size/Footprint
  - Small Footprint
  - Medium Footprint
  - Large Footprint
- Memory Protection
  - No Memory protection
  - MMU/MPU utilized

RTOS Functions

Process Scheduling

- Clock Driven
- Priority Driven

Inter Process Communication

- Data exchange via Shared Memory
- Data exchange via Signals

RTOS Functions

Clock Driven - Fixed Time Scheduling

Fixed time scheduling focuses on exact deadlines and responses.

In this type of scheduling

- Processes counts and Periods should be defined in design time...
- Worst Case Execution Times (WCET) should be calculated and analyzed.
- There should be no interrupts (used polling instead) or Interrupt handling times should be added to WCET.
- Processes should be designed to be independent, (no blocking calls)...
- Aperiodic processes are not allowed, but aperiodic events can be checked and processed in periodic processes...

RTOS Functions

Clock Driven - Fixed Time Scheduling

Advantages :

- Easy to implement a single table is all what is needed, no complex run queues etc.
- No concurrency control or synchronization is required.
- Easy to be proven (It can be validated, tested and certified with very high confidence).

Disadvantages:

- Inflexible.
- Each process should start and finish in one call.
- All timing calculations should be handled in cycles.

Best suited for systems which are rarely modified once built.

RTOS Functions

Clock Driven - Fixed Time Scheduling

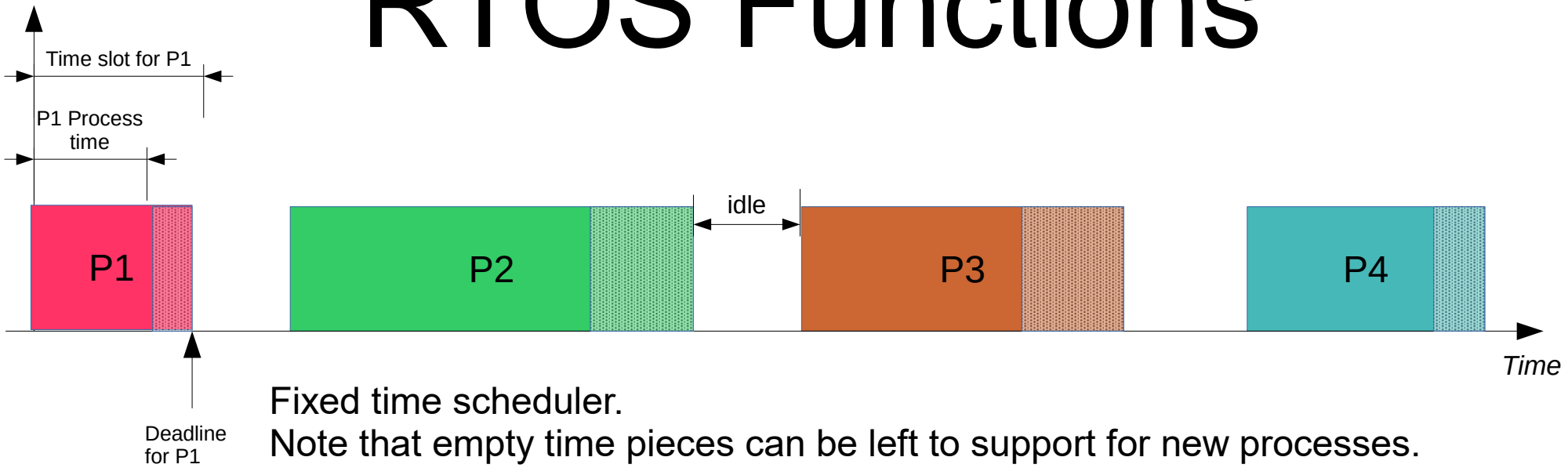
Some RTOS allow multiple fixed time scheduler schemes (with multiple time tables).

This may be useful in Initialization, Operation and Failure states of system.

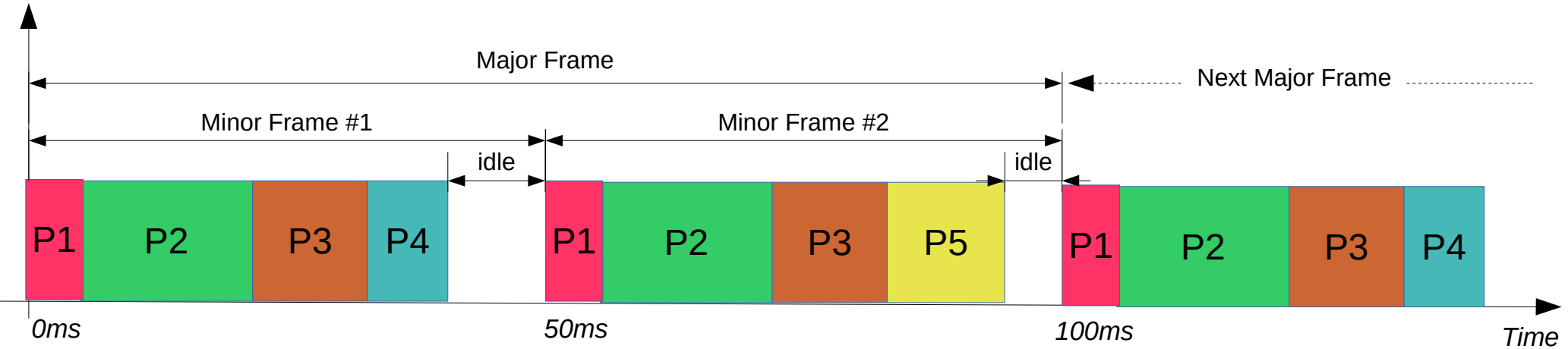
Also this can be used for framing technique that will allow more flexible scheduling.

FTS is preferred in safety critical hard systems. (Autopilots, guidance computers, air data computers etc...)

RTOS Functions



Fixed time scheduler.
Note that empty time pieces can be left to support for new processes.



Fixed Time Scheduling with 2 minor frames

RTOS Functions

Priority Driven - Weighted Round Robin

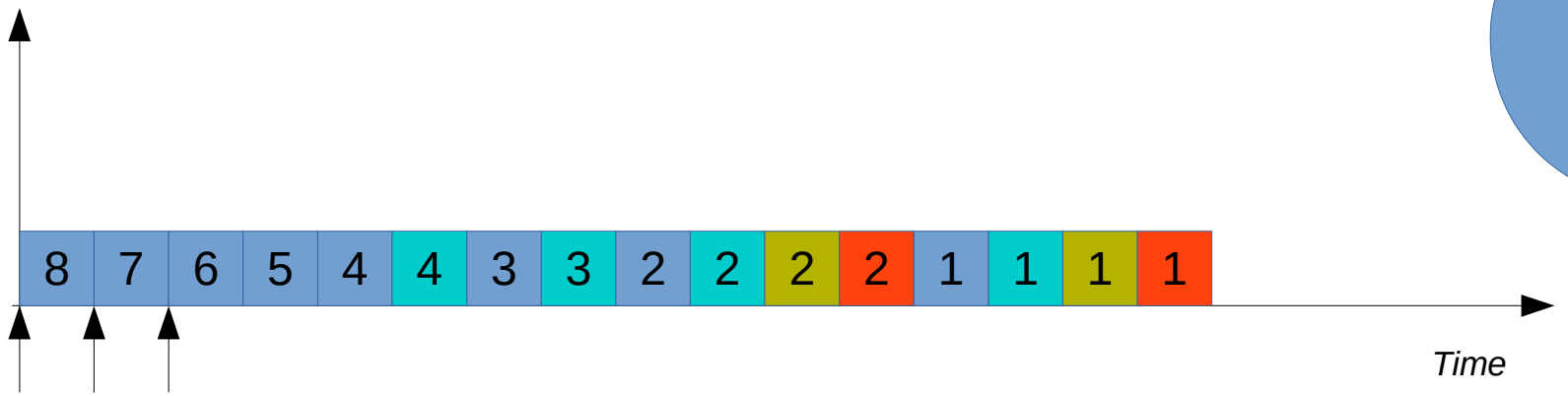
This type of scheduling is guarantees to give some CPU load for every process.

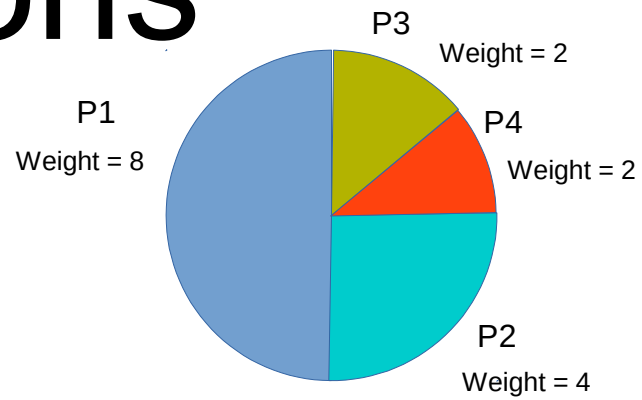
Processes with more weight will take more CPU time.

Scheduler computes for each time quanta this can be a overhead for crowded systems.

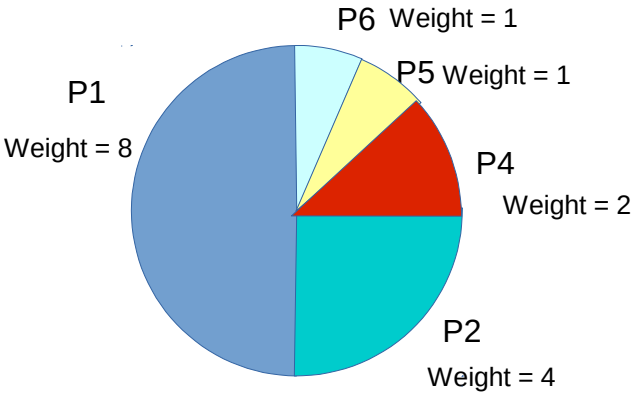
Rarely used in hard heal-time systems but it is used effective in soft real-time systems.

RTOS Functions

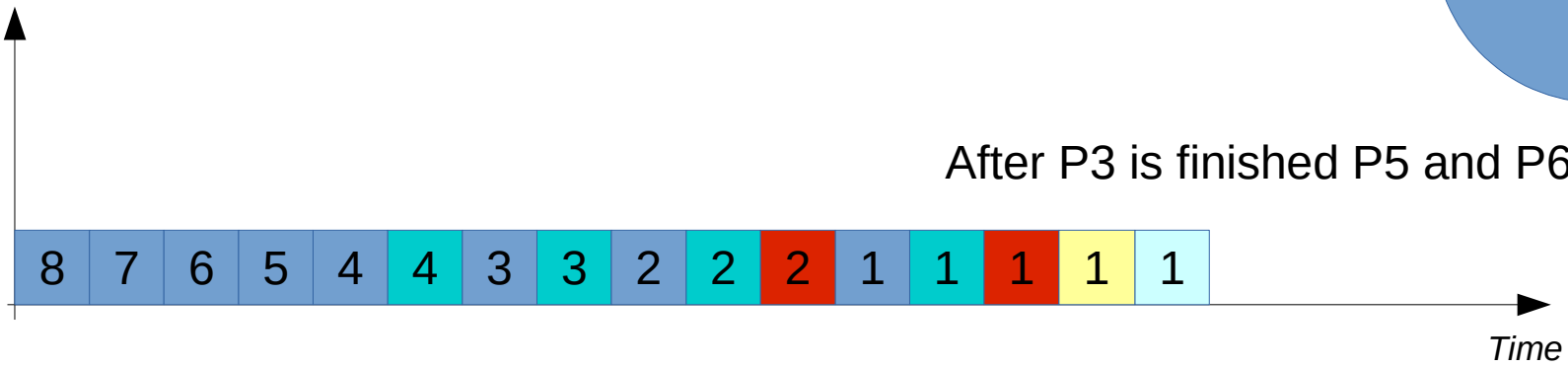




Scheduling order computed in each time slice.



After P3 is finished P5 and P6 introduced.



Weighted Round Robin

RTOS Functions

Priority Driven – Priority Preemptive

This type of scheduler is based on process priority.

Higher priority processes have higher chance to run and finish before deadline.

Scheduling decisions are made at

- process releases,
- process completions
- CPU idle

Processes are placed in one or more queues; at each event, the ready job with the highest priority is executed.

RTOS Functions

Priority Driven – Priority Preemptive

Fixed Priority

Process priority is defined during design time.

- Rate Monotonic
  - Priorities are assigned to processes based on their periods: the shorter the period, the higher the priority.
- Deadline Monotonic
  - Priorities are assigned to processes according to their deadlines: the shorter deadline, the higher the priority.

RTOS Functions

Priority Driven – Priority Preemptive

Dynamic Priority

Process priority is recalculated at schedule points.

- Earliest Deadline First (EDF)
  - Priorities are assigned to processes according to their deadlines. The earlier the deadline, the higher the priority.
- Least-Slack-Time-First (LST)
  - Priorities are assigned to processes based on their slacks: the smaller the slack, the higher the priority.

RTOS Functions

Priority Driven – Priority Preemptive

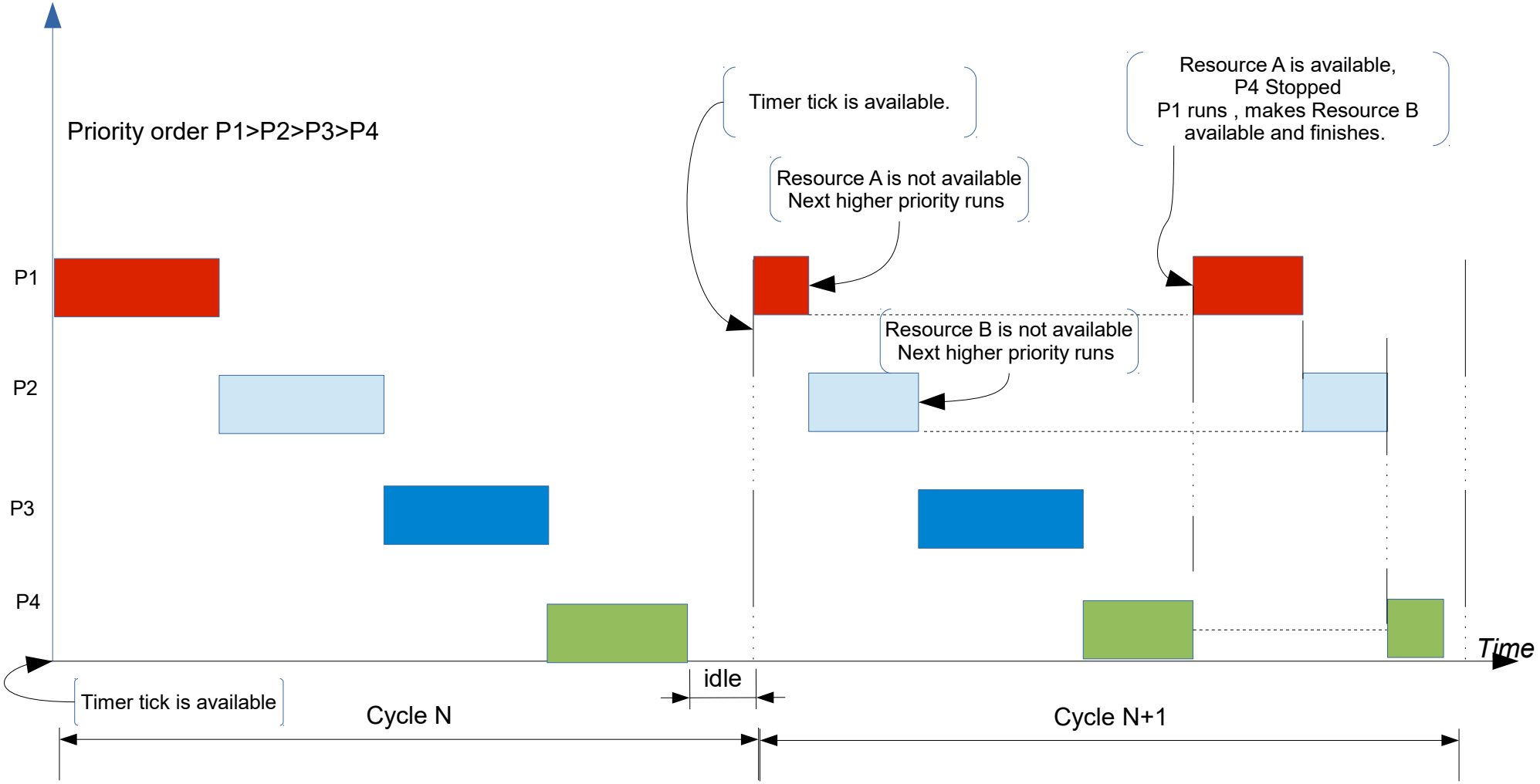
Advantages

- Easy to implement. It does not require the information on the release times and execution times of the jobs a priori.
- Better for applications with varying time and resource requirements.
- Small run-time overheads.

Disadvantages

- The timing behavior of a priority driven system is non-deterministic. It is harder to validate for correctness.
- Interrupts and IO operations may change the behavior if not analyzed well.

RTOS Functions



Each process needs a timer tick to start.
P1 uses a resource system (ex. Serial port) Resource A
P2 uses another resource produced by P1: Resource B

Example : Priority preemptive scheduling with blocking

RTOS Functions

Inter Process Communication

- Shared Memory
- Message Queues

RTOS Functions

IPC – Shared Memory (SM)

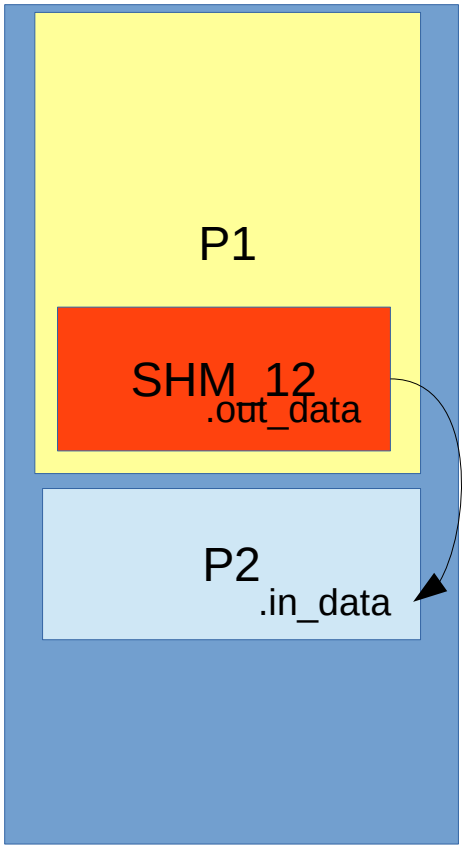
SM is defined in different ways according to RTOS and MMU usage.

If system does not have a MMU then application can access data in SM address directly.

If system has a MMU then the only way is requesting a shared memory from kernel via a system call.

This call sets MMU for assigning a memory window pointing SM.

SM can reside either in process memory or a discrete memory area.



RTOS Functions

IPC – Shared Memory (SM)

Shared memory usage is prone to corruptions in simultaneous accesses.

Best practices for using shared memory :

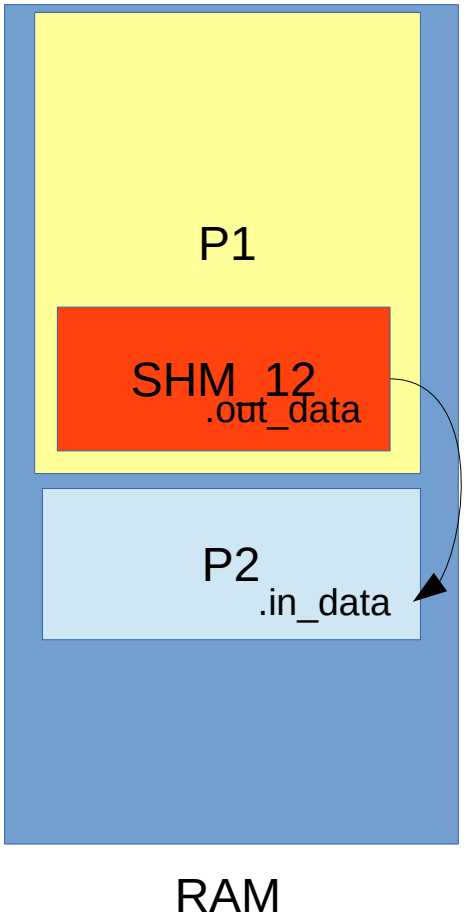
- Using the memory one way. Single process writes, others read.
- If it is needed, another SM can be used for opposite direction.
- If there is a chance of a race condition (read while write or vice versa), it must be eliminated.

Advantages:

- Fast and easy, minimum kernel interaction.
- Allows multiple reads.
- Good for polling
- Flexible. Special messaging protocols can be defines above SM.

Disadvantages:

- Uncontrolled accesses corrupt data.



RTOS Functions

IPC – Message Queues (MQ)

MQ's are one of the most used methods for IPC.

Mostly used for peer to peer communication.

Data transferred in FIFO order.

MQ creation sending and receiving usually done with kernel calls.

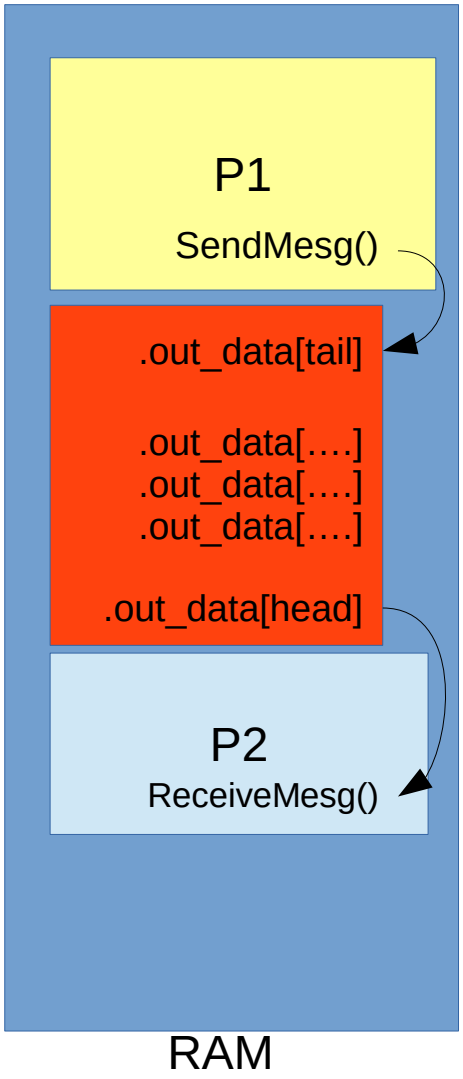
Queue length must be given enough to avoid Producer-Consumer problem.

Reads are destructive, received messages must be kept until they are obsolete

RTOS provide synchronous and asynchronous calls for MQ.

Slower than SM, good for small data.

MQ buffer can reside in process or system memory.



Debugging

On the Host

- RTOS Simulator
- Instruction Set Simulator

In the Target

- Debug Agent
- Debugger Probe

Debugging

On the host- RTOS Simulators

RTOS vendors provide a RTOS simulator for the host platform. User can load its own programs to see the behavioral model. This simulators are optimized for the host system. If the project is not complex it gives satisfactory results. Programs run on similar speeds like they run on the actual system. But I/O operations cannot be tested if not supported by host.

On the host- Instruction Set Simulators

Some RTOS vendors provide a instruction set simulator (ISS) to let the user to see the actual performance. ISS runs the original RTOS binary executable, instruction by instruction as it runs in the target architecture. Programs runs slower but very accurately so the user can see the real systems results. These results can be used for analyzing target performance. ISS allows I/O operations to be simulated.

Debugging

In the target – Using Debug Agent

Almost every RTOS provides a **debug agent** program runs in the target. This agent can manipulate the processes and controlled by a debug software on the host platform. Debugger agent communicates with the host via a serial or a ethernet port at target. This may be not suitable for some targets that have less ports.

User can stop the processes, adjust the process resources, changes its registers, provides inputs. Almost all debugging operations are available.

Some debug agents are just a high priority process, it may not access the whole RTOS resources.

If a debugger has the ability to access and control OS resources this is called OS aware Debugger.

By using debug agent, user can run the program in the real hardware so that real performance and I/O based processes can be tested in the target.

Debugging

In the Target – Using Debugger Probe

Modern targets contain JTAG (Joint Test Action Group) based interfaces with additional signals which can control the CPU. Debugger Probe is connected from this port to control and collect data from the CPU. Debugger probe connects the host environment via USB or Ethernet. There are many advantages using debugger probes over other methods:

- The debugger probe occupies a special port not used by software so all I/O can be used by the system.
- The debugger probe has control over CPU so not only the processes but also operating system kernel can be debugged.
- The hardware break points can be used so there are no chance of missing events.
- System can be debugged from boot to death.
- If the CPU architecture allows, some debuggers can record all execution history. This allows postmortem analysis. That kind of probes often called “trace probes” or “tracers”.

Which RTOS ?

There are many RTOS present in the market. Most of them differ from each other from scale, capabilities, performances, licensing and prices.

Apart from the budgetary concerns the following topics can be considered:

- Functionality
  - Flexibility of Schedulers, Inter Process Communications, I/O operations etc.
  - Memory Management
  - Development and Debugging Environment
- Performance
  - Context switching time, Interrupt response time
  - Maximum number of resources
- Scale
  - File systems, Networking, Special I/O
- Reliability
  - Error Handling, Health Monitoring

Additionally if the project requires :

- Portability (different targets/ different architecture)
- Safety (Evidences, test reports, documentation)

Which RTOS?

Management and Financial:

- Commercial or Free
  - Commercial RTOS has advantages in terms of product support and maintenance. Companies have permanent staff to develop and strengthen their products. Mostly solid support and documentation.
  - Free products (if not supported with very crowded community) have less staff responsible. Products' road maps are not clear. Because of the agility of development some documentation can be outdated.
  - User may take a free RTOS, test and use it but there should be a team for support.
- Open or closed source
  - Open source has advantages for users especially for debugging. Commercial RTOS vendors usually supply source codes with additional fee.
- Rental or Perpetual Licensing
  - Perpetual licenses are expensive but it can be used for many long term projects. Rental licenses are good for short term mini projects.
- Royalty
  - Some RTOS vendors request per target where RTOS used in. This may be 10K US$ for a military equipment and 5US$ for a pocket camera.

Sources:

Liu , J.W.S.- Real-Time Systems

Cooling, J. – Software Engineering for Real-Time Systems

Burns, A. - Real Time Systems and Programming Languages