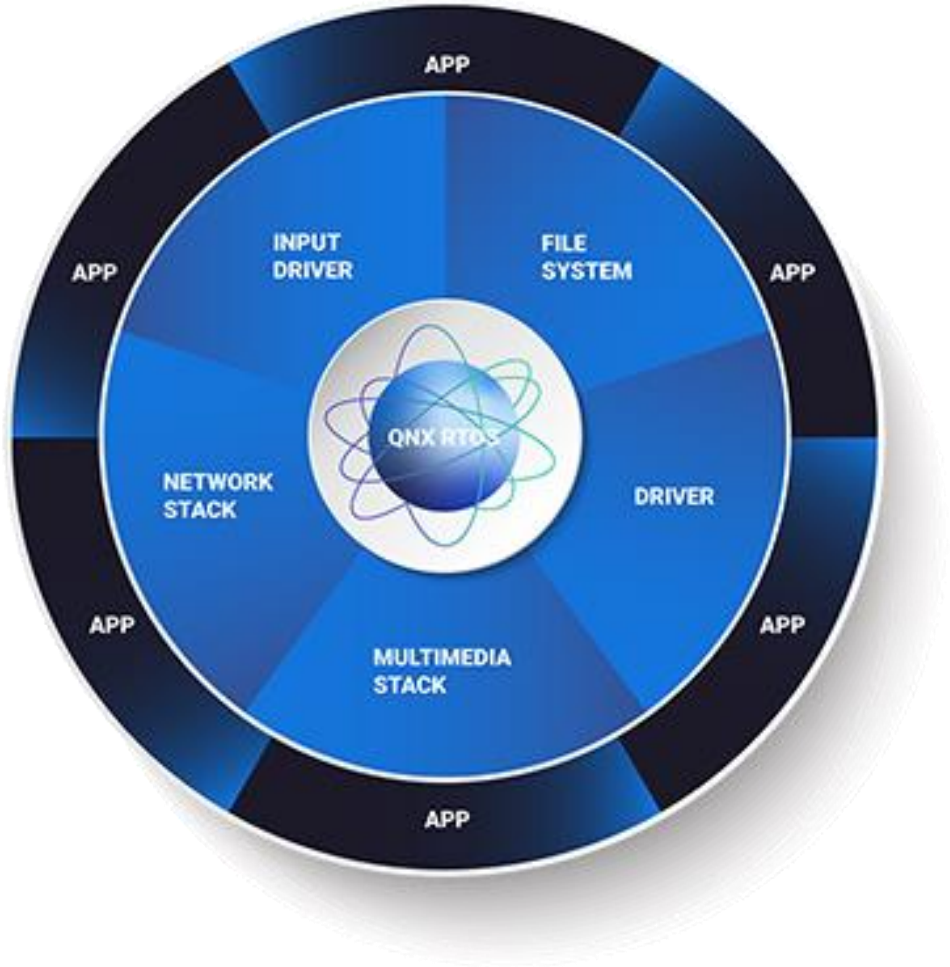
QNX Neutrino RTOS

Michele Boldo - Francesco Tosoni

What is QNX Neutrino RTOS?

- Commercial microkernel RTOS
- Since 1980
- Many application fields (e.g. medical, military, automotive, robotics …)
- Highly optimized
- Low cost



What is a microkernel RTOS?

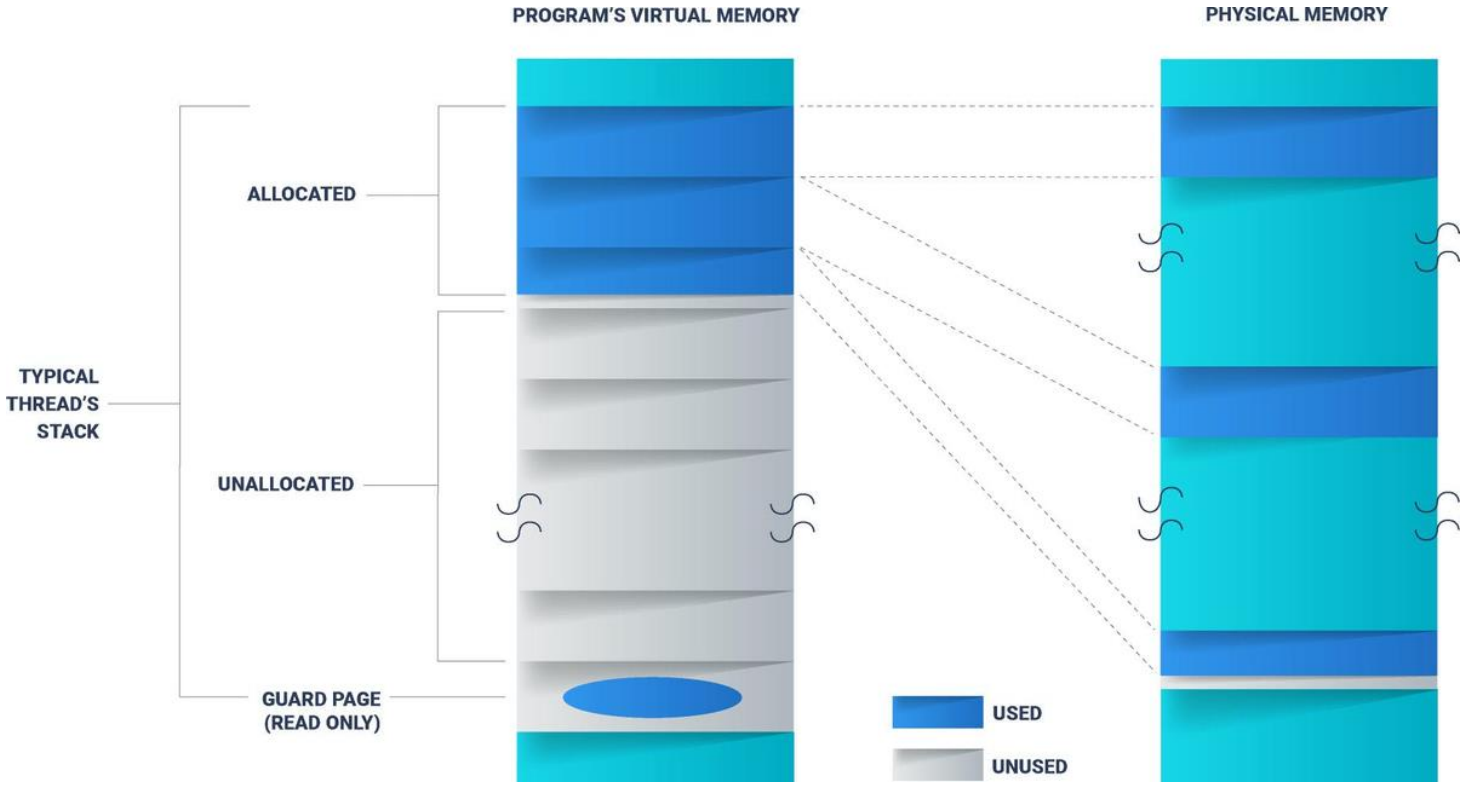
- Tiny kernel with minimal services
- Higher-level OS functionality at user level
- Modularity
- High-Availability Manager (HAM)

Microkernel RTOS

Advantages	Disadvantages
Fault isolation for high availability	Context-switch overhead
Rerstart a failed service without impact to the kernel	
Easy expansion (both drivers and OS extensions)	
Easy to debug	
Small footprint	
Less code in kernel mode increases security	

Main Features

- Spatial separation
- Temporal separation
- IPC
- Interrupt handling
- Accessing resources
- Synchronization

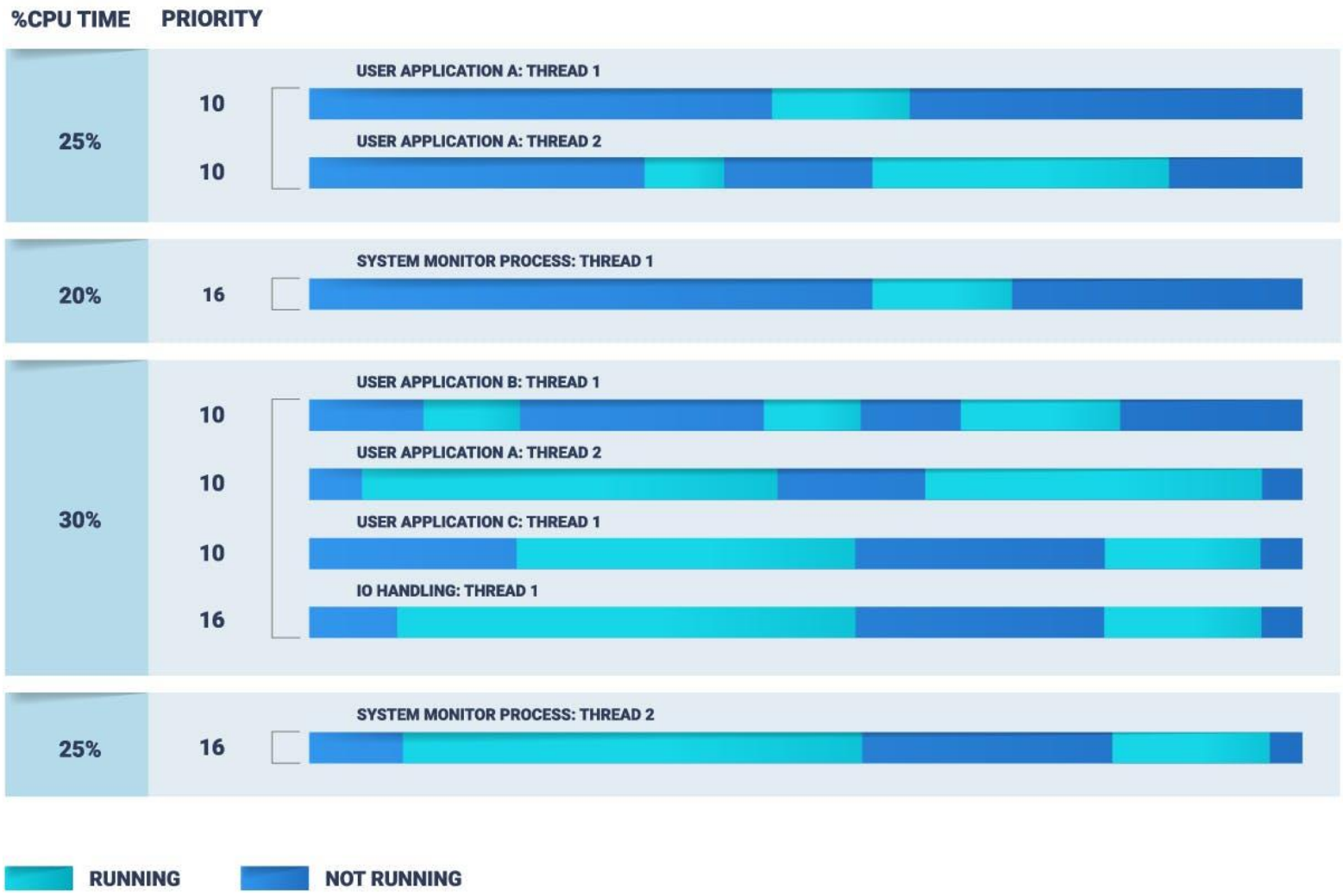


Spatial separation

- Each process with its own private address space
- Memory Management Unit (MMU): maps phisical memory to virtual memory and blocks unwanted accesses
- Without MMU, data and kernel share the same memory space (less reliability)

Time separation

- Each process has to run without depending on the timing of other systems that share the same resources
- RTOS scheduling provides temporal separation
- Solutions: partitioning resources



Static and Adaptive partitioning

- Partitioning prevents processes from monopolizing CPU cycles needed by others
- Virtual walls to split a set of resources between applications
- Primary resource: CPU time

STATIC PARTITIONING	ADAPTIVE PARTITIONING
• Divides tasks into groups and allocates a percentage of CPU time *T* to each partition • No task can consume more than *T* • When partitions need less than *T*, CPU cicles are left unused • Interrupts have to wait until the partition runs (unacceptable latency)	• Allows to reserve the minimum amount of CPU cycles *T* for a group of processes and to dynamically reassign part of *T* from partition of lower need to higher need • Guarantee CPU cycles to critical processes without compromising system performances • Faster, more efficient, low latency, minimal unused CPU cycles • Adopted by QNX Neutrino RTOS

Scheduling algorithms

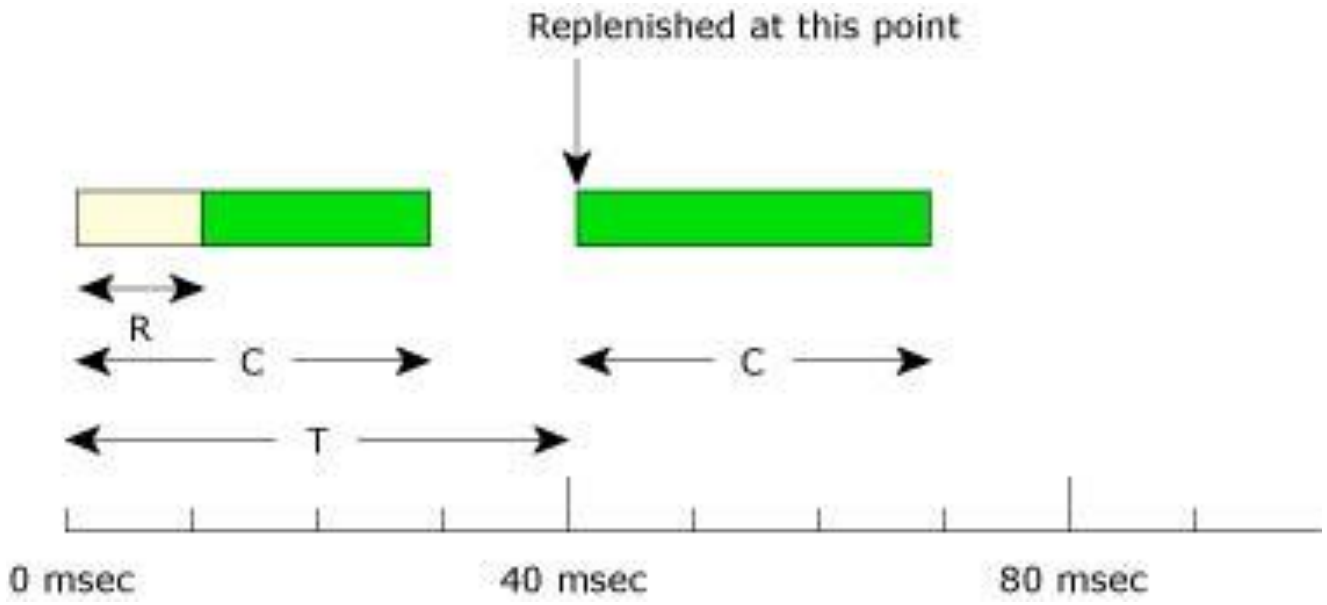
- Sporadic (Priority based)
- Round Robin
- FIFO

Priority-based preemptive scheduling

- Each process has a priority (0 to 255)
- Allows to high-priority threads to meet their deadlines, even when there is a lot of competition for resources
- High-priority thread *t1* can take over the CPU from any lower-priority thread
- *t1* can run uninterrupted unless it is preempted by an higher-priority thread

Budget

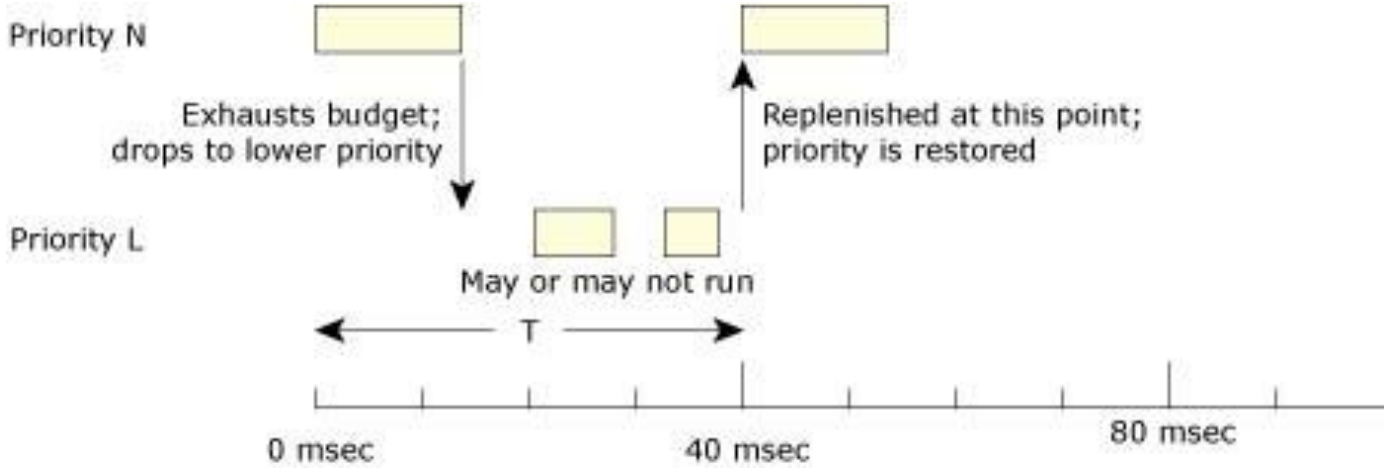
A thread's initial execution budget **C** is established, which is consumed by the thread as it runs.
C is replenished periodically (for the amount **T**).
When a thread blocks, the amount of the execution budget that's been consumed **R** is arranged to be replenished later (e.g. at 40 msec) after the thread first became ready to run.



- *Initial budget (C)*
- *Portion of budget consumed at the beginning of T (R)*
- *Replenishment period (T)*

Priority

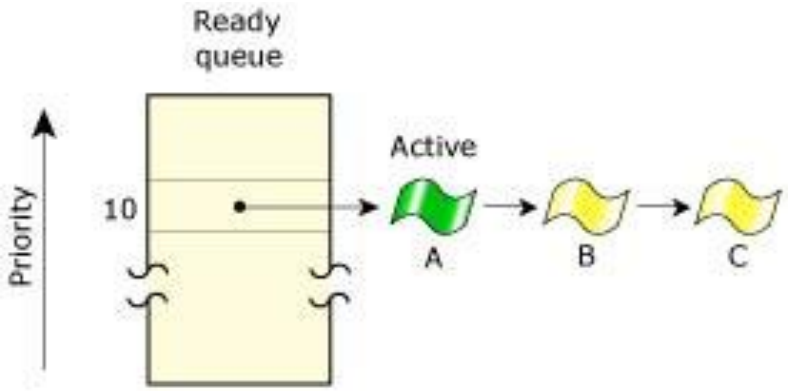
At its normal priority N, a thread will execute for the amount of time defined by its initial execution budget C. As soon as this time is exhausted, the priority of the thread will drop to its low priority L until the replenishment operation occurs.



- *Execution Budget (C)*
- *Normal priority (N)*
- *Low priority (L)*
- *Replenishment period (T)*

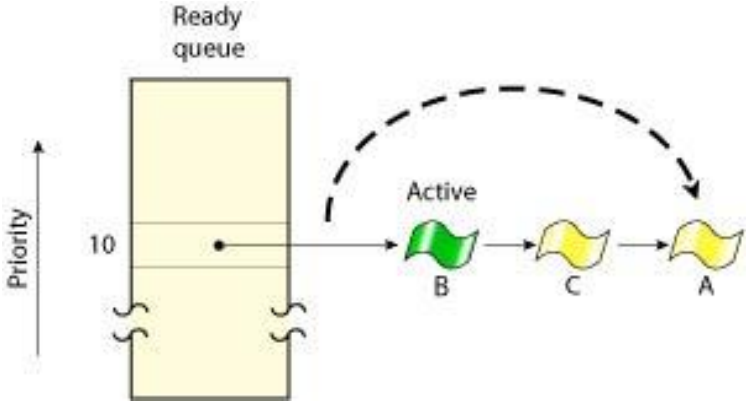
When two or more threads that share the *same priority* are *ready*, FIFO or ROUND ROBIN are applied.
Both algorithms are preemptive.

FIFO

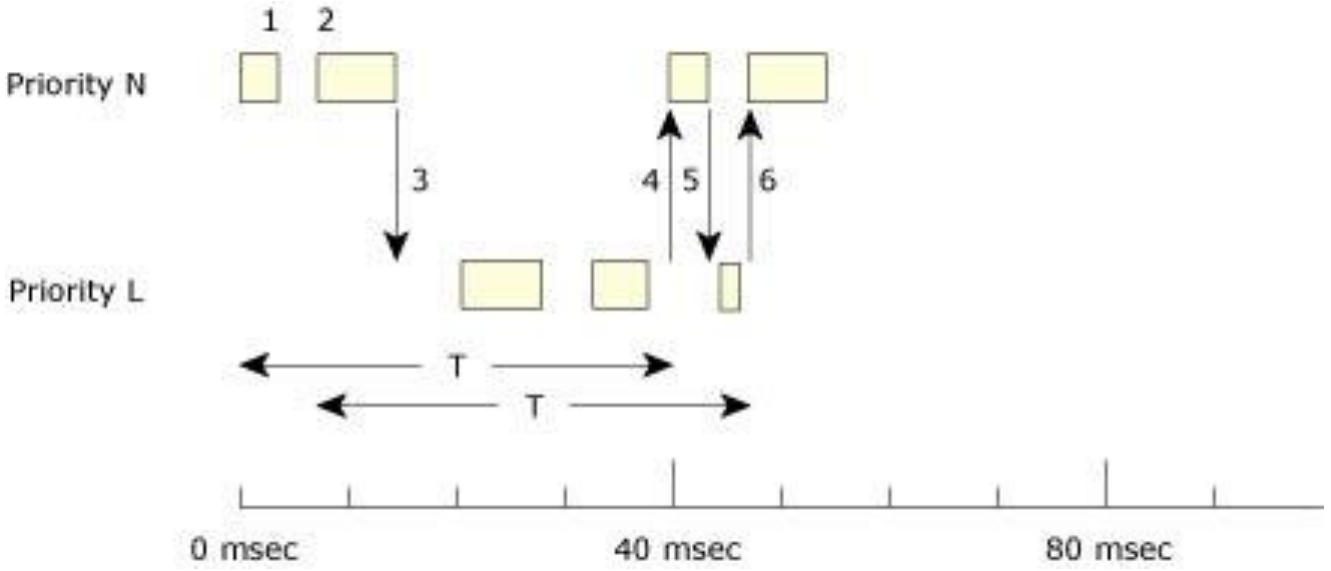


The selected thread to run continues executing until it voluntarily relinquishes control

ROUND ROBIN



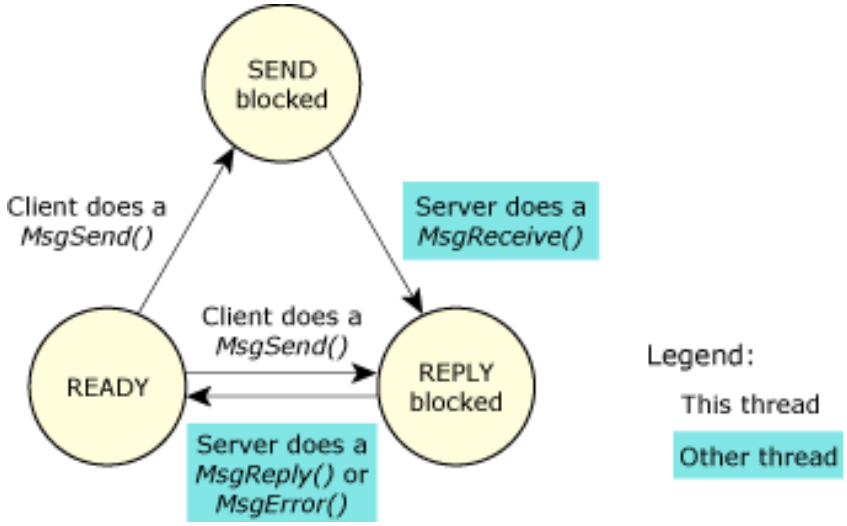
Once it consumes its timeslice, a thread is preempted and the next READY thread at the same priority level is given control. A timeslice is 4 * the clock period

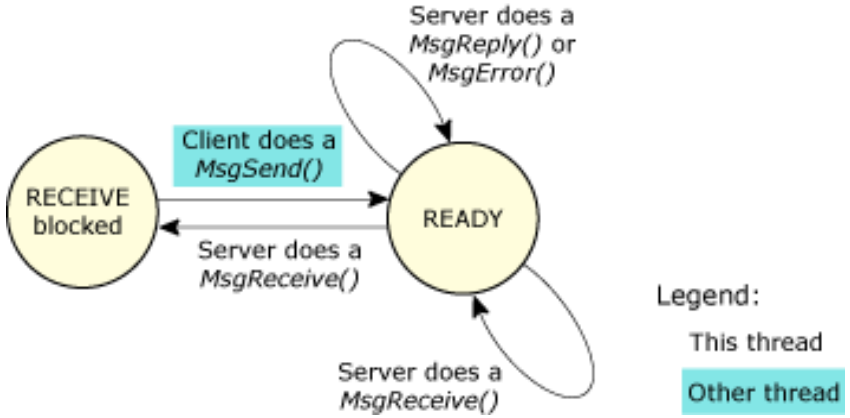


- C = 10 ms
- T = 40 ms

Example

1. The initial run of the thread is blocked after 3 msec, so a replenishment operation of 3 msec is scheduled to begin at the 40-msec mark, i.e. when its first replenishment period has elapsed.
2. The thread gets an opportunity to run again at 6 msec, which marks the start of another replenishment period (T). The thread still has 7 msec remaining in its budget.
3. The thread runs without blocking for 7 msec, thereby exhausting its budget, and then drops to low priority L, where it may or may not be able to execute. A replenishment of 7 msec is scheduled to occur at 46 msec (40 + 6), i.e. when T has elapsed.
4. The thread has 3 msec of its budget replenished at 40 msec (see Step 1) and is therefore boosted back to its normal priority.
5. The thread consumes the 3 msec of its budget and then is dropped back to the low priority.
6. The thread has 7 msec of its budget replenished at 46 msec (see Step 3) and is boosted back to its normal priority.





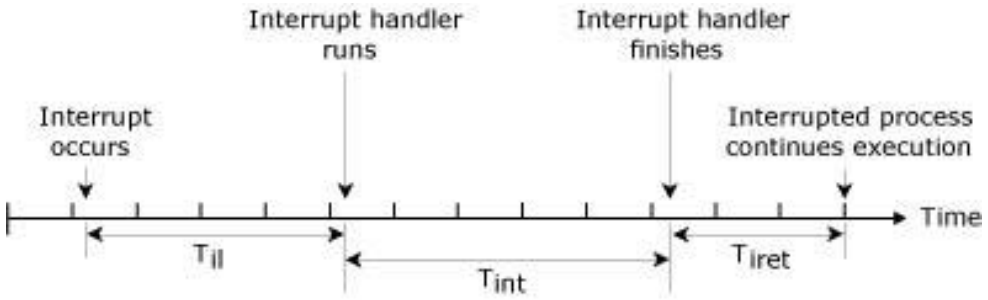
Interprocess communication (IPC)

- Additional layer of isolation between address spaces
- In QNX, IPC maps POSIX calls to messages
- High level of architectural decoupling for safety standards (i.e. ISO 26262)
- Other forms of IPC are also available
- POSIX-compliant

IPC Methods

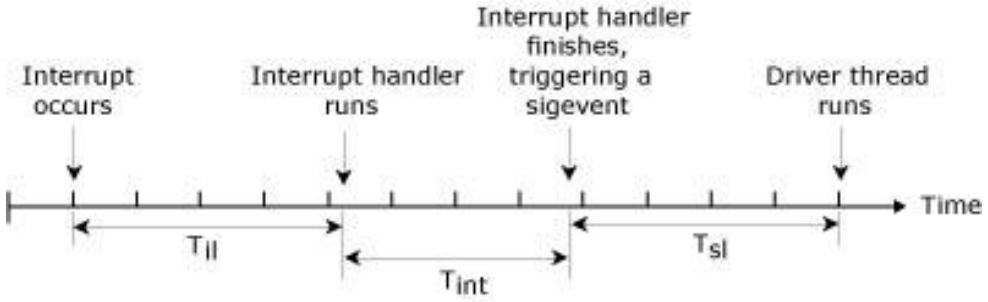
- Message-passing
- Signals
- POSIX message queues
- Shared memory
- Pipes External
- FIFOs

INTERRUPT LATENCY



T_{il} interrupt latency
T_{int} interrupt processing time
T_{iret} interrupt termination time

SCHEDULING LATENCY



T_{il} interrupt latency
T_{int} interrupt processing time
T_{sl} scheduling latency

Interrupts handling

Interrupt handlers are of higher priority than any thread, but they're not scheduled in the same way as threads. If an interrupt occurs, then:

- Whatever thread was running loses the CPU handling the interrupt
- The hardware runs the kernel (if there isn't a critical section running)
- The kernel calls the appropriate interrupt handler

Possible NO interrupt mode

Accessing resources

PRIORITY INHERITANCE

- *t1*, *t2* threads where $P_1 > P_2$
- *t2* locks a mutex
- If *t1* attempts to lock the mutex, then the effective priority of *t2* will be increased to P_1.
- *t2* will return to its real priority when it unlocks the mutex.
- This scheme not only ensures that the higher-priority thread will be blocked waiting for the mutex for the shortest possible time, but also solves the classic priority-inversion problem.

- Mutex
- Condvars
- Barriers
- Sleepon locks
- Reader/Writer locks
- Semaphores
- FIFO scheduling
- Send/Receive/Reply
- Atomic operations

Synchronization

Why choosing QNX Neutrino?

- Safety
- Security
- Development environment
- Graphics annd human interface
- Mantainance and updates
- Hardware support (ARM and x86)