Unit 5: System Mechanisms

5.1. Object Manager, Trap Dispatching, Synchronization

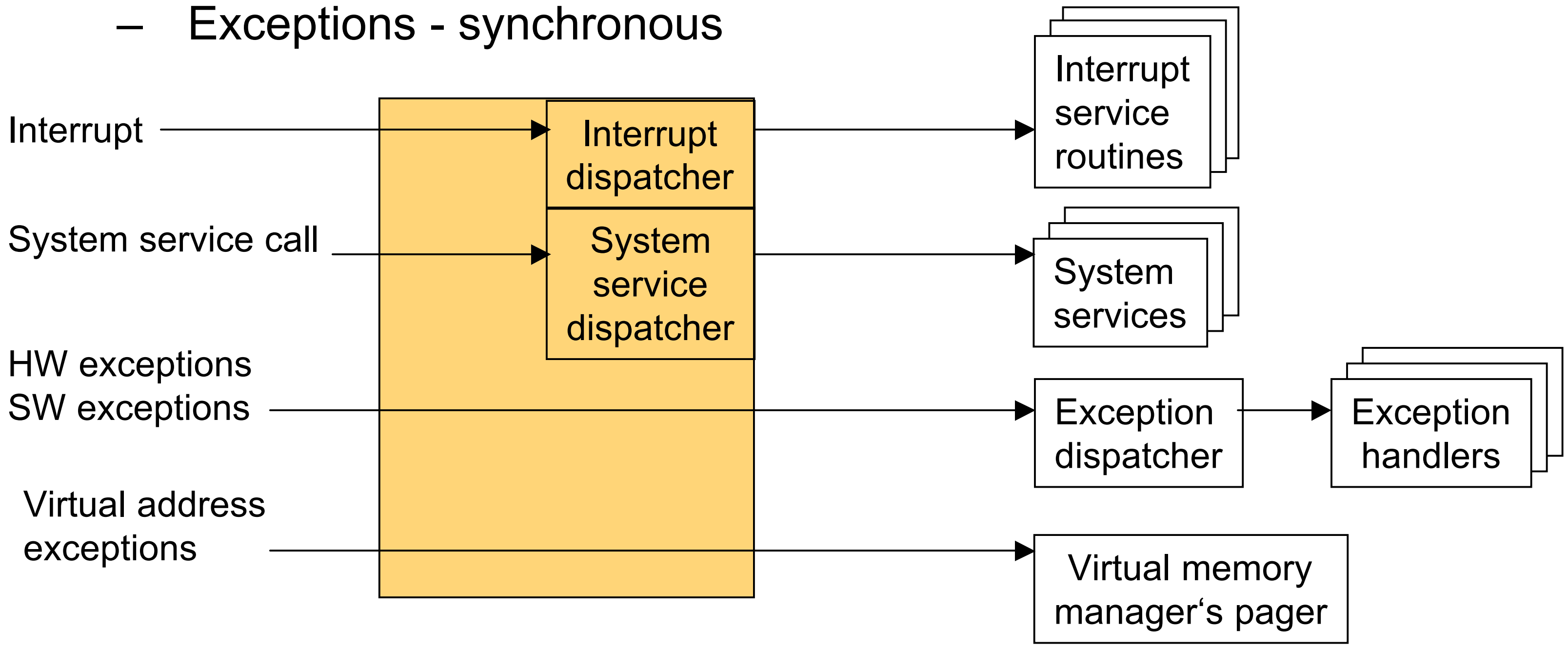
System Mechanisms

- Trap dispatching,
  - interrupts,
  - deferred procedure calls (DPCs),
  - asynchronous procedure calls (APCs)
  - Exception dispatching,
  - system service dispatching
- Executive object manager
- Synchronization, spinlocks, kernel dispatcher objects
- Local procedure calls

Trap dispatching

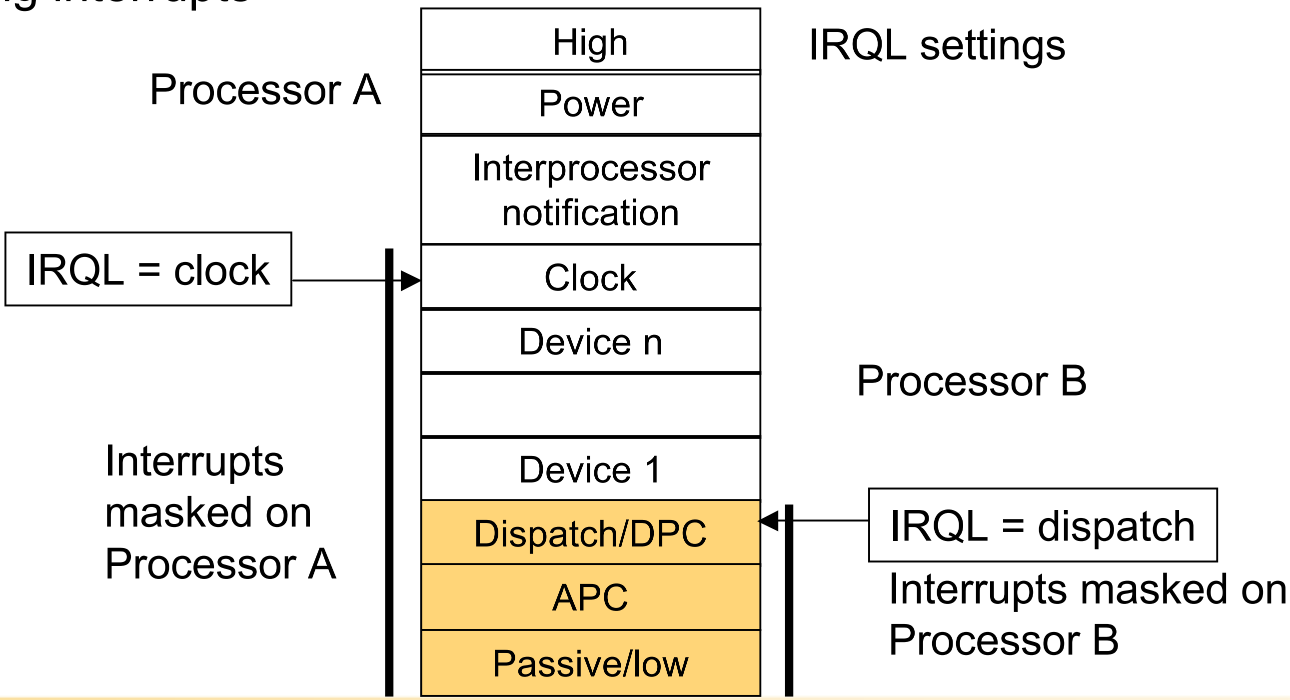
Trap: processor's mechanism to capture executing thread

- Switch from user to kernel mode
- Interrupts – asynchronous
- Exceptions - synchronous



Interrupt dispatching

- HW interrupts are mapped to *interrupt request levels* (IRQLs)
  - Synchronize access to kernel-mode data structures
  - Masking interrupts



Interrupt processing

- Interrupt dispatch table (IDT)
  - Links to interrupt service routines
- x86:
  - Interrupt controller interrupts processor (single line)
  - Processor queries for interrupt vector; uses vector as index to IDT
- Alpha:
  - PAL code (Privileged Architecture Library – Alpha BIOS) determines interrupt vector, calls kernel
  - Kernel uses vector to index IDT
- After ISR execution, IRQL is lowered to initial level

Interrupt object

- Allows device drivers to register ISRs for their devices
  - Contains dispatch code (initial handler)
  - Dispatch code calls ISR with interrupt object as parameter (HW cannot pass parameters to ISR)
- Connecting/disconnecting interrupt objects:
  - Dynamic association between ISR and IDT entry
  - Loadable device drivers (kernel modules)
  - Turn on/off ISR
- Interrupt objects can synchronize access to ISR data
  - Multiple instances of ISR may be active simultaneously (MP machine)
  - Multiple ISR may be connected with IRQL

Predefined IRQLs

- **High**
  - used when halting the system (via *KeBugCheck()*)
- **Power fail**
  - originated in the NT design document, but has never been used
- **Inter-processor interrupt**
  - used to request action from other processor (dispatching a thread, updating a processors TLB, system shutdown, system crash)
- **Clock**
  - Used to update system‘s clock, allocation of CPU time to threads
- **Profile**
  - Used for kernel profiling (see Kernel profiler – Kernprof.exe, Res Kit)

Predefined IRQLs (contd.)

- **Device**
  - Used to prioritize device interrupts
- **DPC/dispatch** and **APC**
  - Software interrupts that kernel and device drivers generate
- **Passive**
  - No interrupt level at all, normal thread execution

Software interrupts

- Initiating thread dispatching
  - DPC allow for scheduling actions when kernel is deep within many layers of code
  - Delayed scheduling decision, one DPC queue per processor
- Handling timer expiration
- Asynchronous execution of a procedure in context of a particular thread
- Support for asynchronous I/O operations

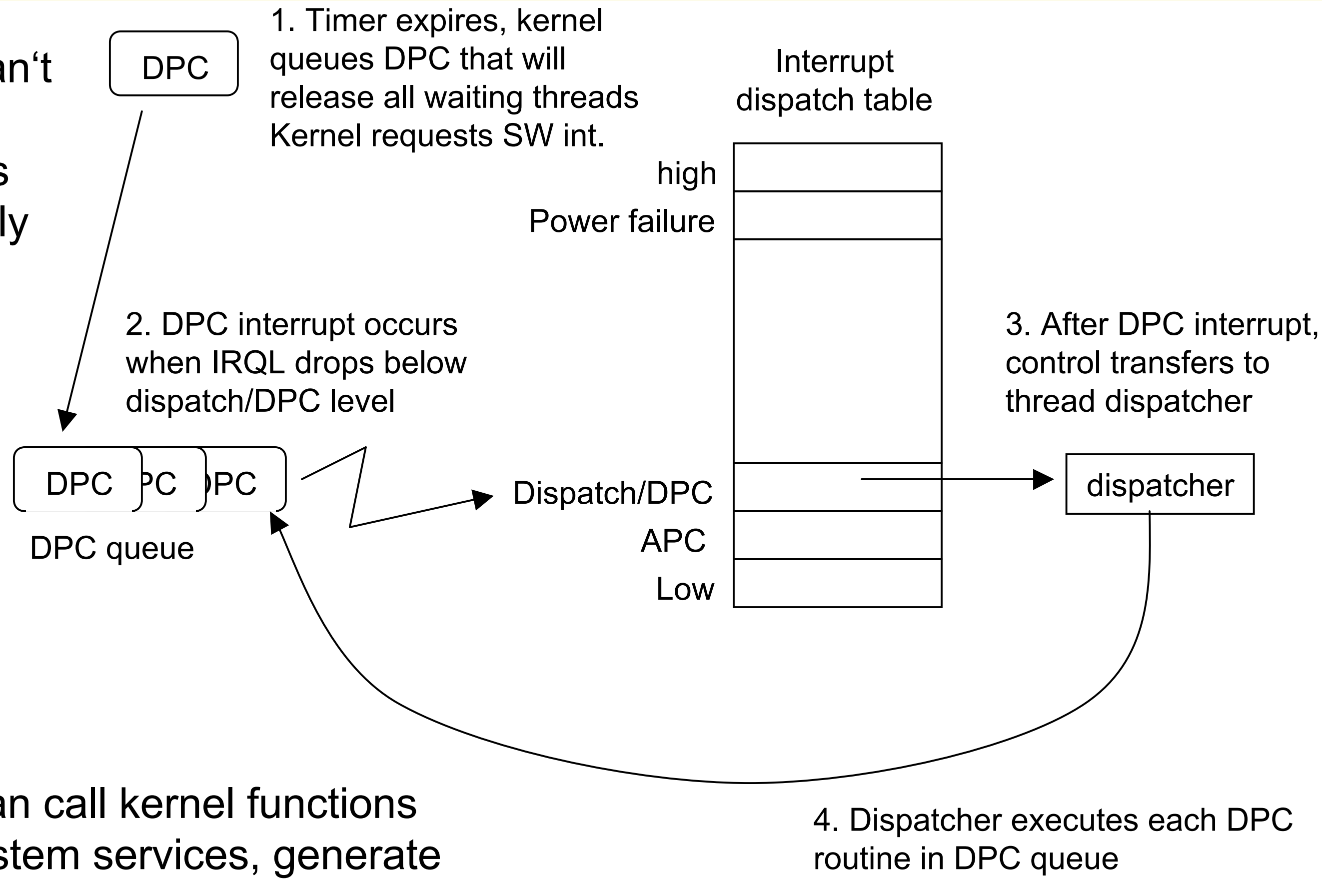
Deferred Procedure Calls (DPCs)

- DPCs provide the OS with the capability to generate an interrupt and execute a system function in kernel mode
- Kernel uses DPCs
  - To process timer expiration (and release waiting threads)
  - To reschedule processor after a thread‘s quantum expires
- Device drivers use DPCs to complete I/O requests

- DPCs are represented by DPC objects
  - Containing address of a system functions to be called
  - Waiting DPC routines are stored in per-processor DPC queues
  - DPC queues are ordered FIFO by default

Delivering a DPC

DPC routines can‘t assume what process address space is currently mapped



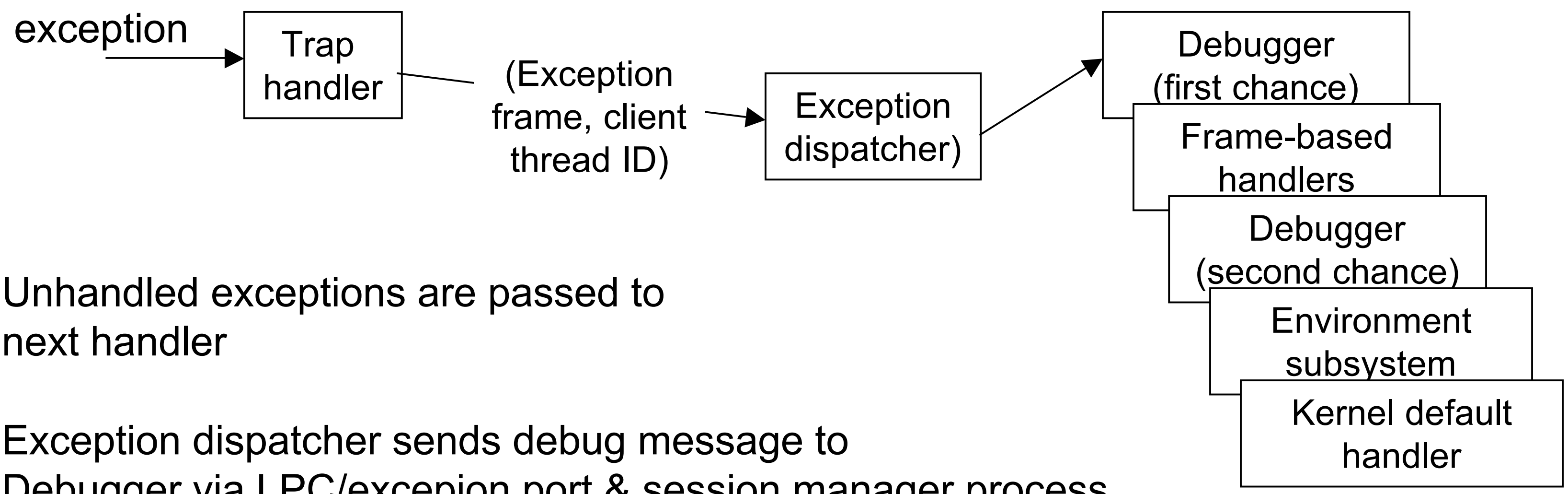
DPC routines can call kernel functions but can‘t call system services, generate page faults, or create or wait on objects

Asynchronous Procedure Calls (APCs)

- Execute code in context of a particular user thread
  - APC routines can acquire resources (objects), incur page faults, call system services
- APC queue is thread-specific
- User mode & kernel mode APCs
  - Permission required for user mode APCs
- Executive uses APCs to complete work in thread space
  - Wait for asynchronous I/O operation
  - Emulate delivery of POSIX signals
  - Make threads suspend/terminate itself (env. subsystems)
- APCs are delivered when thread is in *alertable wait state*
  - WaitForMultipleObjectsEx(), SleepEx()

Exception dispatching

- Structured exception handling;
  - Accessible from MS VC++ language: __try, __except, __finally
  - See Jeffrey Richter, „Advanced Windows“, MS Press
  - See Johnson M.Hart, „Win32 System Programming“, Addison-Wesley



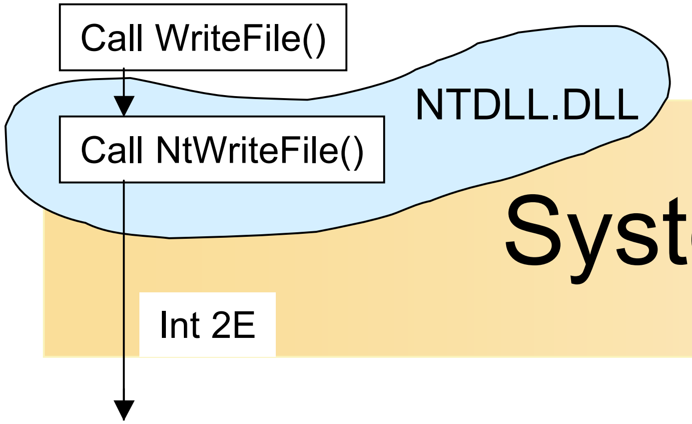
Unhandled exceptions are passed to next handler

Exception dispatcher sends debug message to Debugger via LPC/excepion port & session manager process

Internal Win32 exception handler

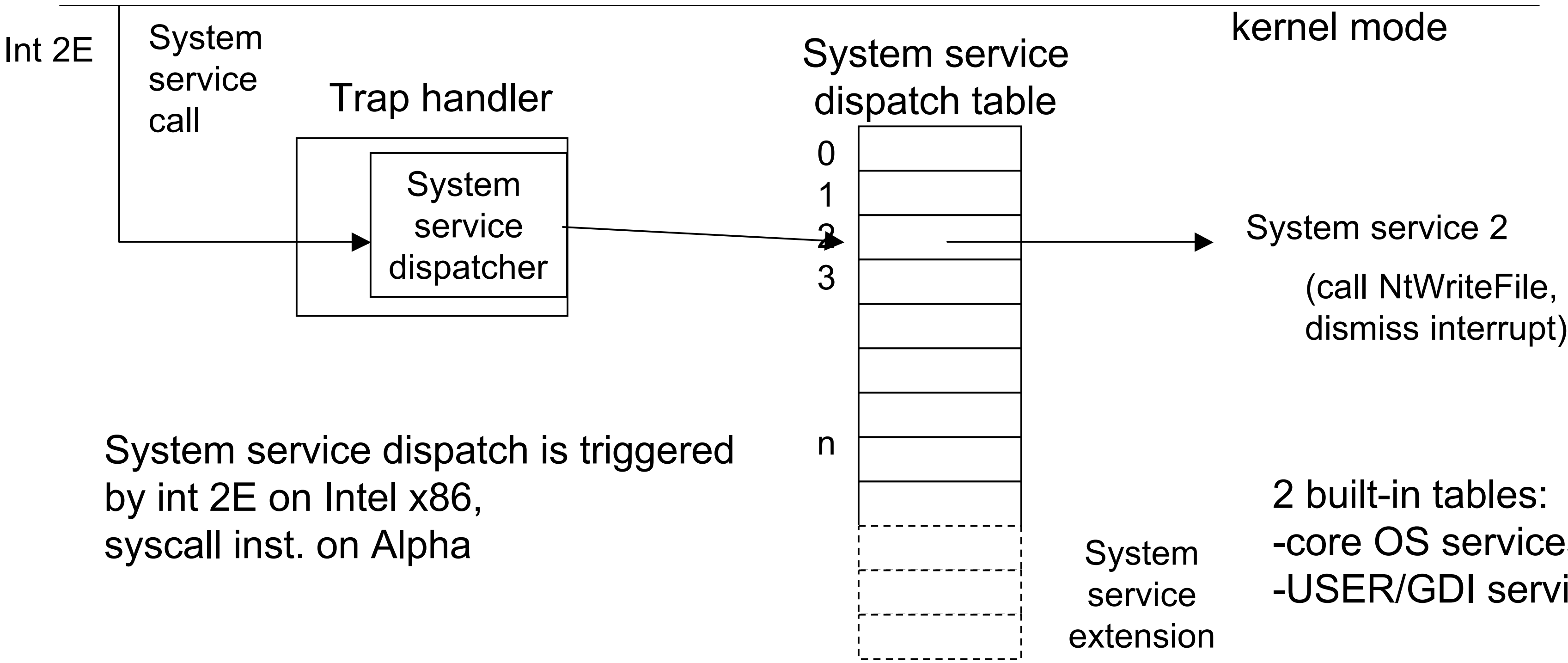
- Processes unhandled exceptions
  - At top of stack, declared in StartOfProcess()/StartOfThread()

```
void Win32StartOfProcess(LPTHREAD_START_ROUTINE lpStartAddr,
                         LPVOID lpvThreadParm) {
    __try {
        DWORD dwThreadExitCode = lpStartAddr(lpvThreadParm);
        ExitThread(dwThreadExitCode);
    } __except(UnhandledExceptionFilter(
        GetExceptionInformation())) {
        ExitProcess(GetExceptionCode());
    }
}
```



System Service Dispatching

- Kernel's trap handler dispatches interrupts, exceptions, and system service calls



System service dispatch is triggered by int 2E on Intel x86, syscall inst. on Alpha

2 built-in tables:
-core OS services
-USER/GDI services

Object Manager

- Common mechanism for using system resources
- Isolate protection to one location in OS (C2 security)
- Accounting: charge processes for resource usage
- Object naming scheme
  - Incorporate existing objects: devices, files, directories
- Support various OS personalities
  - resource inheritance (Win32, POSIX)
  - Case-sensitive filenames (POSIX)
- Uniform rules for object retention / lifecycle

Internally: executive objects and kernel objects (simpler)
Many executive objects encapsulate kernel objects

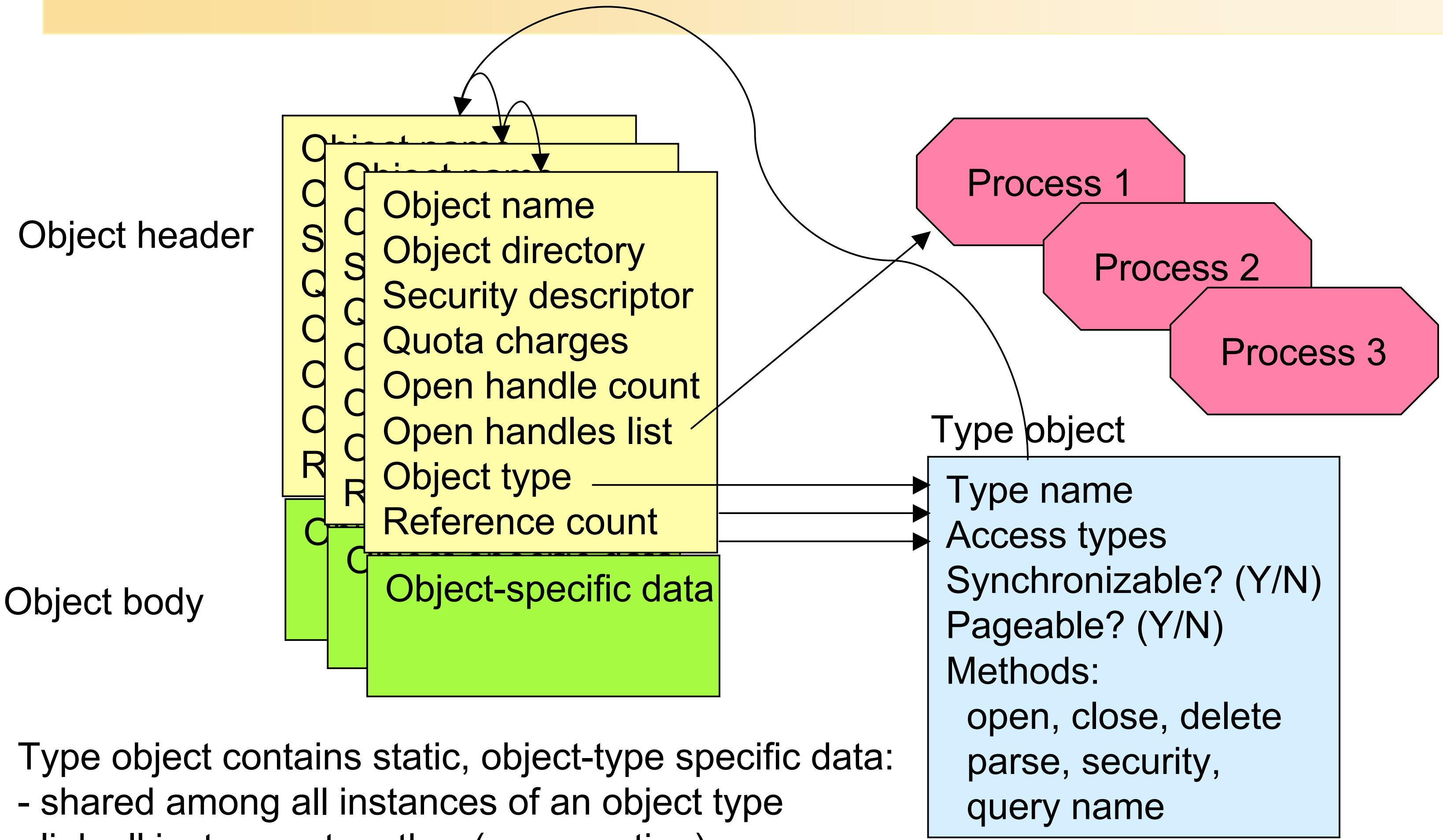
Executive Objects

Object type	Represents
Object directory	Container object for other objects: implement hierarchical namespace to store other object types
Symbolic link	Mechanism for referring to an object name indirectly
Process	Virtual address space and control information necessary for execution of thread objects
Thread	Executable entity within a process
Section	Region of shared memory (file mapping object in Win32)
File	Instance of an opened file or I/O device
Port	Mechanism to pass messages between processes
Access token	Security profile (security ID, user rights) of a process or thread

Executive Objects (contd.)

Object type	Represents
Event	Object with persistent state (signaled or not) usable for synchronization or notification
Semaphore	Counter and resource gate for critical section
Mutant	Synchronization construct to serialize resource access
Timer	Mechanism to notify a thread when a fixed period of time elapses
Queue	Method for threads to enqueue/dequeue notifications of I/O completions (Win32 I/O completion port)
Key	Reference to registry data – visible in object manager namespace
Profile	Mechanism for measuring execution time for a process within an address range

Object Structure



Type object contains static, object-type specific data:
- shared among all instances of an object type
- link all instances together (enumeration)

Object Methods

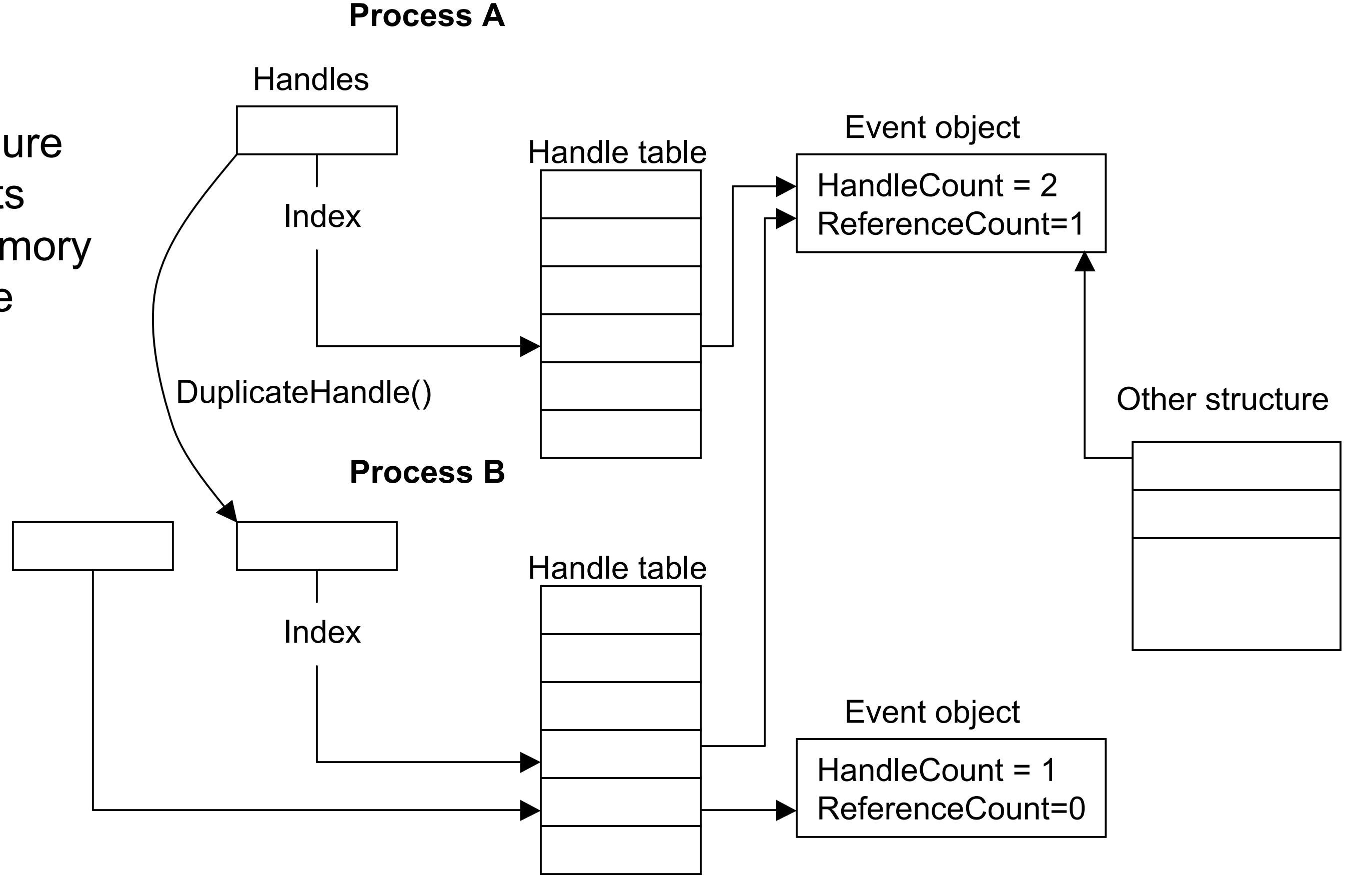
Method	When method is called
Open	When an object handle is opened
Close	When an object handle is closed
Delete	Before the object manager deletes an object
Query name	When a thread requests the name of an object, such as a file, that exists in a secondary object domain
Parse	When the object manager is searching for an object name that exists in a secondary object domain
Security	When a process reads/changes protection of an objects, such as a file, that exists in a secondary object domain

Example:

- Process opens handle to object \Device\Floppy0\docs\resume.doc
- Object manager traverses name tree until it reaches Floppy0
- Calls parse method for object Floppy0 with arg \docs\resume.doc

Object reference counting – lifecycle

Application can ensure that an object and its name remain in memory by keeping a handle to the object open



Object Names
Standard Object Directories

Directory	Types of Object Names Stored
\??	MS-DOS device names (\DosDevice is a symbolic link to this dir)
\BaseNamedObjects	Mutexes, events, semaphores, waitable timers, and section objects
\device	Device objects
\driver	Driver objects
\FileSystem	File system driver objects and file system recognizer device objects
\KnownDlls	Section names and path for known DLLs (mapped by the system at startup time)
\nls	Section names for mapped national language support tables
\ObjectTypes	Names of types of objects
\RPCControl	Port objects used by remote procedure calls (RPCs)
\security	Names of objects specific to security subsystem
\windows	Win32 subsystem ports and window stations

Wino Object Names (contd.)

- Only \BaseNamedObjects and \?? visible to user programs
- Object names are global on a single computer
  - Not visible across the network
  - Object manager‘s parse method is hook to remote objects
- I/O manager & remote files:
  - When asked to open remote file, Object Manager contact I/O manag.
  - I/O manager calls network redirector
  - Server code on remote machine calls remote object manager and I/O manager and delivers data back

Object Names (contd.)

- Only \BaseNamedObjects and \?? visible to user programs
- Object names are global on a single computer
  - Not visible across the network
  - Object manager‘s parse method is hook to remote objects
- I/O manager & remote files:
  - When asked to open remote file, Object Manager contact I/O manag.
  - I/O manager calls network redirector
  - Server code on remote machine calls remote object manager and I/O manager and delivers data back

Kernel Synchronization

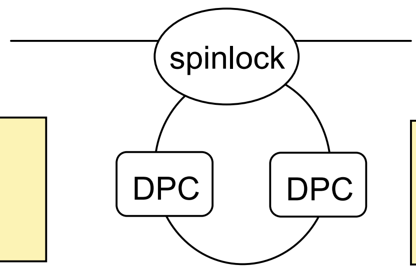
Processor A

.
.
.

do
    acquire_spinlock(DPC)
until (SUCCESS)

begin
    remove DPC from queue
end

release_spinlock(DPC)



Processor B

.
.
.

do
    acquire_spinlock(DPC)
until (SUCCESS)

begin
    remove DPC from queue
end

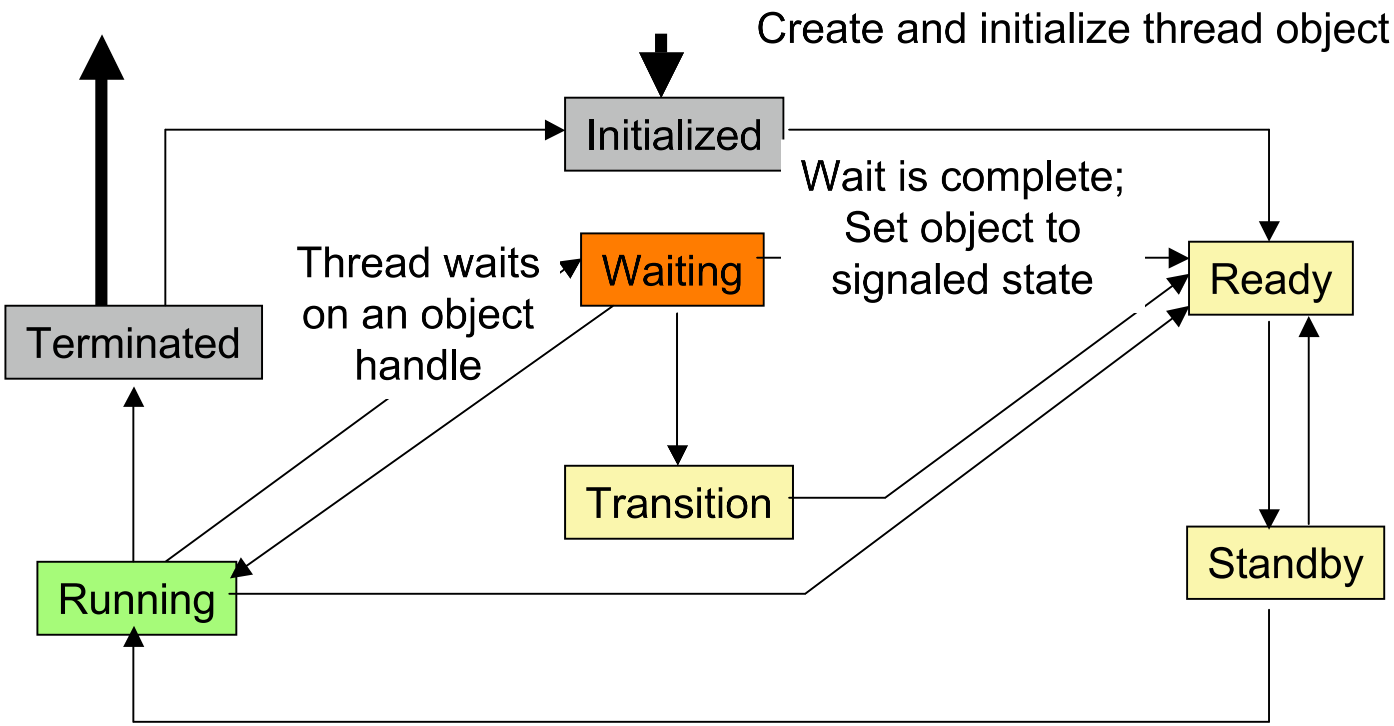
release_spinlock(DPC)

Critical section

A spinlock is a locking primitive associated with a global data structure, such as the DPC queue

Executive Synchronization

- Waiting on Dispatcher Objects – outside the kernel

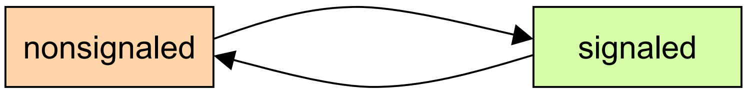
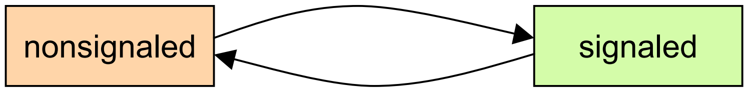
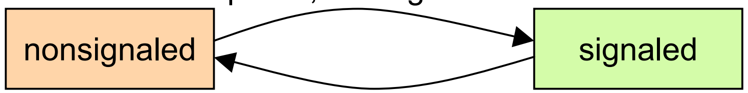


Interaction with thread scheduling

Interactions between Synchronization and Thread Dispatching

1. User mode thread waits on an event object‘s handle
2. Kernel changes thread‘s scheduling state from ready to waiting and adds thread to wait-list
3. Another thread sets the event
4. Kernel wakes up waiting threads; variable priority threads get priority boost
5. Dispatcher re-schedules new thread – it may preempt running thread it it has lower priority and issues software interrupt to initiate context switch
6. If no processor can be preempted, the dispatcher places the ready thread in the dispatcher ready queue to be scheduled later

What signals an object?

Dispatcher object	System events and resulting state change	Effect of signaled state on waiting threads
Mutex (kernel mode)	Owning thread releases mutex  Resumed thread acquires mutex	Kernel resumes one waiting thread
Mutex (exported to user mode)	Owning thread or other thread releases mutex  Resumed thread acquires mutex	Kernel resumes one waiting thread
Semaphore	One thread releases the semaphore, freeing a resource  A thread acquires the semaphore. More resources are not available	Kernel resumes one or more waiting threads

What signals an object? (contd.)

Dispatcher object	System events and resulting state change	Effect of signaled state on waiting threads
Event	nonsignaled → signaled: A thread sets the event signaled → nonsignaled: Kernel resumes one or more threads	Kernel resumes one or more waiting threads
Event pair	nonsignaled → signaled: Dedicated thread sets one event in the event pair signaled → nonsignaled: Kernel resumes the other dedicated thread	Kernel resumes waiting dedicated thread
Timer	nonsignaled → signaled: Timer expires signaled → nonsignaled: A thread (re) initializes the timer	Kernel resumes all waiting threads
Thread	nonsignaled → signaled: Thread terminates signaled → nonsignaled: A thread reinitializes the thread object	Kernel resumes all waiting threads

Local Procedure Calls (LPCs)

- IPC – high-speed message passing
- Not available through Win32 API – W2K OS internal

Application scenarios:

  - RPCs on the same machine are implemented as LPCs
  - Some Win32 APIs result in sending messages to Win32 subsyst. proc.
  - WinLogon uses LPC to communicate with local security authentication server process (LSASS)
  - Security reference monitor uses LPC to communicate with LSASS

- LPC communication:
  - Short messages < 256 bytes are copied from sender to receiver
  - Larger messages are exchanged via shared memory segment
  - Server (kernel) may write directly in client‘s address space

Port Objects

- LPC exports port objects to maintain state of communication:
  - **Server connection port**: named port, server connection request point
  - **Server communication port**: unnamed port, one per active client, used for communication
  - **Client communication port**: unnamed port a particular client thread uses to communicate with a particular server
  - **Unnamed communication port**: unnamed port created for use by two threads in the same process
- Typical scenario:
  - Server creates named connection port
  - Client makes connection request
  - Two unnamed ports are created, client gets handle to server port, server gets handle to client port

Use of LPC ports