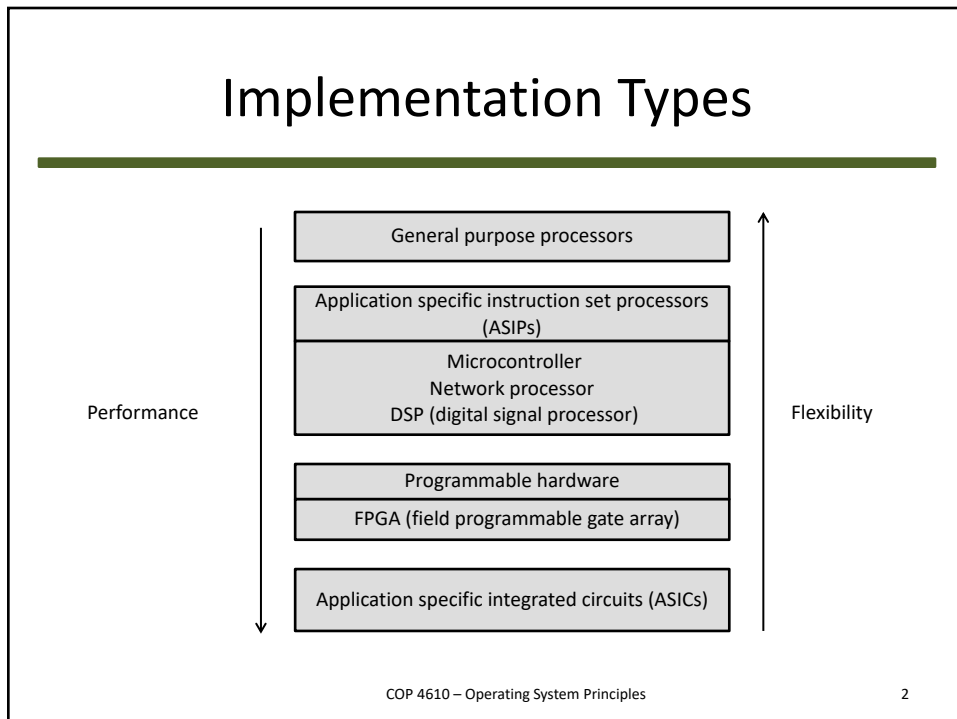


COP 4610

Operating System Principles

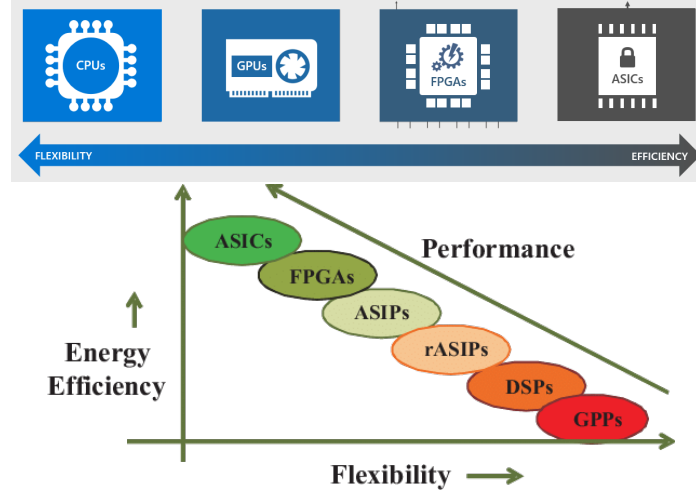
Embedded (Real-Time) Systems

1



2

Implementation Types



COP 4610 – Operating System Principles

3

3

Processor Spectrum

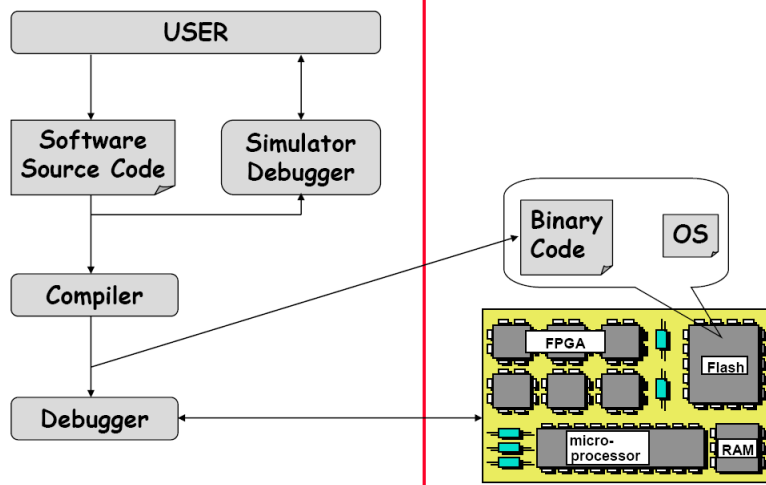
- General purpose microprocessor
Workstations, PCs, high-performance applications, ...
- Microcontroller (MCU)
Embedded systems, control systems, ...
- Digital signal processors
Signal processing applications, ...
- Special purpose (co-)processor
Security or image processing applications, ...
- FPGA/ASIC-based processors
Very dedicated scenarios (engine control, cell phones, cryptocurrency mining, ...)

COP 4610 – Operating System Principles

4

4

Cross Compilation

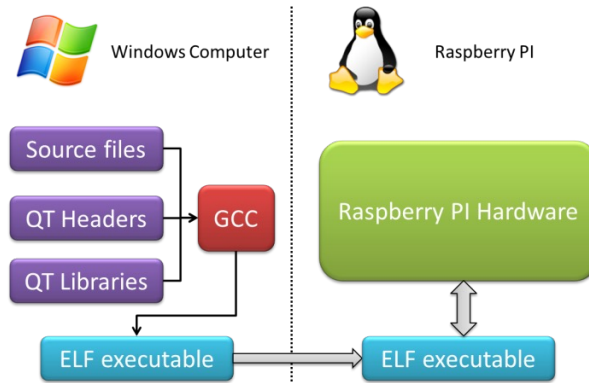


COP 4610 – Operating System Principles

5

5

Example

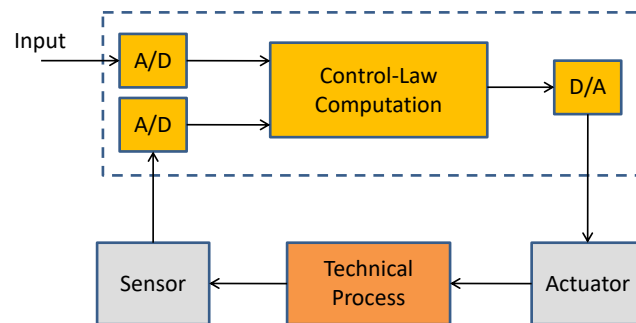


COP 4610 – Operating System Principles

6

6

Generic Real-Time Control System



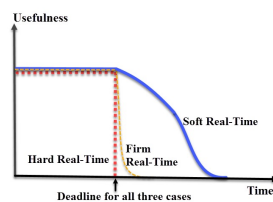
COP 4610 – Operating System Principles

7

7

Real-Time Systems

- A real-time system (RTS) is a system that must satisfy explicit (bounded) response-time constraints or risk severe consequences, including failure.
 - **Soft RTS:** Miss deadline(s)-> performance degraded
 - **Hard RTS:** Miss deadline(s)-> system failure
 - **Firm RTS:** Miss deadline(s)-> some tolerance



Who decides deadlines?

COP 4610 – Operating System Principles

8

8

Real-Time Systems

System	Real-Time Classification	Explanation
Avionics weapons delivery system in which pressing a button launches an air-to-air missile	Hard	Missing the deadline to launch the missile within a specified time after pressing the button may cause the target to be missed, which will result in a catastrophe
Navigation controller for an autonomous weed-killer robot	Firm	Missing a few navigation deadlines causes the robot to veer out from a planned path and damage some crops
Console hockey game	Soft	Missing even several deadlines will only degrade performance

COP 4610 – Operating System Principles

9

9

Schedule

- Given a set of tasks $J = \{J_1, J_2, \dots\}$:
 - A **schedule** is an assignment of tasks to the processor, such that each task is executed until completion.
 - A schedule can be defined as an integer step function $\sigma : R \rightarrow \{0, 1, 2, \dots, n\}$ where $\sigma(t)$ is the task executed at time t ; if $\sigma(t) = 0$ then the processor is called **idle**.
 - If $\sigma(t)$ changes its value at some time, then the processor performs a **context switch**.
 - Each interval, where $\sigma(t)$ is constant, is called a **time slice**.
 - A **preemptive schedule** is a schedule where the running task can be arbitrarily suspended at any time, to assign the CPU to another task according to a predefined scheduling policy.

COP 4610 – Operating System Principles

10

10

Schedule & Timing

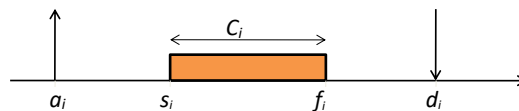
- A schedule is said to be **feasible**, if all tasks can be completed according to a set of specified constraints.
- A set of tasks is said to be **schedulable** with algorithm A, if A can produce a feasible schedule.
- **Arrival time** a_i or **request** or **release time** r_i is the time at which a task becomes ready for execution.
- **Computation time** C_i is the time necessary to the processor for executing the task without interruption.
- **Deadline** d_i is the time by which a task should be completed.
- **Start time** s_i is the time by which a task starts its execution.
- **Finishing time** f_i is the time by which a task finishes its execution.

COP 4610 – Operating System Principles

11

11

Schedule & Timing



- Using the above definitions, we have $d_i \geq a_i + C_i$
- **Lateness** $L_i = f_i - d_i$ represents the delay of a task completion with respect to its deadline; note that if a task completes before the deadline, its lateness is negative.
- **Tardiness** or **exceeding time** $E_i = \max(0, L_i)$ is the time a task stays active after its deadline.
- **Laxity** or **slack time** $X_i = d_i - a_i - C_i$ is the maximum time a task can be delayed after its activation to still complete by its deadline.

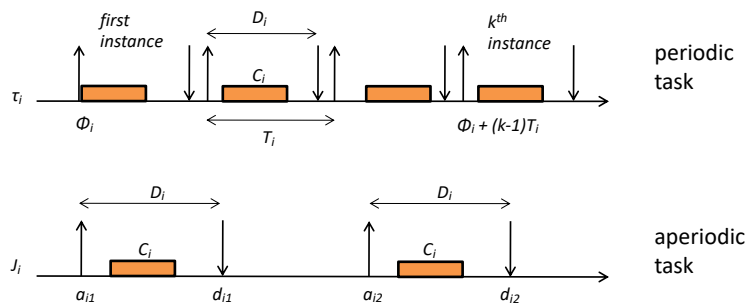
COP 4610 – Operating System Principles

12

12

Schedule & Timing

- **Periodic task** τ_i : infinite sequence of identical activities, called instances or jobs, that are regularly activated at a constant rate with period T_i . The activation time of the first instance is called phase Φ_i .

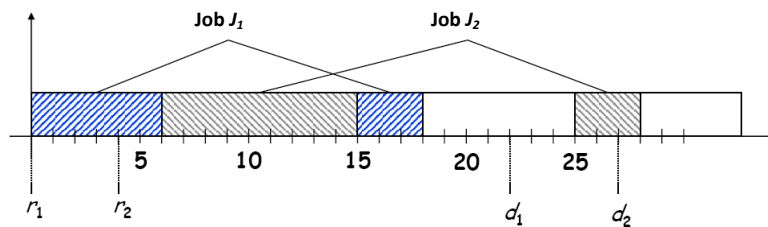


COP 4610 – Operating System Principles

13

13

Example



- Computation times: $C_1 = 9, C_2 = 12$
- Start times: $s_1 = 0, s_2 = 6$
- Finishing times: $f_1 = 18, f_2 = 28$
- Lateness: $L_1 = -4, L_2 = 1$
- Tardiness: $E_1 = 0, E_2 = 1$
- Laxity: $X_1 = 13, X_2 = 11$

COP 4610 – Operating System Principles

14

14

Worst-Case Execution Time (WCET)

- Analyze and instrument the task
- Analyze the compiler
- Analyze the operating system
- Analyze the hardware
- **Analytical Approach:** all sub-problems are solved analytically.
- **Pragmatic Approach:** investigate and instrument the source program to generate test cases that are biased towards the maximum execution time. Execute the test cases on the target system.

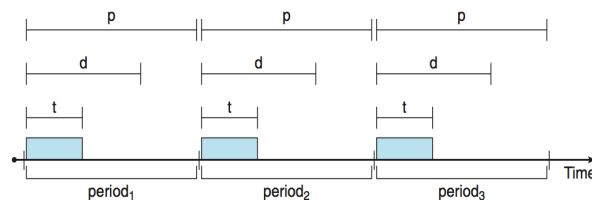
COP 4610 – Operating System Principles

15

15

Priority-Based Scheduling

- Many real-time processes are **periodic**, i.e., they require CPU at constant intervals
 - Has processing time t , deadline d , period p
 - $0 \leq t \leq d \leq p$
 - **Rate** of periodic task is $1/p$



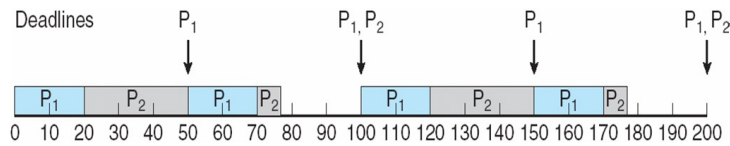
COP 4610 – Operating System Principles

16

16

Rate Monotonic Scheduling

- **Priority = the inverse of its period**
- Shorter periods = higher priority
- P₁: period of 50 (c=20); P₂: period of 100 (c=35)
- P₁ is assigned a higher priority than P₂



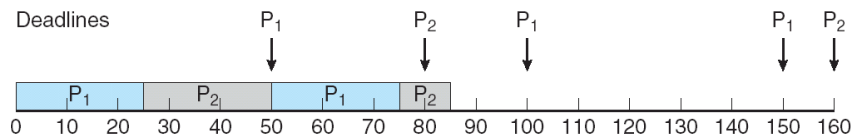
COP 4610 – Operating System Principles

17

17

Missed Deadlines with Rate Monotonic Scheduling

(p, c), p=d
 P1(50, 25)
 P2(80, 35)



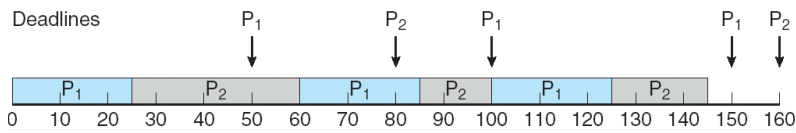
COP 4610 – Operating System Principles

18

18

Earliest Deadline First Scheduling (EDF)

- Priorities are assigned according to **deadlines**:
 - the earlier the deadline, the higher the priority
 - the later the deadline, the lower the priority

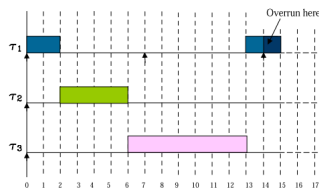


COP 4610 – Operating System Principles

19

19

Preemption



Assume all tasks are released at time 0.

	Priority	Period	Computation time
τ_1	1	7	2
τ_2	2	16	4
τ_3	3	31	7

Figure 1: Priorities without preemption

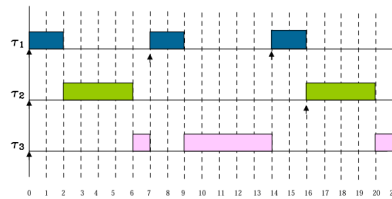


Figure 2: Priorities with preemption

20

Utilization

- **Execution time of a task** - time it takes for a task to run to completion
- **Period of a task** - how often a task is being called to execute; can generally assume tasks are periodic although this may not be the case in real-world systems
- **CPU utilization** - the percentage of time that the processor is being used to execute a specific scheduled task

$$U = \frac{e_i}{P_i}$$

– where e_i is the execution time of task i , and P_i is its period

- **Total CPU utilization** - the summation of the utilization of all tasks to be scheduled

$$\sum_{i=1}^n \frac{e_i}{P_i}$$

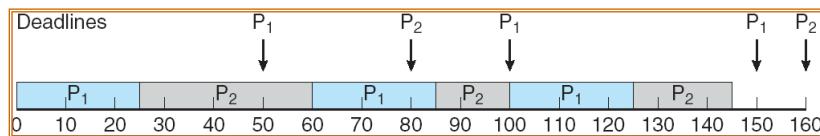
21

Utilization: EDF

- EDF: shorter **absolute** deadline → Higher priority
- Utilization bound $U_b = 1$
- U_b is **necessary** and **sufficient**

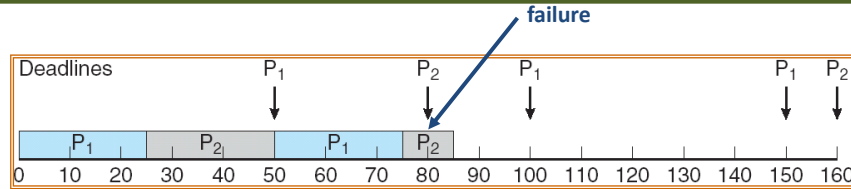
Process P_1 : service time = 25, period = 50, deadline = 50

Process P_2 : service time = 35, period = 80, deadline = 80



22

Utilization: RMS



Process P₁: service time = 25, period = 50, deadline = 50
 Process P₂: service time = 35, period = 80, deadline = 80

RMS is guaranteed to work if

N = number of processes
sufficient condition

$$u = \sum_{i=1}^N \frac{t_i}{p_i} \leq N(\sqrt[N]{2} - 1);$$

$$\lim_{N \rightarrow \infty} N(\sqrt[N]{2} - 1) = \ln 2 \approx 0.693147$$

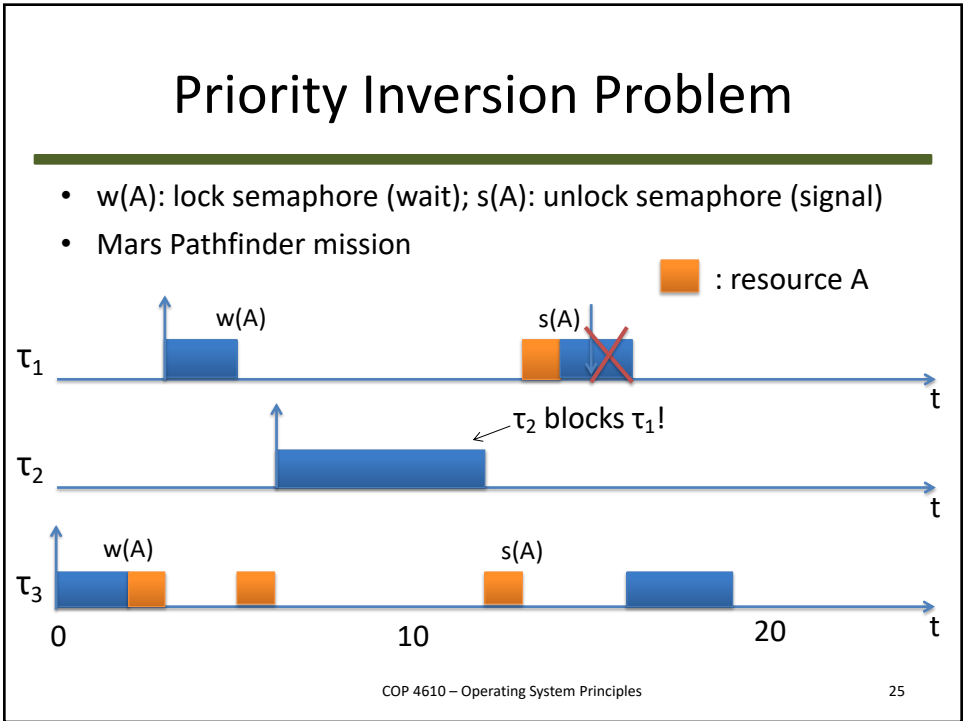
N	$N(\sqrt[N]{2} - 1)$
2	0,828427
3	0,779763
4	0,756828
5	0,743491
10	0,717734
20	0,705298

23

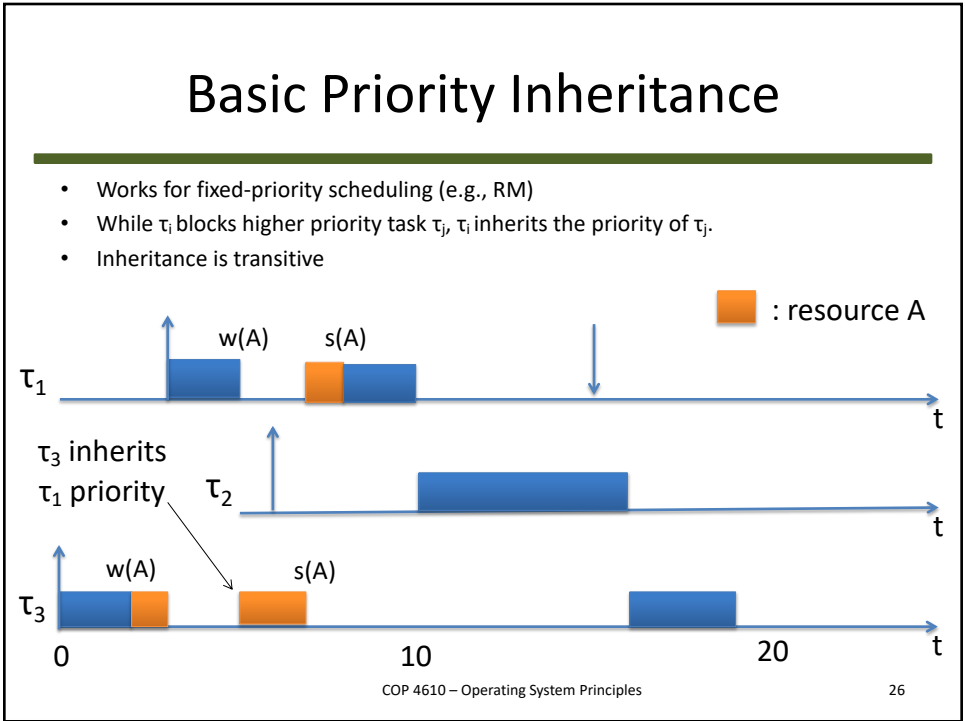
EDF vs RMS

- RMS
 - RMS may not guarantee schedulability even when $U < 1$
 - Low overhead: priorities do not change for a fixed task set
- EDF
 - EDF guarantees schedulability as long as $U \leq 1$
 - High overhead: task priorities may change dynamically

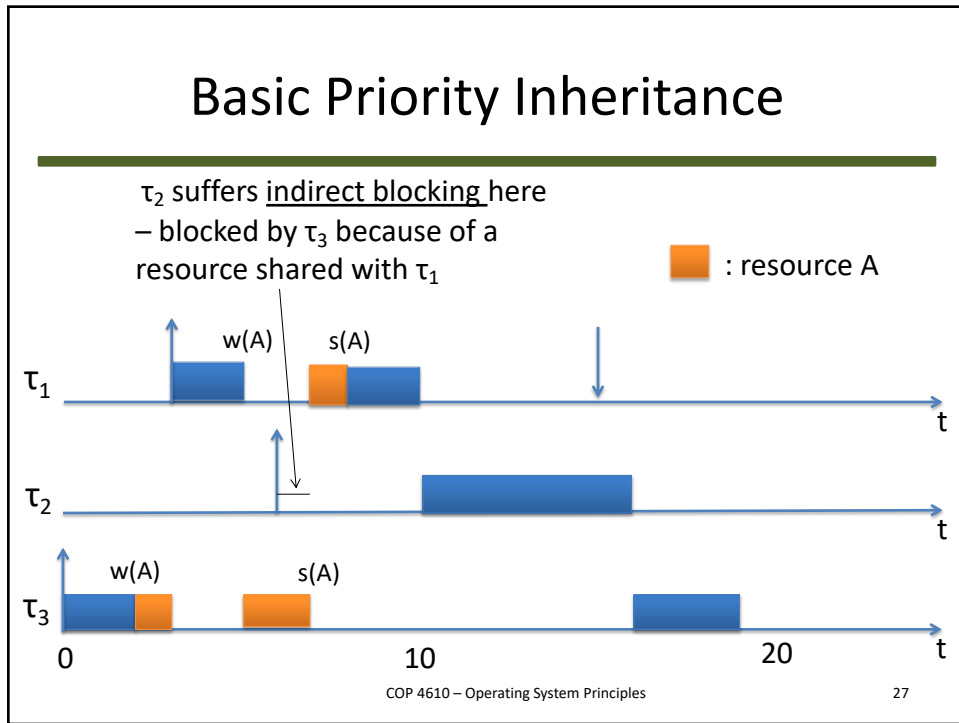
24



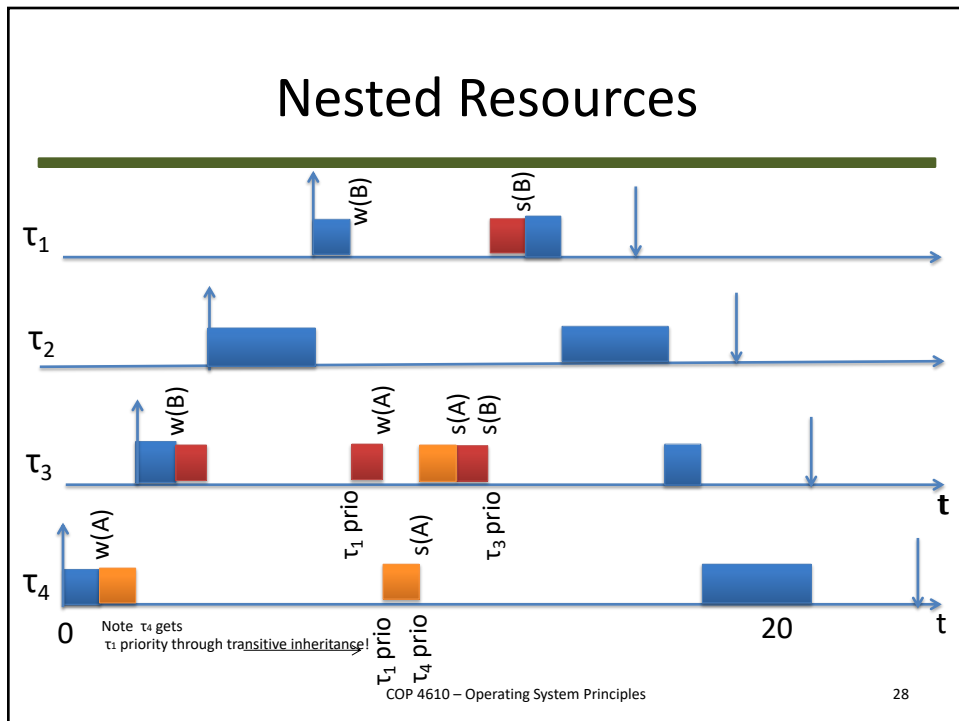
25



26



27



28

Commercial RTOS

- Real-Time Operating Systems (RTOS)
- Different from traditional OS: more predictability
- Used in the following areas such as:
 - Embedded Systems or Industrial Control Systems
 - Parallel and Distributed Systems
- E.g., LynxOS, VxWorks, pSoS, QNX , bluecat
- Traditionally these systems can be classified into a Uniprocessor, Multiprocessor, or Distributed Real-Time OS

29

Lynx OS

- Microkernel design
 - Means the kernel footprint is small
 - Only 28 KB in size
- The small kernel provides essential services in scheduling, interrupt dispatching, and synchronization
- The other services are provided by kernel lightweight service modules, called Kernel Plug-Ins (KPIs)
- New KPIs can be added to the microkernel and can be configured to support I/O, file systems, TCP/IP, streams and sockets
- Can function as a multipurpose UNIX OS

30

Lynx OS

- Here KPIs are multi-threaded, which means each KPI can create as many threads as it want
- There is no context switch when sending a message to a KPI
 - For example, when a RFS (Request for Service) message is sent to a File System KPI, this does not request a context switch
 - Hence run-time overhead is minimum
 - Further, inter KPI communication incurs minimal overhead with it consuming only very few instructions
- Lynx OS is a self hosted system – wherein development can be done in the same system

COP 4610 – Operating System Principles

31

31

VxWorks

- Created by Wind River.
- Current Version: VxWorks 7
- VxWorks is the most established and most widely deployed device software operating system
- Currently there are more than 300 million devices that are VxWorks enabled
- The core attributes of VxWorks, include high performance, reliability, determinism, low latency and scalability

COP 4610 – Operating System Principles

32

32

VxWorks

- Enhanced error management
- Backward compatibility to previous version features
- Extensive POSIX 1003.1, .1b, .1c compatibility (including Pthreads)
- Scheduling: preemptive priority with round robin (supports both real-time and non-real-time tasks)
- Multi-processor support
- Shell for user interface
- Extensive debugging and performance monitoring capabilities
- Mars Exploration Rovers *Spirit* and *Opportunity* and the Mars Reconnaissance Orbiter use the VxWorks operating system

33

RTLinux

- Available as a patch to the regular Linux kernel
- Provides a real-time API for developers
- RTLinux is a hybrid OS that runs a Linux kernel as an idle thread (lowest priority) of the real-time kernel
- Predictable delays (small size, limited operations)
- Finer timer resolution.
- RT kernel and RT applications are kept as simple as possible and non-time critical applications (GUIs, file systems) are handled by the standard Linux

34

RTLinux

- Real time threads and interrupt handlers never delayed by non-real-time operations
- Preemptible kernel
 - Its routines are very small and fast; limit delays
 - Interrupts from Linux are disabled
- RT-Linux has many kinds of schedulers
 - FIFO
 - Used to pass information between real-time process and ordinary Linux process
 - Designed to never block the real-time task
 - EDF
 - RMS

COP 4610 – Operating System Principles

35

35

RTLinux

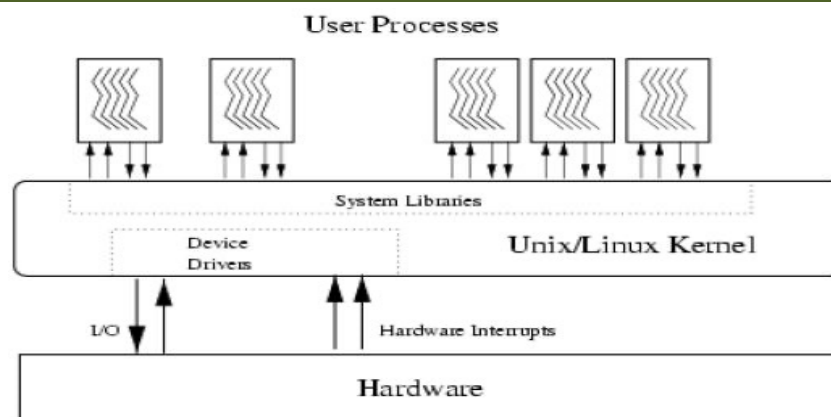


Figure 1.1: Detail of the bare Linux kernel

COP 4610 – Operating System Principles

36

36

RTLinux Kernel

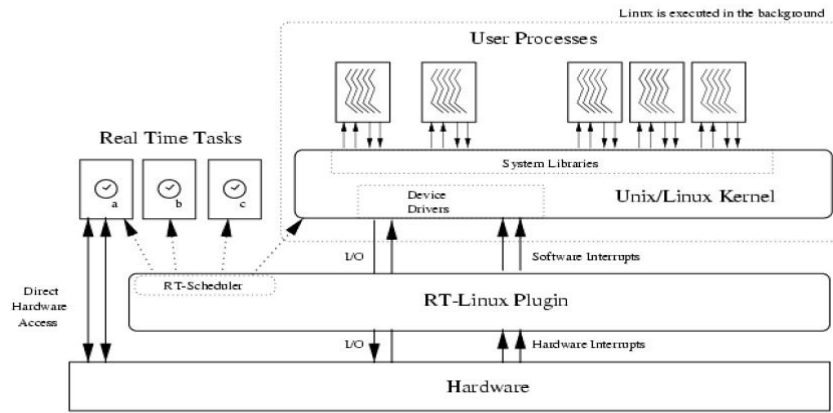


Figure 1.2: Detail of the RTLinux kernel