Slide 1

OVERVIEW

Last Week:

- ➜ Scheduling Algorithms
- ➜ Real-time systems

Today:

- ➜ Yet another real-time scheduling algorithm
- ➜ Case studies
  - Changes in the Linux kernel
  - Real-time operating systems
  - Windows 2000: Scheduling, VM
- ➜ Overview

Next Week:

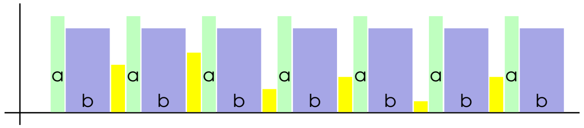
- ➜ Q & A session: send me a list of topics you would like me to explain again

Slide 2

Problem:

- ➜ in real life applications, many tasks are not always periodic.
- ➜ static priorities may not work

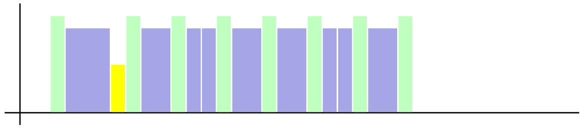
If real time threads run periodically with same length, fixed priority is no problem:



- a: periodic real time thread, highest priority
- b: periodic real time thread
- various different low priority tasks (e.g., user I/O)

Slide 3

But if frequency of high priority task increases temporarily, system may encounter overload:



- system not able to respond
- system may not be able to perform requested service

Slide 4

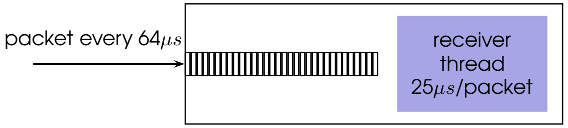
Example: (from *Scheduling Sporadic Events*, Lonni Vanzandt)

Network interface control driver, requirements:

- ➜ avoid if possible to drop packets
- ➜ definitely avoid overload

If receiver thread get highest priority permanently, system may go into overload if incoming rate exceeds a certain value.

- ➜ expected frequency: packet once every 64μs
- ➜ CPU time required to process packet: $25\mu s$
- ➜ 32-entry ring buffer, max 50% full



SPORADIC SCHEDULING

POSIX standard to handle

- aperiodic or sporadic events
- with static priority, preemptive scheduler

Implemented in hard real-time systems such as QNX, some real-time versions of Linux, real-time specification for Java (RTSJ)(partially)

Can be used to avoid overloading in a system

Basic Idea: "simulation" of periodic behaviour of thread by assigning

- realtime priority: P_r
- background priority: P_b
- execution budget: E
- replenishment interval: R

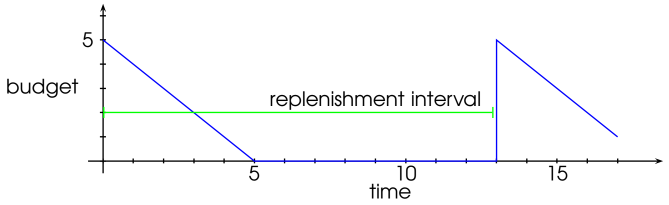
to thread.

- Whenever thread exhausts execution budget, priority is set to background priority P_b
- When thread blocks after n units, n will be added to execution budget R units after execution started
- When execution budget is incremented, thread priority is reset to P_r

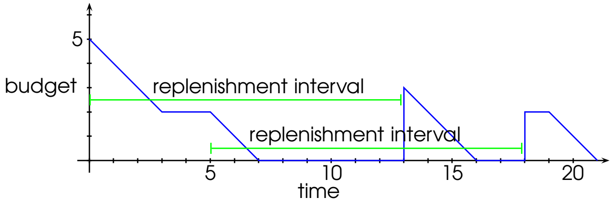
Example:

- execution budget: 5
- replenishment interval: 13

Thread does not block:



Thread blocks:

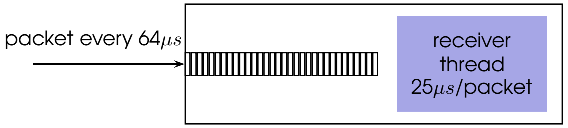


(0) exection starts, 1st replenishment interval starts
(3) thread blocks
(5) continues execution, 2nd replenishment interval starts
(7) budget exhausted
(13) budget set to 3, thread continues execution
(16) budget exhausted
(18) budget set to 2
(19) thread continues execution

Slide 9

Example: Network interface control Driver

- ➜ use expected incoming rate and desired max CPU utilisation of thread to compute execution budget and replenishment period
- ➜ if no other threads wait for execution, packets can be processed even if load is higher
- ➜ otherwise, packets may be dropped



- ➜ period: $64\mu s$ * 16 = $1024\mu s$
- ➜ execution time: $25\mu s$ * 16 = $400\mu s$
- ➜ CPU load caused by receiver thread: 400/1024 = 0.39, about 39%

Slide 10

HARD REAL TIME OS

We look at examples of three types of systems:

- ➜ real-time support in a general purpose operating system
- ➜ configurable hard real time systems
  - system designed as real time OS from the start
- ➜ hard real-time variants of general purpose OSs
  - try to alleviate shortcomings of OS with respect to real time apps

Slide 11

REAL-TIME SUPPORT IN LINUX 2.4.

- ➜ Scheduling:
  - POSIX `SCHED_FIFO`, `SCHED_RR`,
- ➜ Virtual Memory:
  - no VM for real-time apps
  - `mlock()` and `mlockall()` to switch off paging (which other applications might need to do this?)
- ➜ Timer: resolution: 10ms, too coarse grained for real-time apps

Slide 12

IMPROVEMENTS IN 2.6 KERNEL

- ➜ Kernel Preemption
  - kernel code laced with preemption points
  - calling process can block and thereby yield CPU to higher-priority process
- ➜ Kernel can be built without VM
- ➜ Improved scheduler
- ➜ Timer resolution: 1ms

Slide 13

SCHEDULING IN 2.4 AND 2.6: COMPARISON

2.4:

- ➜ CPU time divided into epochs
- ➜ Each process has a (poss. different) time quantum it is allowed to run in every epoch
- ➜ Epoch ends when all runnable processes have exhausted their quantum
- ➜ Time quantum for each process recomputed after every epoch
- ➜ To find the next process which should be scheduled, the complete ready-queue has to be scanned
- ➜ SMP: only single ready-queue
- ➜ $\mathcal{O}(n)$ algorithm: overhead grows linearly with number of PE's
- ➜ Ready queue access bottle neck for SMP

Slide 14

2.6:

- ➜ Queue for each priority
- ➜ Thread can be in active (quantum not yet expired) or expired (quantum already used up) queue.
- ➜ Priority is re-calculated after quantum is expired
- ➜ Interactive processes inserted back into active-queue
- ➜ SMP: One set of queue per processor, idle processors steal work from other processors
- ➜ $\mathcal{O}(1)$ algorithm: time required for scheduling decision does not depend on number of processes
- ➜ Ready queue access not a bottle neck for SMP
- ➜ Better locality

Slide 15

RTLINUX

- ➜ abstract machine layer between actual hardware and Linux kernel
- ➜ takes control of
  - hardware interrupts
  - timer hardware
  - interrupt disable mechanism
- ➜ real time scheduler runs with no interference fron Linux kernel
- ➜ programmer must utilise RTLinux API for real time applications

Slide 16

QNX

- ➜ Microkernel based architecture
- ➜ POSIX standard API
- ➜ Modular — can be costumised for very small size (eg, embedded systems) or large systems
- ➜ Memory protection for user applications and os components

Scheduling:

- ➜ FIFO scheduling
- ➜ Round-robin
- ➜ Adaptive scheduling
  - thread consumes its timeslice, its priority is reduced by one
  - thread blocks, it immediately comes back to its base priority
- ➜ POSIX sporadic scheduling

Slide 17

Kernel Services:

- Thread services: provides the POSIX thread creation primitives.
- Signal services: provides the POSIX signal primitives.
- Message passing services: handles the routing of all messages between all threads through the whole system.
- Synchronization services: provides the POSIX thread synchronization primitives.
- Scheduling services: schedules threads using the various POSIX realtime scheduling algorithms.
- Timers services: provides the set of POSIX timer.

Slide 18

Process Manager:

The process manager is capable of creating multiple POSIX processes (each of which may contain multiples POSIX threads). Its main areas of responsability include:

- Process management: manages process creation, destruction, and process attributes such us user ID and group ID.
- Memory management: manages memory protection, shared libraries, and POSIX shared memory primitives.
- Pathname management: manages the pathname space (mountpoints).

Slide 19

WINDOWS CE 5.0

Componentised OS designed for embedded systems with hard real-time support

- handles nested interrupts
- handles priority inversion based on priority inheritance

Offers

- guaranteed upper bound on high priority thread scheduling
- guaranteed upper bound on delay for interrupt service routines

Slide 20

WINDOWS 2000 CASE STUDY

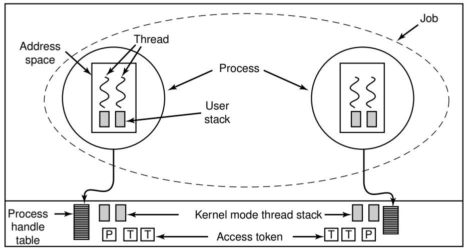
- Scheduling
- Virtual Memory Management

Slide 21

WINDOWS 2000 SCHEDULING

- ➜ priority driven, preemptive scheduling system
- ➜ SMP: set of processors a thread can run on may be restricted (processor affinity)
- ➜ scheduling decision may be necessary when
  - a new thread has been created
  - a thread released from wait state
  - time quantum of a thread is exceeded
  - a thread's priority changes
  - processor affinity of a thread changes
- ➜ no dedicated scheduler thread — each thread chooses successor while running in kernel mode

Slide 22



Slide 23

- ➜ if thread with higher priority becomes ready to run, current thread is preempted
- ➜ scheduled at thread granularity
  - processes with many threads get more CPU time

Slide 24

WINDOWS 2000 SCHEDULING

- ➜ Windows 2000 priority levels:
  - ➜ 0 (zero-page thread)
  - ➜ 1-15 (variable levels)
  - ➜ 16-31 (realtime levels — soft)
- ➜ Win32 API priority classes:
  - Real-time
  - High
  - Above Normal
  - Normal
  - Below Normal
  - Idle

  and relative priorities within these classes:

  - Time-critical
  - High
  - . . .

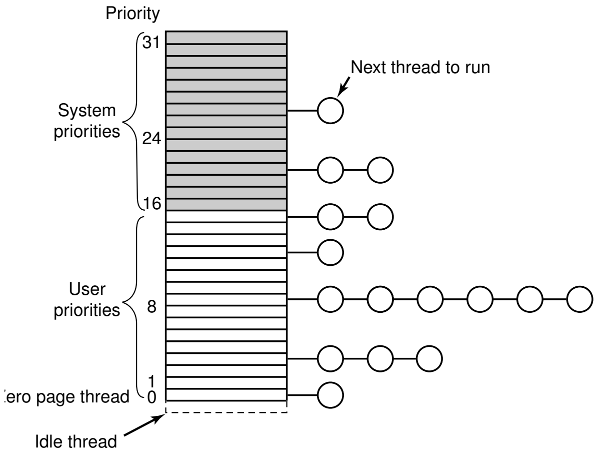
Slide 25

- ➜ each thread has a quantum value, clock-interrupt handler deducts 3 from running thread quantum
- ➜ default value of quantum: 6 Windows 2000 Professional, 36 on Windows 2000 Server
- ➜ most wait-operations result in temporary priority boost, favouring IO-bound threads
- ➜ priority of a user thread can be raised (eg, after waiting for a semaphore etc), but never above 15
- ➜ no adjustments to priorities above 15

Slide 26

		Win32 process class priorities					
		Realtime	**High**	**Above Normal**	**Normal**	**Below Normal**	**Idle**
Win32 thread priorities	**Time critical**	31	15	15	15	15	15
	Highest	26	15	12	10	8	6
	Above normal	25	14	11	9	7	5
	Normal	24	13	10	8	6	4
	Below normal	23	12	9	7	5	3
	Lowest	22	11	8	6	4	2
	Idle	16	1	1	1	1	1

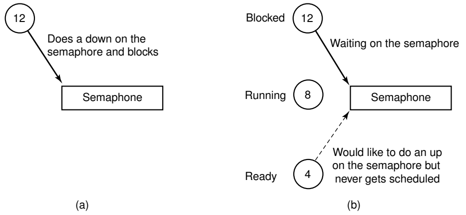
Slide 27



Slide 28

DEALING WITH PRIORITY INVERSION IN WINDOWS 2000

Example: Producer-Consumer problem

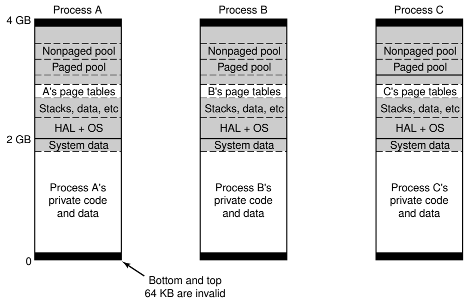


- ➜ System keeps track of how long a ready-thread has been in the queue
- ➜ if waiting time exceeds threshold, priority boosted to 15

Slide 29

MEMORY MANAGEMENT

- ➜ Every process has 4GB virtual address space



Slide 30

MEMORY MANAGEMENT

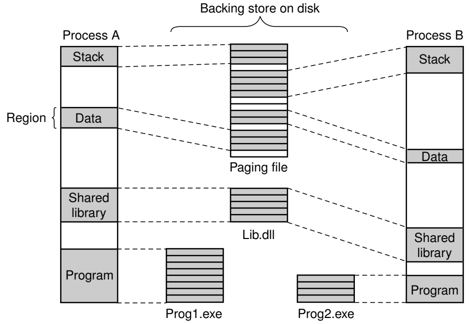
- ➜ A page can be in one of three states:
  - free: not in use, reference to such a page causes a page fault
  - committed: data or code mapped onto the page. If not in main memory, page fault occurs, OS swaps page from disk
  - reserved: not yet mapped, but also not available. Used, for example, to implement thread stacks

  and has the usual readable, writable, executable attributes

Slide 31

MEMORY MAPPED FILES

- ➜ memory mapped filed supported
- ➜ processes may share maps, updates visible to all processes
- ➜ if file is opened for normal reading, current version is shown
- ➜ copy-on-write (cow)



Slide 32

WIN32 API FOR VM

Win32 API function	Description
VirtualAlloc	Reserve or commit a region
VirtualFree	Release or decommit a region
VirtualProtect	Change the read/write/execute protection on a region
VirtualQuery	Inquire about the status of a region
VirtualLock	Make a region memory resident (i.e., disable paging for it)
VirtualUnlock	Make a region pageable in the usual way
CreateFileMapping	Create a file mapping object and (optionally) assign it a name
MapViewOfFile	Map (part of) a file into the address space
UnmapViewOfFile	Remove a mapped file from the address space
OpenFileMapping	Open a previously created file mapping object

MEMORY MANAGEMENT

- ➜ Unlike scheduler, who deals with threads and ignores processes, MM deals only with processes
- ➜ Mapping of pages happens in the usual way, two-level page table used
- ➜ In case of a page fault, a block of consecutive pages are read

PAGE REPLACEMENT ALGORITHM

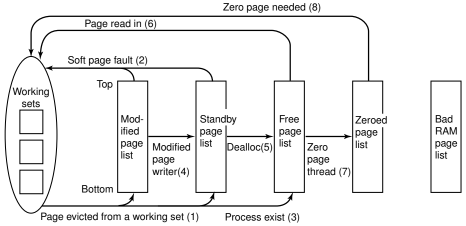
Working Set:

- ➜ set of pages of a process which have been mapped into memory
- ➜ described by (process specific) max and min size
- ➜ all processes start with the same limits, but may change over time
- ➜ not hard bounds
- ➜ if page fault occurs and process has
  - less than min pages: add page
  - between min and max pages: add page if memory is not scarce
  - more than max pages: evict page from working set
- ➜ Working set of system is handled separately.

DAEMON THREADS TO MANAGE WORKING SETS

- ➜ Balance Set Manager: checks whether there are enough free pages, starts Working Set Manager if required
- ➜ Working Set Manager: searches for processes which have exceeded their maximum, didn't have page faults recently and removes some of their pages

A closer look at the free frames management:

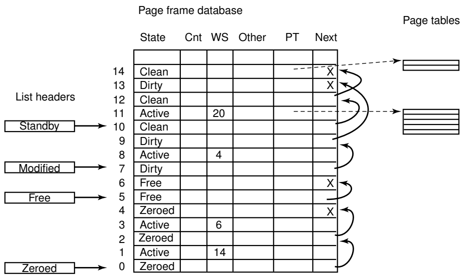


There are actually four separate lists which contain free frames

① Modified Pages
② Standby Pages
③ Free Pages
④ Zeroed Pages

Slide 37

A closer look at the free frames management:



	State	Cnt	WS	Other	PT	Next
14	Clean					X
13	Dirty					X
12	Clean					
11	Active		20			
10	Clean					
9	Dirty					
8	Active		4			
7	Dirty					
6	Free					X
5	Free					
4	Zeroed					X
3	Active		6			
2	Zeroed					
1	Active		14			
0	Zeroed					