Preemption Models

The various preemption models are kernel specific. In principle, user space programs are always preemptible.

Preemption of a running task is performed by the scheduler. This action can be triggered by a kernel interaction like a system call or an asynchronous event like an interrupt. The scheduler saves the context of the preempted task and restores the context of the new task.

The Linux kernel implements several preemption models. The desired model is selected at build time of the kernel. The “Fully Preemptible Kernel” preemption model must be selected to obtain Linux as an RTOS. For the sake of completeness a list and a short explanation of the existing Linux preemption models is given. The last two entries are available only with the PREEMPT_RT patch.

- **No Forced Preemption (server):** *The traditional Linux preemption model, geared towards throughput* 1). System call returns and interrupts are the only preemption points.
- **Voluntary Kernel Preemption (Desktop):** *This option reduces the latency of the kernel by adding more “explicit preemption points” to the kernel code [. . .] at the cost of slightly lower throughput* 2). In addition to explicit preemption points, system call returns and interrupt returns are implicit preemption points.
- **Preemptible Kernel (Low-Latency Desktop):** *This option reduces the latency of the kernel by making all kernel code (that is not executing in a critical section) preemptible* 3). An implicit preemption point is located after each preemption disable section.
- **Preemptible Kernel (Basic RT):** This preemption model resembles the “Preemptible Kernel (Low-Latency Desktop)” model. Besides the properties mentioned above, threaded interrupt handlers are forced (as when using the kernel command line parameter `threadirqs`). This model is mainly used for testing and debugging of substitution mechanisms implemented by the PREEMPT_RT patch.
- **Fully Preemptible Kernel (RT):** All kernel code is preemptible except for a few selected critical sections. Threaded interrupt handlers are forced. Furthermore several substitution mechanisms like sleeping spinlocks and rt_mutex are implemented to reduce preemption disabled sections. Additionally, large preemption disabled sections are substituted by separate locking constructs. This preemption model has to be selected in order to obtain real-time behavior.

1) , 2) , 3) see kernel/Kconfig.preempt