Randy Fort
David Shao

CS 249

Kernel Support for Distributed Systems

November 22th, 2005

Randy Fort

The Torvalds/Tanenbaum Debate

November 22th, 2005

The Torvalds/Tanenbaum Debate

- What is the Torvalds/Tanenbaum Debate, and what does it have to do with Distributed Systems?

  - A series of postings on comp.os.minix in which Andrew Tanenbaum started a thread saying “Linux is Obsolete”
  - Posted from 29 January to 10 February of 1992
  - Started a long discussion of Micro vs. Mono kernels
  - Difficult to piece together the mosaic of the whole story
  - Started a long discussion of Micro vs. Mono kernels
  - Was probably the first serious public criticism of microkernels

Typical Mach quotes of the era

- "Optimizing this one [IPC] path *can result* in significant performance gains." [7]
- "…all pagers can be implemented externally and called by the kernel for the user." [7]
- "An unusual feature of Mach, and a key to the systems efficiency is the blending of memory and IPC features" [7]

- Further experimental research on Mach appears ended [3]

To put this discussion into perspective:

When it occurred in 1992:

- DOS and Windows 3.1 were the common Operating Systems.
- O/S 2 was a viable contender for the desktop market.
- Windows as a standalone product was 3+ years away.
- The 386 was the dominating chip.
- And the 486 had not come out on the market.
- Microsoft was still a small company selling DOS and Word for DOS.
  Lotus 123 ruled the spreadsheet space.
- And WordPerfect ruled the word processing market.
- Netscape, Yahoo, Excite, Google, EBay--simply did not exist [2].

Linus was under pressure because he abandoned the idea of microkernels in academia [2].

How Did the Debate Start?

- LINUX is Obsolete - Andrew Tanenbaum
  - Microkernels
  - Portability
  - Minix/Amoeba
- CON:
  - “Your job is being a professor and researcher: That’s one hell of a good excuse for some of the brain-damages of minix. I can only hope (and assume) that Amoeba doesn’t suck like minix does.”

A Brief Mach Kernel History

- From 1985 to 1994, Carnegie-Mellon University (CMU) developed the to support distributed and parallel computation
- Main design goal was to dramatically reduce the size and complexity of the kernel
- The rest of the OS would run as system services in user level processes
- “For some time it appeared that every operating system in the world would be based on Mach by the late 1990s.”[3]
- Mach was was an “academic darling”, and was everything short of a cure for cancer and world hunger.

What is a Monolithic Kernel?

- A monolithic kernel is a single executable handling all kernel functions.
  - Memory is divided into kernel space and user space.
  - Scheduling
  - Process management
  - Signaling
  - Device I/O
  - Paging
  - Swapping
- Because many of these functions have low level code, it may *appear* to be more architecture specific.

What are the Pros/Cons of Monolithic Kernels

- PRO:
  - Single executable works fine if you have the memory
  - Easy implementation of threading for file I/O
  - Very efficient
  - Easier to implement ???
- CON:
  - Memory footprint increases in direct proportion to code size
  - More complicated monolithic structure requires considerably more time and effort to understand
  - Harder to maintain ???

"Most users could probably care less if the internals of the operating system they use is obsolete. They are rightly more interested in its performance and capabilities at the user level. I would generally agree that microkernels are probably the wave of the future. However, it is in my opinion easier to implement a monolithic kernel. It is also easier for it to turn into a mess in a hurry as it is modified. " [2] – Ken Thompson

What is a Mach/Microkernel Kernel?

- Most of the OS runs outside the kernel.
- These processes communicate by message passing.
- The Kernel's job is simple: handle message passing and low level process management.
- Processed outside the kernel include:
  - File system
  - Memory management
  - I/O drivers
- Since the kernel is very small, and all other processes run outside of it, it may *appear* more portable.

What are the Pros/Cons

- PRO:
  - Simpler to understand
  - Good distributed structure
  - Other “servers” are easily replaced
- CON:
  - 20-25% slower than monolithic
  - Complicated message passing infrastructure
  - System services creep back into kernel
  - More complex exception handling in the kernel

What are some more Cons

- CON:
  - Separation of processes could not be realized:
    - "Development of Mach showed that performance problems forces services originally implemented on top of a microkernel back into the kernel, increasing size…" [6]
  - The size and speed benefits never materialized.
    - In fact, they were "larger and slower than monolithic kernels partially because of the complications of modern virtual memory system (copy-on-write facility)" [6]

What are still more Cons

- ***HUGE*** Overhead due to IPC mechanisms

  –"On a 486 (50 MHz), a "null" system call would have a round trip of about 40 µS. On Mach 3, the call alone was 114 µS, with the total call taking 500 µS" [8]

  –A study by Chen and Bershad determined that performance was 66% worse that a monolithic kernel [3,8]

What are even more Cons

In a microkernel, the kernel is supposed to be isolated from the server processes, providing and elegant separation and maintainability advantages. This means the kernel, which is in theory a message and hardware handler, has not idea what the OS consists of.

What important bit of information would you really really really like to know about those processes???

MEMORY PAGING !!!

With no intimate knowledge of kernel interaction (which is easy on a monolithic kernel), you must adopt a one size fits all memory paging solution.

How is it Different in Practice?

Monolithic:

Kernel system calls (traps) when a system call is invoked, the code “traps” into the kernel and the code is executed, the flow of execution returns to the calling function.

Microkernel:

System calls post messages, and a context switch occurs passing control back to the microkernel via IPC messages. Shared memory used for this role.

Mach IPC in Practice

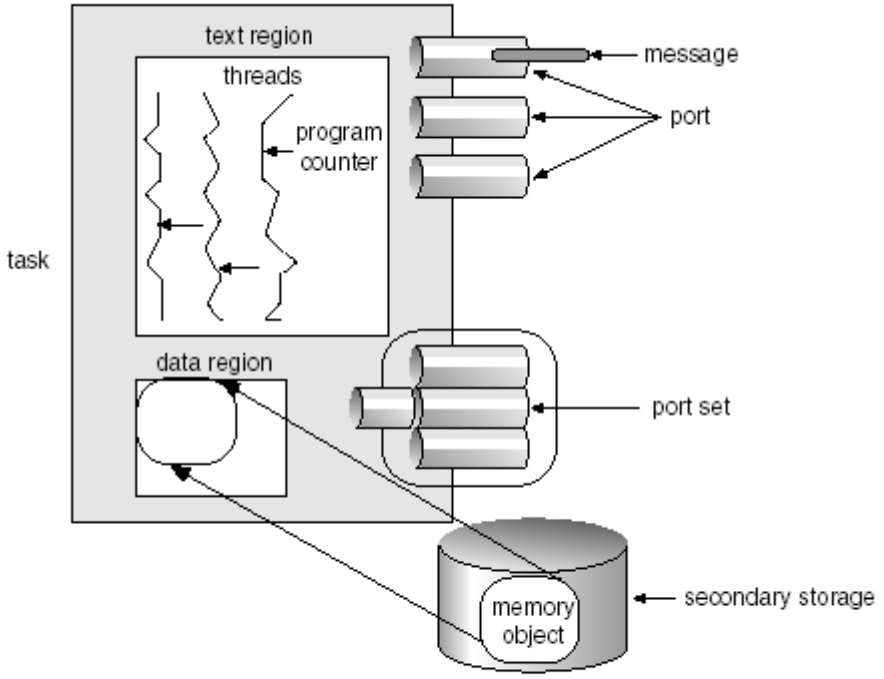


Figure B.2 Mach's basic abstractions.

All IPC happens through messaging. [7]

Mach IPC in Practice

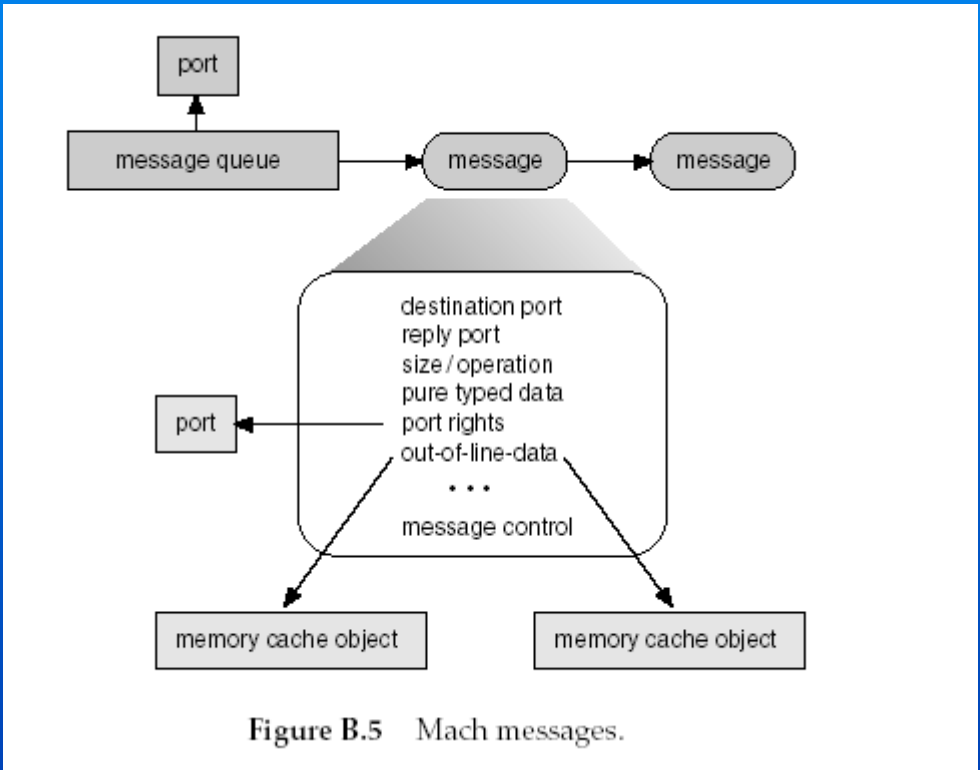


Figure B.5 Mach messages.

All IPC happens through messaging. [7]

What is a Microkernel (supposed to be) in a Nutshell?

- A small coordinating microkernel
- All other processes operate outside the kernel
- Other processes are called “servers”
- All processes communicate via message passing
- Shared memory IPC used extensively

Observations on Microkernels

“So at the time I started work on Linux in 1991, people assumed portability would come from a microkernel approach. You see, this was sort of the research darling at the time for computer scientists.” Linus Torvalds[2]

“microkernel – based operating systems seem to be widely regarded by the research community and the industry as a panacea, whereas, if monolithic operating systems are not quite treated with contempt, they are certainly regarded as old fashioned.” Sape Mullender [1]

Linus' Observation on Portability

"My point is that writing a new OS that is closely tied to any particular piece of hardware,… is basically wrong" – Andrew Tanenbaum [2]

"But ***my*** point is that the operating system ***isn't*** tied to any processor line: UNIX runs on most real processors in existence. Yes, the ***implementation*** is hardware-specific, but there's a HUGE difference. ."

"In fact, the whole linux kernel is much smaller than the 386-dependent things in mach:" - Linus Torvalds [2]

Tanenbaum's Folly (1993)

"A multithreaded file system is only a performance hack. When there is only one job active, the normal case on a small PC, it buys you nothing and adds complexity to the code. On machines fast enough to support multiple users, you probably have enough buffer cache to insure a hit cache hit rate, in which case multithreading also buys you nothing. It is only a win when there are multiple processes actually doing real disk I/O. Whether it is worth making the system more complicated for this case is at least debatable." [2] Andrew Tanenbaum

Tanenbaum in 2002

“An important property of threads is that they can provide a convenient means of allowing blocking system calls without blocking the entire process in which the thread is running. This property makes threads particularly attractive to use in distributed systems as it makes it much easier to express communication in the for of maintaining multiple logical connection at the same time.” [5] Andrew Tanenbaum

The Verdict of History

1985: GNU HURD in development	Trivial development since 1994 (0.2)
1991: Linux started development	2005 Widespread global usage
1992: 3 million microkernel Amigas	2005: What is an Amiga?
1990: Early version of the Mach "microkernel" released	The microkernel was larger than the monolithic kernel it was supposed to replace
1990: First Mach kernel with all servers running in user space is released	Overall system performance compared to monolithic kernels was 66% slower [8]
1985: Mach development at CMU	1994: RIP

Successful Microkernel Implementations

AIX

Mac OS X

QNX

BeOS

Bibliography

- [1] Sape Mullenderlender, Distributed Systems, Addison Wesley, 1993
- [2] Chris DiBona, et al , Open Sources: Voices from the Open Source Revolution, O'Reilly and Associates:
- [3] Mach kernel: http://en.wikipedia.org/wiki/Mach_kernel
- [4] Boykin , Joseph, Programming Under Mach
- [5] Tanenbaum, Andrew S, and Maarten van Steen, Distributed Systems, Pearson Education, 2002
- [6] Nikolai Bezroukov et al, Portraits of Open Source Pioneers, http://www.softpanorama.org/
- [7] Silberschatz A and P. Galvin, Operating System Concepts, Addison Wesley, 1994
- [8] J. Bradley Chen, Brian N. Bershad. "The Impact of Operating System Structure on Memory System Performance", The Fourteenth Symposium on Operating System Principles

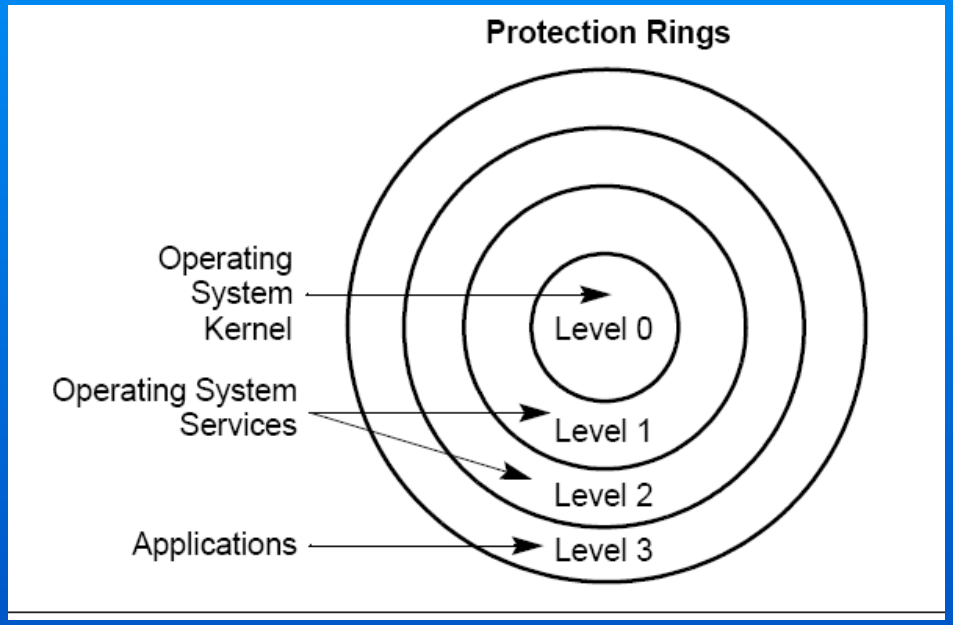
David Shao
IA-32 Architecture and OS Kernel Design

November 22nd, 2005

Overview

- *IA-32, 64-bit extensions, and address spaces*
- *Context switches and the TLB*
- *Threads*
  - *Futex*
  - *Local data*
- *Virtualization*

Protection Levels for IA-32



- *Kernel vs. user: ring 0 vs. ring 3*
- *Analogy: UNIX group permissions*
- *Source: Figure 4-3, p. 4-9,* Intel *Vol. 3*

16-bit Origins

- $2^{16} = 65536$
- *Source: [Intel3, page 3-5]*

Data segments registers FS, GS added in i386

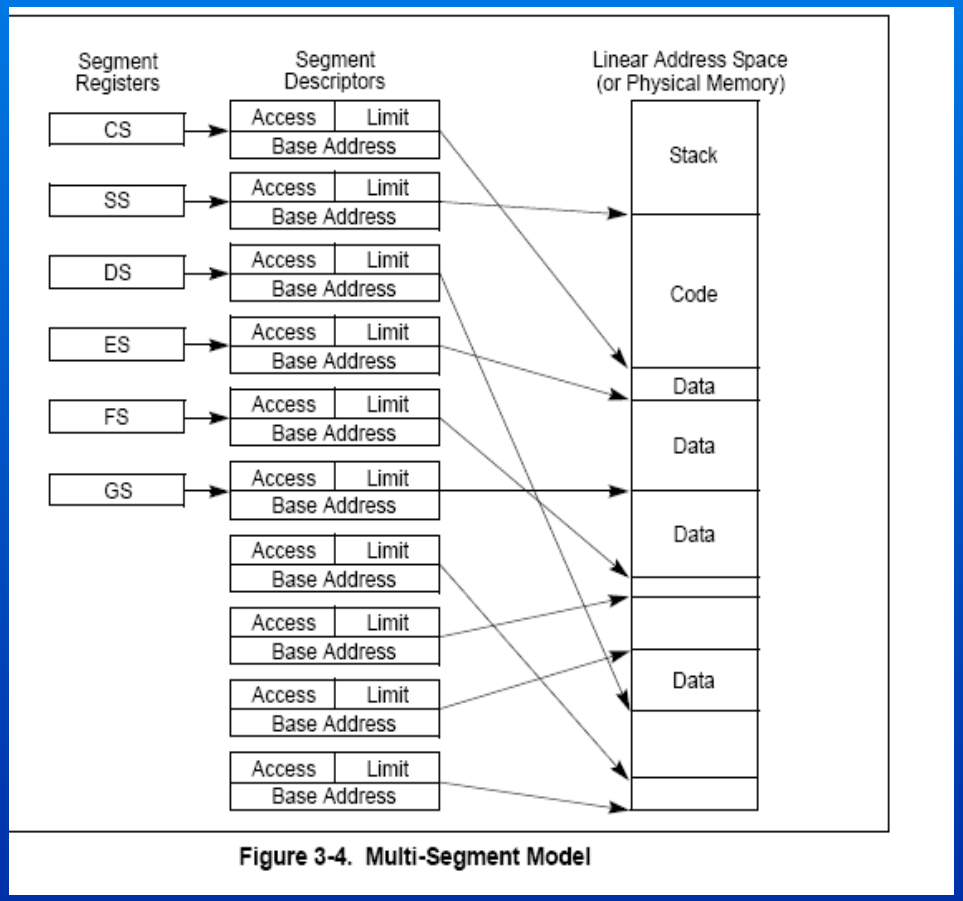


Figure 3-4. Multi-Segment Model

Descriptor Tables

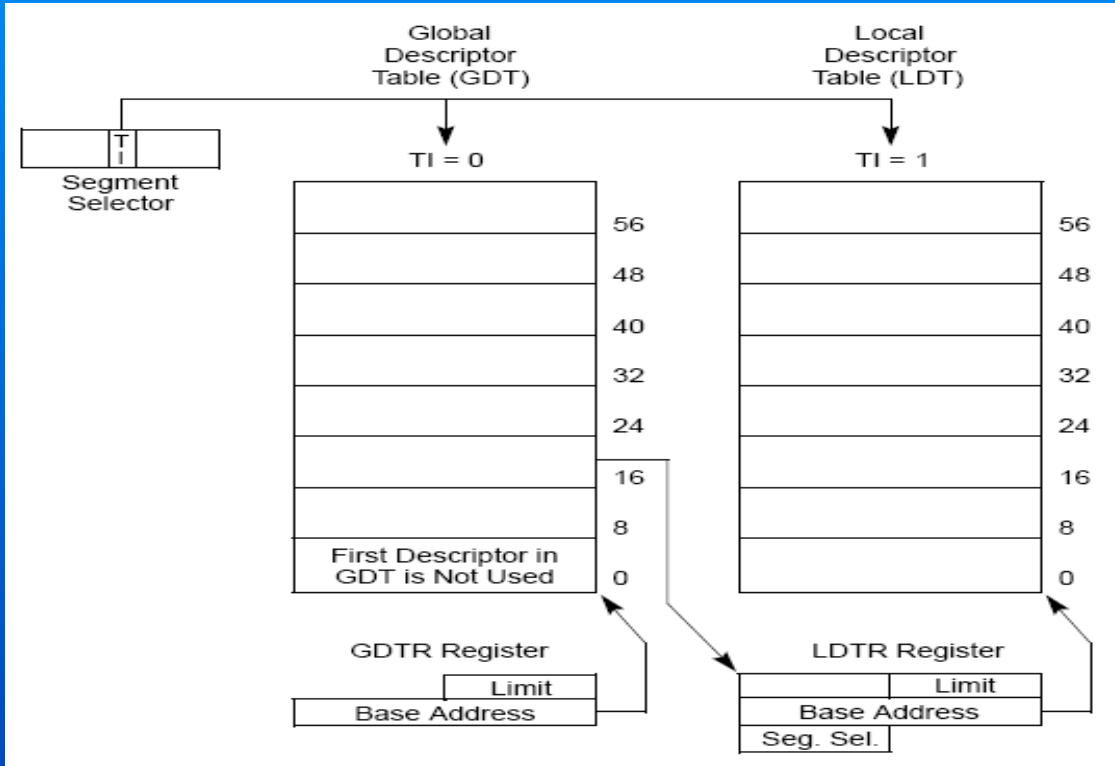


Figure 3-10. Global and Local Descriptor Tables

- *8192 max descriptors in a table*
- *Source: [Intel3, page 3-18]*

Why a New OS for 32-bits, Not 64

- ***Entire process address space could fit within 2^{32} bytes, but not 2^{16} bytes. Write new 32-bit OS to use virtual memory***
- ***Mullender speculates: if had 64-bit address space, could an OS fit all process address spaces?***
- ***Sad answer—won't even have full 64-bits to use.***

Linear Address Translation

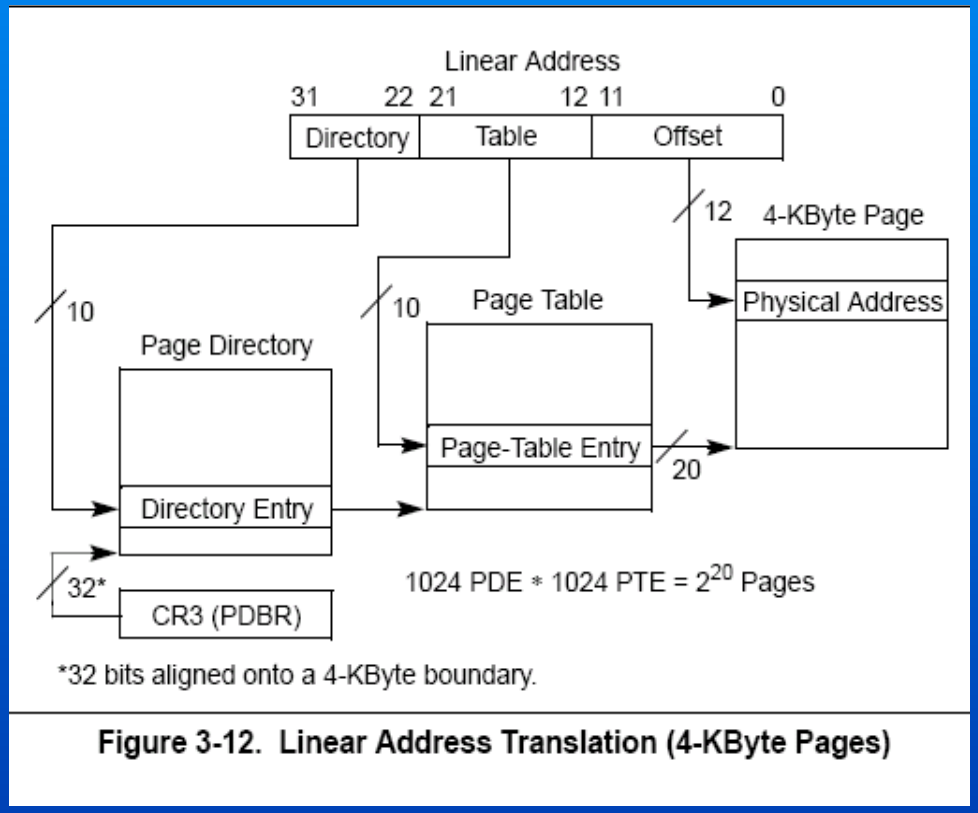


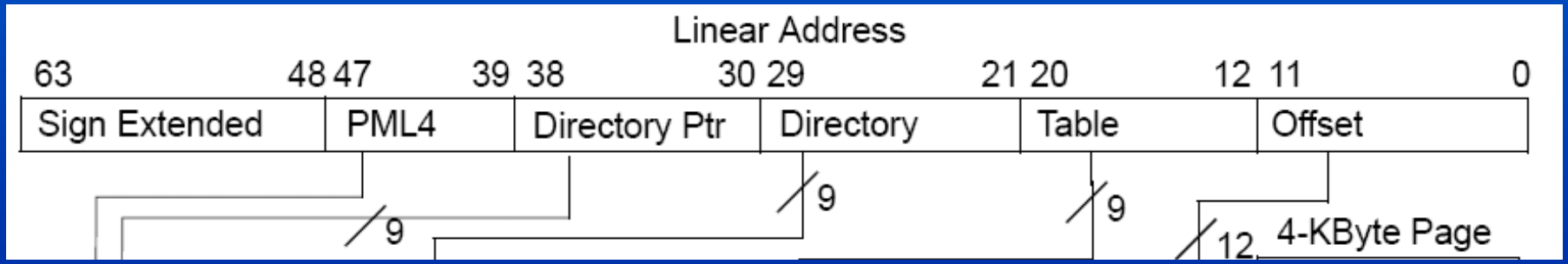
Figure 3-12. Linear Address Translation (4-KByte Pages)

- ***4KB = 2^{12} Bytes page size, and 2^{20} possible pages***
- ***4B = 32 bits entry size***
- ***$2^{12}/4 = 2^{10}$ entries/page***
- ***Each page directory entry can point to a page table that can point to 2^{10} pages***
- ***$2^{10} * 2^{10} = 2^{20}$***
- ***$2^{20} * 2^{12} = 2^{32}$ bytes covered***

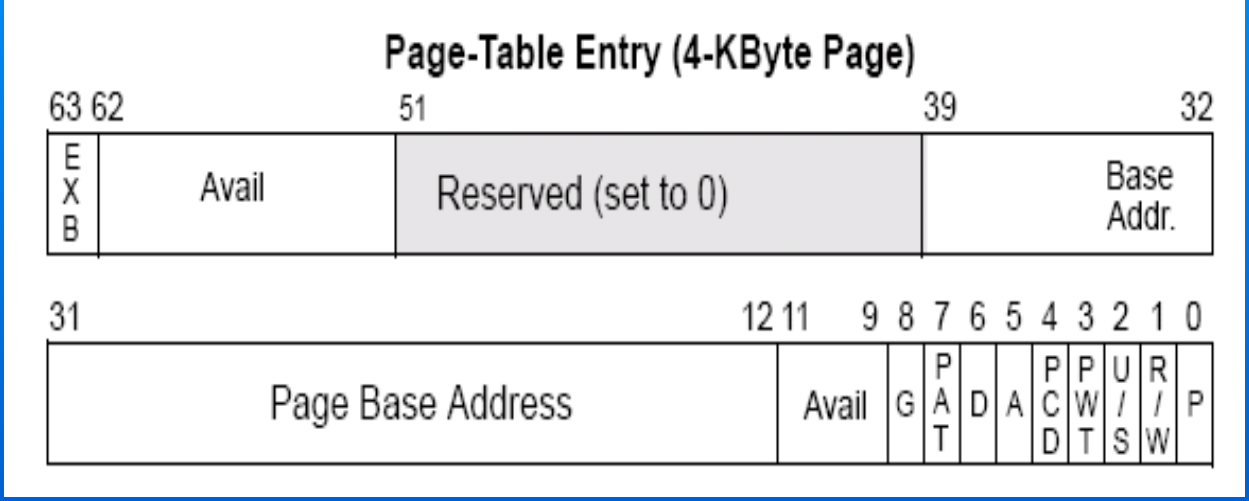
Source: [Intel3, page 3-23]

Why 64-bits Is Not Always 64-bits

- *Assume 8 byte (64-bit) entry size, 4kB page size. Suppose one wants to address 2^{48} bytes physical memory.*
- *4k / 8 = 512 entries / page = 2^9 entries / page max*
- *2^{48} / 2^{12} = 2^{36} pages*
- *$(2^9)^4 = 2^{36}$, so FOUR levels of indirection!*
- *Get that but no more in AMD64/EM64T, so far*
- *Source: [Intel3, Figure 3-24, page 3-40]*



IA-32e Page-Table Entry



39 – 12 + 1 = 28
28 + 12 = 40-bit maximum physical address space?

Flag	Purpose
D (Dirty)	Has page been written to?
U/S (User/Supervisor)	Privilege level to access page?
R/W (Read/Write)	Writes allowed to page?
EXB	Execute-disable bit ("NX")

- ***Source: [Intel3, Figure 3-26, p. 3-42]***

Translation Lookaside Buffer

- *Translation Lookaside Buffer (TLB)—hardware cache of how linear (virtual) addresses mapped to physical addresses*
- *Not easily controlled by software even in Ring 0*
- *Instruction TLB (4KB page)—up to 128 entries; Data TLB (4KB page)—up to 64 entries*
- *Flushed if page-directory base register CR3 changed*
- *(Global) pages can have flag set (G) so that TLB entries not invalidated*
- *Source: [Intel3, page 2-17 and Table 10-1, page 10-2]*

Context Switch/Microkernels on IA-32

- *Context switch requires a change of address spaces.*
- *Changing address spaces forces on IA-32 invalidation of non-global TLB entries.*
- *But tables for address spaces are themselves in memory. Non-cached memory access relatively expensive.*
- *Example microkernel problem: Implementing a filesystem in userspace doubles context switches for file access.*

Mutex Using Futexes

- *Mutex: mutual exclusion, at most one thread holds lock*
- *Drepper's example uses shared object (in C++) with lock() and unlock() methods*
- *Acquire lock (maybe waiting), perform work, release lock, wake up one waiter if possible.*
- *Three states: 0 for "unlocked", 1 for "locked but no waiters", 2 for "locked and maybe waiters".*
- *Shared object has state as private instance variable*
- *Source: [Drepper]*

(Machine) Atomic Instructions

- ***Return previous value of state AND change state without another process/thread interrupting***
- ***Assembly language instructions that can work in userland***
- ***Source: [Drepper, Appendix A, page 11; Intel2A, page 3-144; Intel2B, page 4-363]***

Instruction	Description
XCHG &state, new	Exchange
CMPXCHG &state, expected, new	Compare and exchange
ATOMIC_DEC &state	Atomic decrement
ATOMIC_INC &state	Atomic increment

Added Kernel Functionality

- *futex_wait(&state, expected)*
  - *IF value of state equals expected, add thread to &state's wait queue and return after thread woken up or error*
  - *ELSE immediately return, unexpected state*
- *futex_wake(&state, number)*
  - *Wake number waiting on &state queue.*
- *Works for inter-process because kernel keeps queue based on physical memory address.*
- *Above not actual system calls for Linux*
- *Source [Drepper, Appendix A, page 11]*

Acquiring Lock

1. *Use CMPXCHG to test whether "unlocked" and try to change to "locked but no waiters".*
2. *IF successful now have lock and no need to invoke kernel. DONE.*
3. *ELSE keep using XCHG to set state to 2 "locked and maybe waiters", then test if previous state was "unlocked", state 0*
   - *IF previous state was state 0, now have lock. DONE.*
   - *ELSE previous state was 1 or 2 (lock held). Use futex_wait with expected state 2. After return from futex_wait, restart step 3 above.*

- *Source: [Drepper, Section 6, pages 7-8]*

Releasing Lock

- *Use atomic_dec on state.*
- *IF previous state was "locked but no waiters", no need to ask kernel to wake up waiter. Lock had been released by previous atomic_dec. DONE.*
- *ELSE previous state was "locked and maybe waiters".*
  - *Set state to "unlocked", releasing lock.*
  - *Use futex_wake to ask kernel to awaken exactly one waiter, if possible. (Only one thread can have the lock at any given time. If all threads woke, cache line passing between CPUs expensive.) DONE.*
- *Source: [Drepper, page 4 and page 8]*

Atomic Implementation: Few CPUs

- ***Synchronize on access to one bus shared by all processors***
- ***Simplest cache coherency mechanism: each processor simply listens on bus for cache messages***

Above called **snoopy cache**

Source: http://www.unitedmedia.com/comics/peanuts/

Thread-Local Data in Linux

- *Opportunity: Thread-local data located at similar offsets*
- *Source: [DrepperT, page 6]*

thread register (pointer)

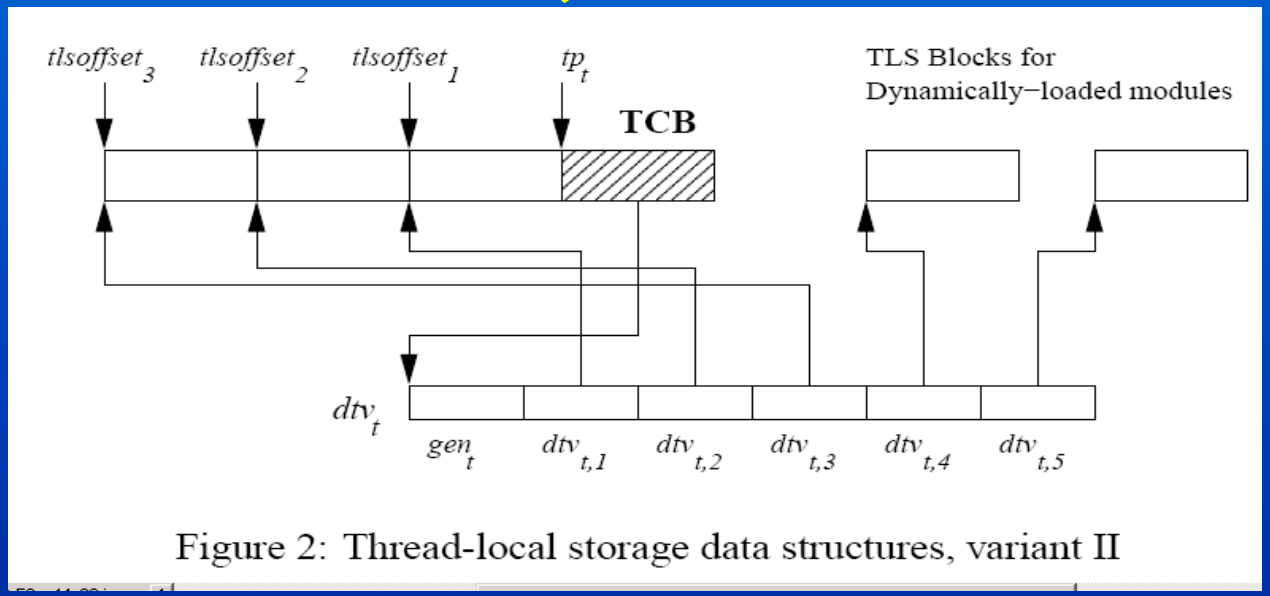


Figure 2: Thread-local storage data structures, variant II

Evolution Between Linux 2.4 and 2.6

2.6: GDT, scheduler updates?

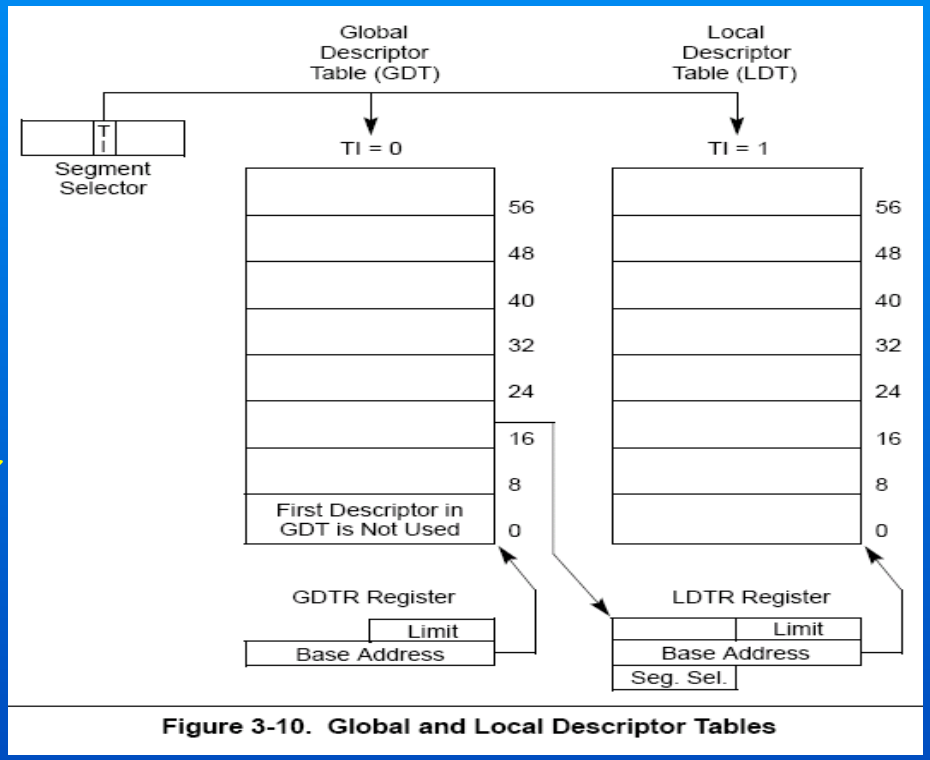
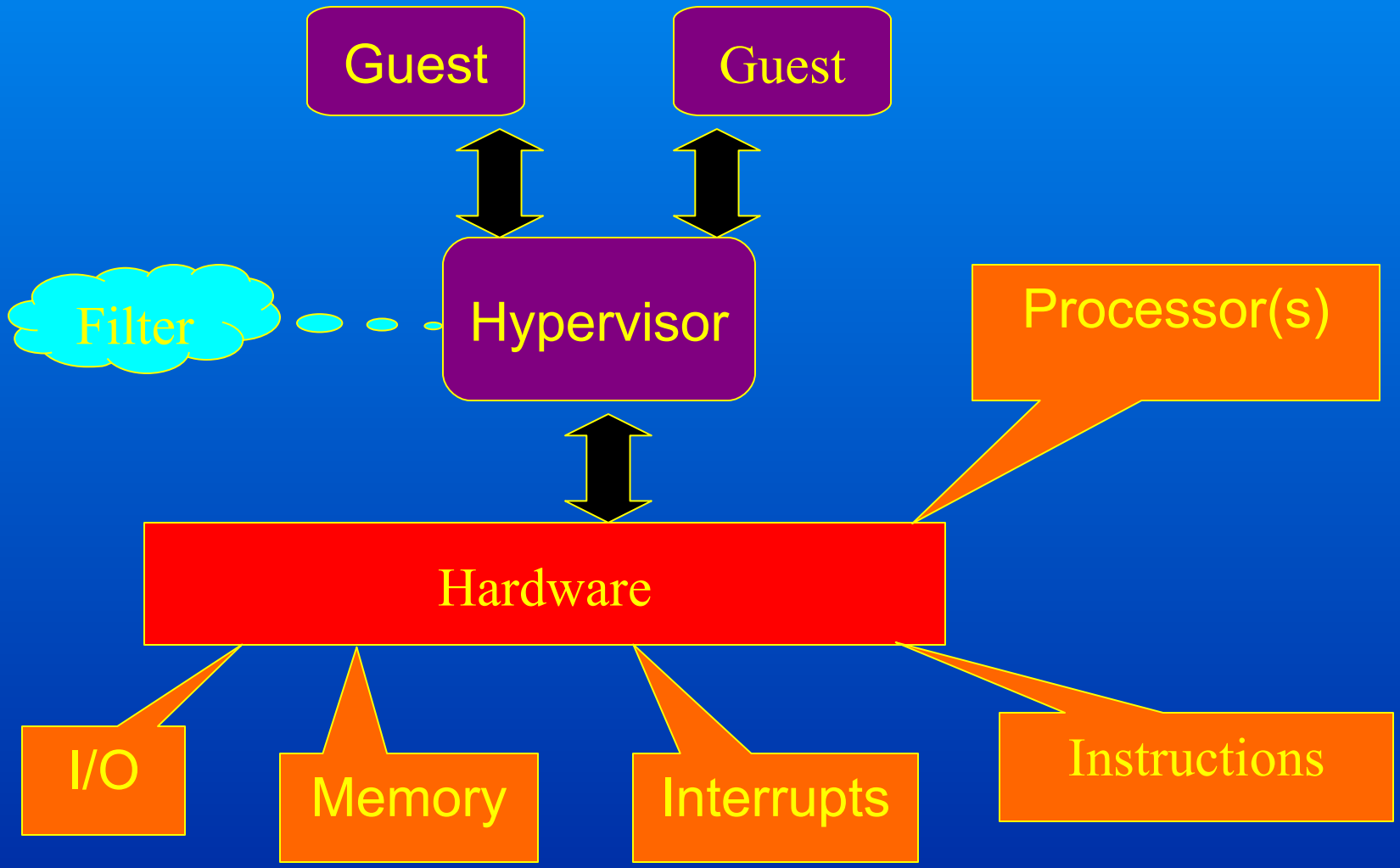


Figure 3-10. Global and Local Descriptor Tables

2.4: LDT, limit of 8192 threads

- *Kept 1-to-1 threading model, rejected M-to-N (many-to-many)*
- *Source [Intel3, page 3-18; DrepperN, page 10]*

Virtualization



Intel, AMD Both Back Virtualization

- *Moore's Law not really about speed but about shrinking circuit size*
- *How to use extra chip room: both Intel and AMD introducing multi-core processors*
- *Both adding instructions to support full virtualization. Ring 0 software works unmodified.*
- *Intel's additions ("Vanderpool")—renamed to Intel Virtualization Technology?*
- *AMD's additions: "Pacifica"*
- *Totally different instruction sets, in contrast to AMD64 vs. EM64T*
- *Reference: [AMDV and IntelV]*

Xen

- *Developed at the University of Cambridge Computer Laboratory, Systems Research Group*
- *http://www.cl.cam.ac.uk/Research/SRG/netos/xen/*
- *Original goal: "wide area distributed computing"*
- *Has had funding from Microsoft, HP, Intel, etc.*
- *Xen source code GPLed. No restrictions on guest OSes using Xen API.*
- *Hypervisor runs on Linux.*
- *Xen + LAMP (Linux, Apache, MySQL, PHP) mindshare?*

(Xen)Virtualization Terminology

- ***A hypervisor, or virtual machine monitor (VMM), regulates how each guest OS uses hardware.***
- ***Rather than completely emulating hardware, paravirtualization may require guest OS to be partially rewritten***
- ***Tradeoff between fully emulating x86 and partially rewriting O. Ring 0 software might need to be rewritten to be ring 1.***
- ***Xen's API consists of hypercalls***

AMD Virtualization

- *Feature difference: AMD supports "tagged TLB"*
- *Address Space Identifier (ASID) added to TLB entries*
- *Process switch without TLB flush?*
- *Cannot given ASID flush only its TLB entries?*
- *For one page and one ASID can flush*
- *Reference [AMDV, pages 28-29]*

References

- *[AMDV]* Secure Virtual Machine Architecture Reference Manual, *Revision 3.01, May, 2005. Downloaded October 2005 from http://www.amd.com/us-en/Processors/TechnicalResources/0,,30_182_739_7044,00.html*
- *[Drepper] Drepper, Ulrich. Futexes Are Tricky. Version: December 13, 2004. Downloaded October 2005 from http://people.redhat.com/~drepper/futex.pdf*
- *[DrepperN] Drepper, Ulrich and Ingo Molnar. The Native POSIX Thread Library for Linux. Version: February 21, 2005. Downloaded October 2005 from http://people.redhat.com/~drepper/nptl-design.pdf*
- *[DrepperT] Drepper, Ulrich. ELF Handling for Thread-Local Storage. Version: February 8, 2003. Downloaded October 2005 from http://people.redhat.com/~drepper/tls.pdf*

References (cont.)

- *[Intel1]* IA-32 Intel® Architecture Software Developer's Manual, Volume 1: Basic Architecture, *version September 2005, downloaded October 2005 from http://developer.intel.com/design/pentium4/manuals/index_new.htm*
- *[Intel2A]* IA-32 Intel® Architecture Software Developer's Manual, Volume 2A: Instruction Set Reference, A-M, *version September 2005, downloaded October 2005 from http://developer.intel.com/design/pentium4/manuals/index_new.htm*
- *[Intel2B]* IA-32 Intel® Architecture Software Developer's Manual, Volume 2B: Instruction Set Reference, N-Z, *version September 2005, downloaded October 2005 from http://developer.intel.com/design/pentium4/manuals/index_new.htm*
- *[Intel3]* IA-32 Intel® Architecture Software Developer's Manual, Volume 3: System Programming Guide, *version September 2005, downloaded October 2005 from http://developer.intel.com/design/pentium4/manuals/index_new.htm*
- *[IntelV]* Intel® Virtualization Technology Specification for the IA-32 Intel® Architecture, *version April 2005, downloaded October 2005 from http://www.intel.com/technology/computing/vptech/*

References (cont.)

- *Pratt, Ian, et al. Xen Status Report. Ottawa Linux Symposium 2005 presentation. Downloaded October 2005 from http://www.cl.cam.ac.uk/netos/papers/2005-xen-ols.ppt*