Process and Thread Management

„Ausgewählte Betriebssysteme“
Professur Betriebssysteme
Fakultät Informatik

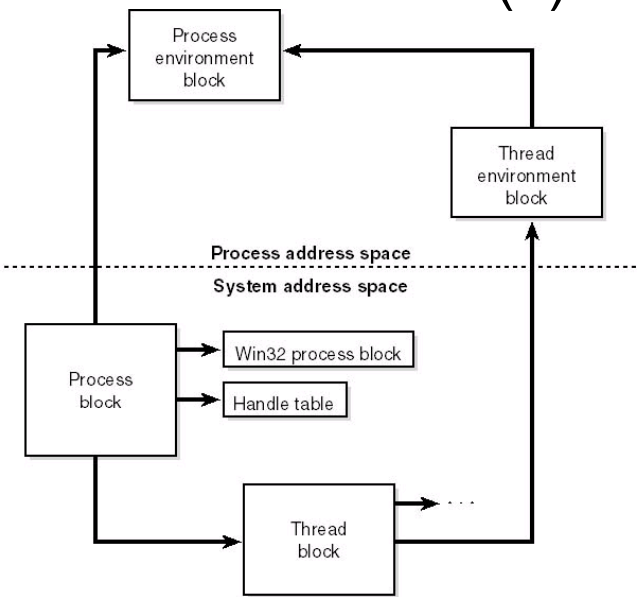
Outline

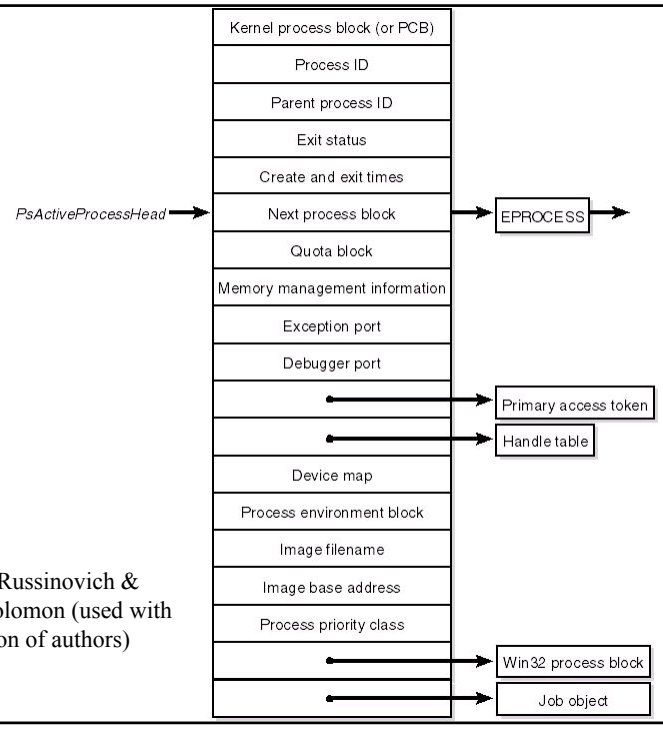
- Data Structures
- Process Creation
- Thread Creation
- Scheduling

Data Structures

- Process represented by EPROCESS (executive process) block
- Thread represented by ETHREAD block
- One process contains at least one thread
- EPROCESS and ETHREAD plus associated structures exist in system address space
- Process and thread environment blocks (PEB, TEB) exist in user address space

Data Structures (2)



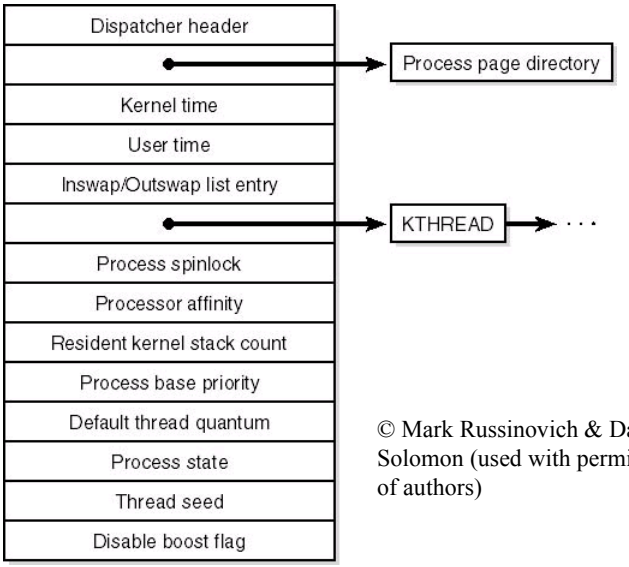


5

Kernel Process Block (PCB)

- Contains basic information needed to manage processes and their threads
  - Page directory
  - Kernel thread block list
  - State
  - Etc.

Kernel Process Block (2)

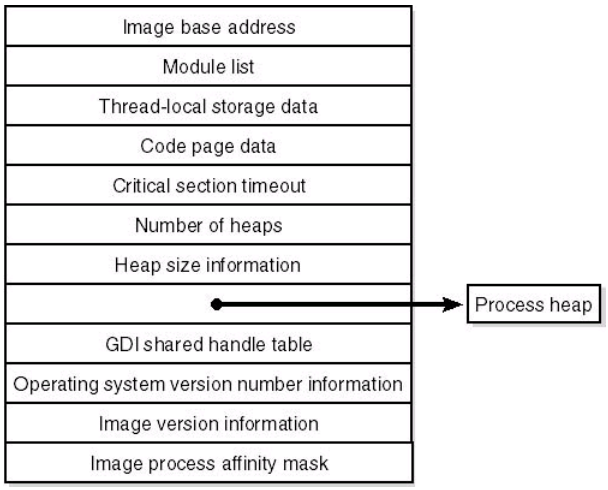


7

Process Environment Block

- Contains information for:
  - Image loader
  - Heap manager
  - Other Win32 system DLLs
- Always mapped at 0x7ffdf00

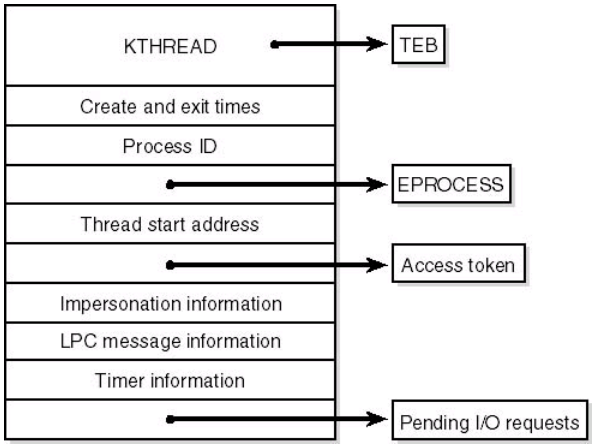
Process Environment Block (2)



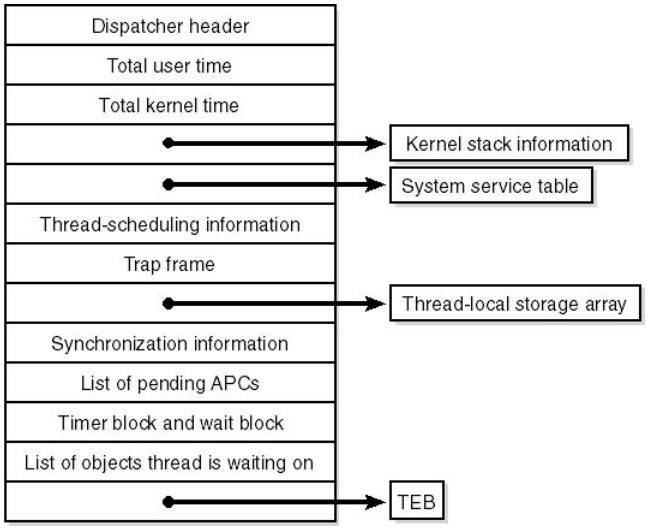
Thread Control Block

- Kernel object used to schedule threads
- Win32 subsystem maintains parallel structure for each Win32 thread
- Kernel-mode portion of Win32 subsystem maintains parallel structure for thread using GUI or USER functions
- Fibers:
  - Managed in user-mode by Win32 subsystem
  - First fiber created from thread
  - Successive fiber created explicitly
  - Executed if called through *SwitchToFiber*

Thread Control Block (2)



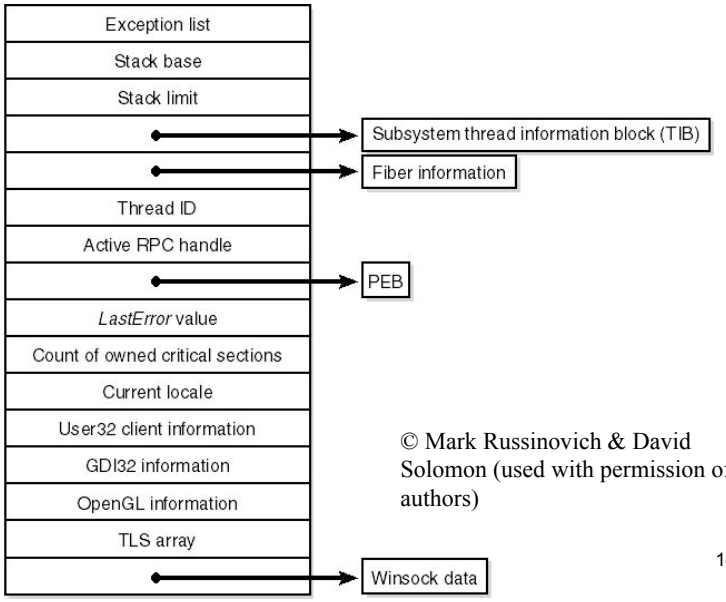
Kernel Thread Block



Kernel Thread Block (2)

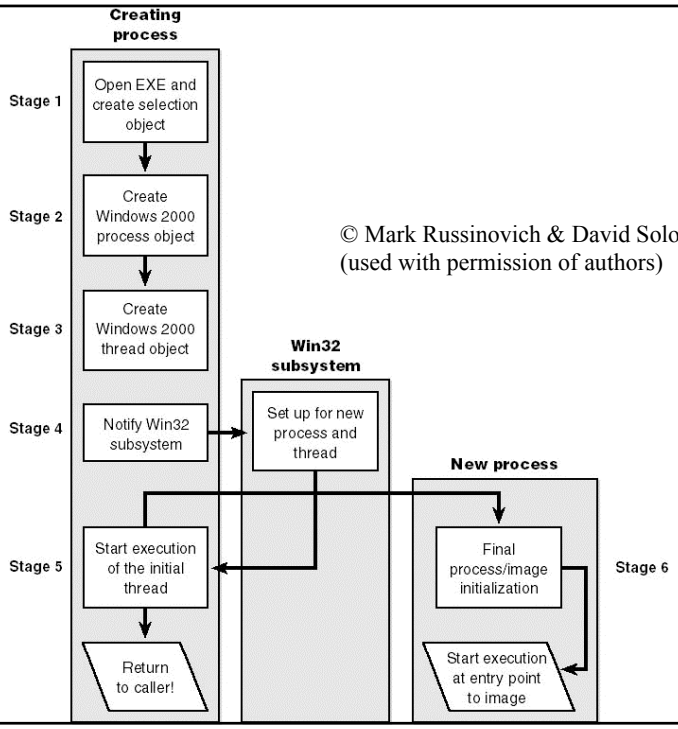
- Dispatcher header: thread can be waited on, need dispatcher object
- System service dispatch table (see Introduction – 20)
- Scheduling information: base priority, current priority, quantum, affinity mask, ...
- Pending APCs (see Interrupts – 17)
- ...

Thread Environment Block



Outline

- Data Structures
- Process Creation
- Thread Creation
- Scheduling



Create a Process

1. Open image file
2. Create EPROCESS object
3. Create initial thread (stack, context, ETHREAD)
4. Notify Win32 subsystem (set up for new process and thread)
5. Start execution of initial thread (unless CREATE_SUSPENDED)
6. In context of new process and thread:
   - Complete initialization of address space (load DLLs, ...)
   - Begin execution of program

Assign Priority to Process

- Can specify more than one priority class in Create Process call → lowest used
- If no priority class specified *Normal* is used, unless priority class of parent is *Idle* or *Below Normal*
- If *Real-time* is specified and parent doesn‘t have Increase Scheduling Priority privilege *High* is used

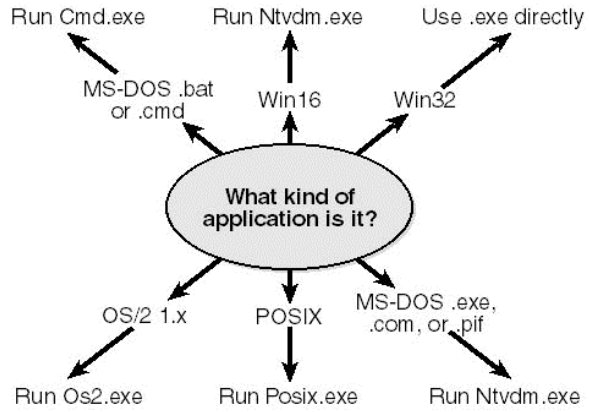
Opening an Image (stage 1)

- If image is OS/2 application, image changes to os2.exe and restart
- If image is MS-DOS application
  - Check for running Virtual Dos Machine (VDM)
  - If exists, use it to run application
  - If not, image is changed to `ntvdm.exe` an restart
- If image has `.bat` or `.cmd` extension image changed to `cmd.exe` and restart
- If image is Win16, decide if has to create new VDM (flags of Create Process call)

Opening an Image (2)

- Now valid Win2K exe is opened and a section object exists for it (not mapped yet)
- If image is Posix, image changes to Posix.exe and restart
- If image is DLL Create Process fails
- Now check registry for entry (if exists use specified image and restart) – see example

Choosing Win32 image



Create EPROCESS Object

- Set up EPROCESS block (quota, process ID, access token)
- Creating initial process address space
  - Create three initial pages for page directory, hyperspace page, working set list
  - Page table pages for non-paged system space and system cache are mapped into process
- Initialize kernel process block (pointers)

Create EPROCESS Object (2)

- Conclude address space setup
  - VM manager initializes internal data structures
  - Ntdll.dll mapped
- Set up PEB
  - Image base address, heap variables, version numbers, ...
- Complete Setup
  - Initialize handle table
  - Set flags from image file in PEB

Notify Win32 Subsystem

- Duplicate handles for process and thread
- Set priority class
- Csrss process block is allocated
- Exception port is set to Win32 subsystem‘s exception port
- Debug port is set to subsystem‘s debug port
- Csrss thread block allocated and initialized and queued in process thread list
- Subsytem internal counters incremented

Initialize in Context of Process

- Initialize loader, heap manager, critical section structures, ...
- Load required DLLs
- If debugged, suspend all threads and attach to debugger (via debug port)
- Start execution

Outline

- Data Structures
- Process Creation
- Thread Creation
- Scheduling

Create a Thread

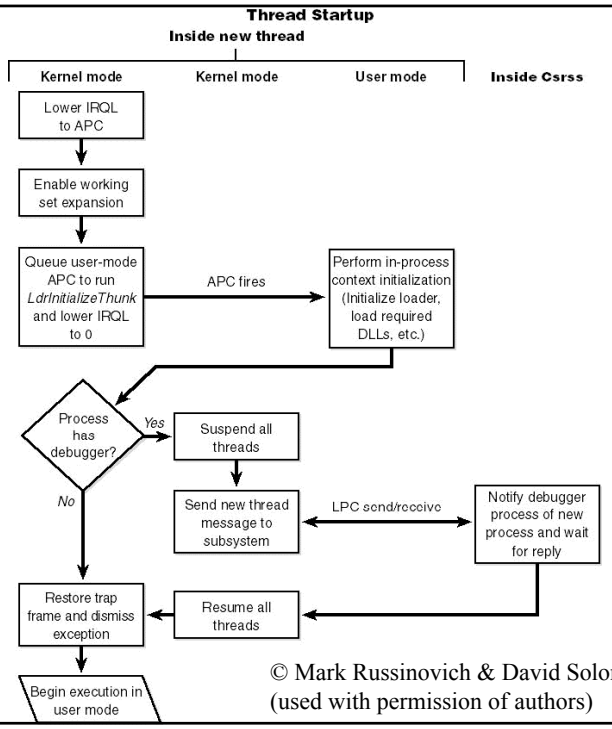
1. Create user-mode stack
2. Initialize thread‘s hardware context
3. Create executive object (kernel) and initialize it (access token, ID, kernel stack, priority, ...)
4. Win32 subsystem is informed about thread
5. Thread handle and ID returned to caller
6. When running finish initialization: register with DLLs and debugger, etc.

Create Thread (2)

- to 3.:
  - Thread count in process is incremented
  - Executive thread block created (ETHREAD)
  - Thread ID created
  - Thread‘s kernel stack allocated
  - TEB is set up in user-mode address space
  - Thread start address stored on kernel stack
  - Set up KTHREAD block (priority, affinity, quantum, machine-dependant hardware context (trap, exception frames, ...), ...)
  - Thread access token set to process‘ access token

Create Thread (3)

- to 6.:
  - lower IRQL to APC → system init thread routine fires
  - enable working set expansion and start loader initialization
  - call loaded DLLs to notify of new thread
  - If debugger is attached suspend all threads and call debugger
  - main thread begins execution in user mode (use trap, which has been initialized earlier)



© Mark Russinovich & David Solomon (used with permission of authors)

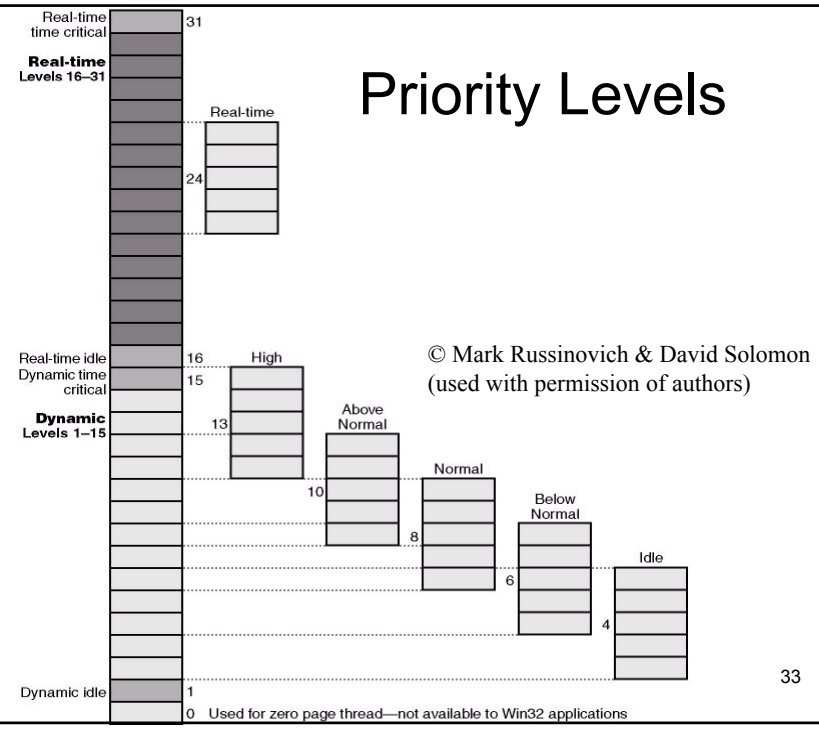
Outline

- Data Structures
- Process Creation
- Thread Creation
- Scheduling

Thread Scheduling

- Priority driven and preemptive Scheduling
- Runs for amount of time called quantum
- Can be restricted to subset of processors (processor affinity)
- „Scheduler“ spread throughout kernel (scheduling code called dispatcher)
- Dispatching occurs at DPC/dispatch level
- Dispatching triggered by:
  - Thread becomes ready (create, return from wait)
  - Thread leaves running state (terminate, quantum end, ...)
  - Thread‘s priority changes
  - Processor affinity of running thread changes

Priority Levels

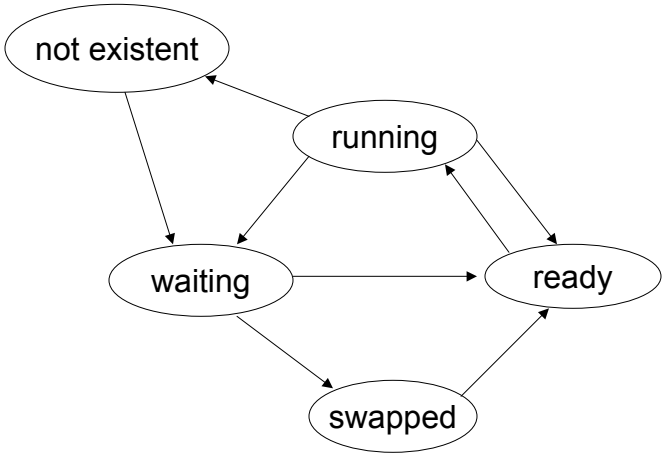


33

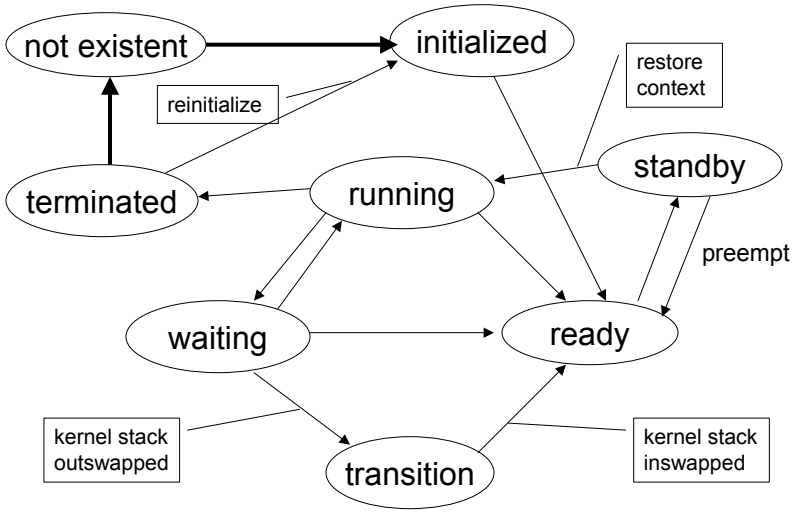
Priority Levels (2)

- Thread priority is based on priority class and relative priority (called base and current priority)
- Some system processes have priority slightly higher than *Normal* (default)
- Real-time thread never have priority changed (kernel mode system threads use this range)
- Real-time threads have their quantum reset when preempted
- Kernel-mode threads may raise IRQL to 1 (APC)
- On multi-processor system spin lock used for synchronization of dispatcher data

Thread states



Thread States



Quantum

- Thread starts with quantum of 6 (Professional) or 36 (Server)
- On each clock interrupt 3 is subtracted from quantum (if 0 another thread is scheduled)
- Even if not running (IRQL >= DPC level) quantum is reduced (wait)
- Threads at priority < 14 and wait have quantum reduced by 1 when returning from wait
- Threads at priority >= 14 and wait have quantum reset when returning from wait
- Clock interrupt on single-processor x86 about 10ms (multi-processor x86 15ms)

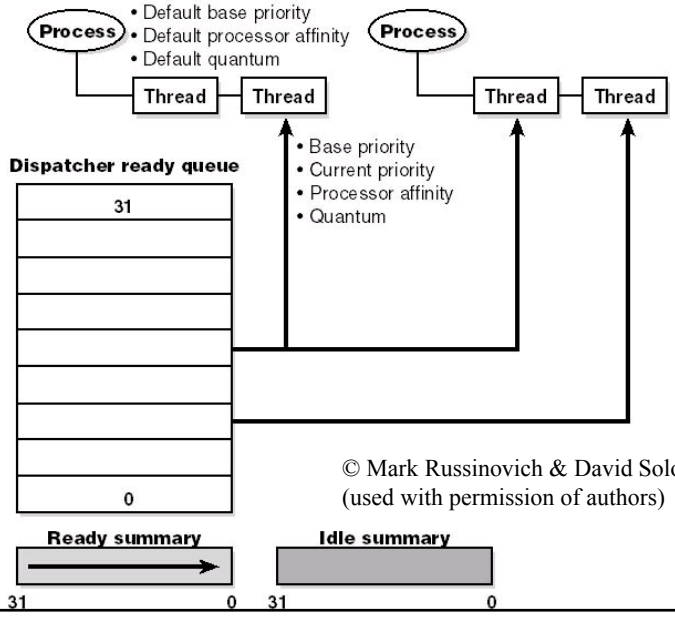
Quantum boost

- Foreground window‘s threads have quantum boosted
- Better than priority boost, because background processes still run
- Value determined by registry; can vary between 6 to 36

Scheduling Data Structures

- Dispatcher database:
  - Which threads are waiting
  - Which threads are running
  - Which processes are executing which threads
- Dispatcher ready queue:
  - One queue per priority
  - Bit mask, which priority has ready threads
- Idle summary:
  - Which processor is idle

Scheduling Data Structures (2)



Scheduling Scenarios

- Voluntary Switch:
  - Wait for object (semaphore, event, I/O completion, ...)
  - Thread goes to wait queue
- Preemption
  - Higher priority thread‘s wait completes
  - Thread priority is changed
  - Running thread is put at head of it‘s priorities ready queue
- Quantum end
  - Thread moves to tail of ready queue
- Termination

Priority Boost

- Can boost priority in five cases:
  - On I/O completion (boost defined by driver)
  - After waiting on events or semaphores (boosted by 1)
  - After foreground thread completes wait
  - When GUI threads wake up (boosted by 2)
  - When ready thread hasn‘t run for some time (ready and not run for > 300 clock intervals ~3-4 seconds; boosted to 15 + double quantum)

Priority Boost (2)