Tanenbaum-Torvalds microkernel vs monolithic kernel Debate came in two waves!

- First wave was in 1992. Tanenbaum claims “LINUX is obsolete”.
- Second wave was in 2006. Linus claims “The whole 'microkernels are simpler' argument is just bull,...”
- Linus lost 1st argument and won the 2nd.

http://www.Free.BlackPatchPanel.com/pme/linux/ukernel.pdf

Tanenbaum attacks with "LINUX is obsolete"

- 1992 "While I could go into a long story here about the relative merits of the two designs, suffice it to say that among the people who actually design operating systems, the debate is essentially over. Microkernels have won. The only real argument for monolithic systems was performance, and there is now enough evidence showing that microkernel systems can be just as fast as monolithic systems (e.g., Rick Rashid has published papers comparing Mach 3.0 to monolithic systems) that it is now all over but the shoutin'."

Linus responds (1992)

- with a lot of ad hominem heat and flame that he later had to apologize for. But on the main question “Are microkernels better?”
- “True, linux is monolithic, and I agree that microkernels are nicer. With a less argumentative subject, I'd probably have agreed with most of what you said. From a theoretical (and aesthetical) standpoint Linux looses. If the GNU kernel had been ready last spring, I'd not have bothered to even start my project: the fact is that it wasn't and still isn't.”

microkernels (1992)

- I question whether Linus really believed this admission. The fact is that he continued with LINUX and did not convert it to a microkernel design. His unerring software engineering instinct must have told him that it would have been a mistake.
- But he was not prepared to challenge a respected and polished professor on his own home ground of academic debate. Hence this confession. All the academic pundits were saying “microkernels are better”.

Are Microkernels better?

- As a result of persistent performance and design problems, the conventional wisdom that “microkernels are always better” is not now so common.
- “Regardless of the advantages of the Mach approach, these sorts of real-world performance hits were simply not acceptable. As other teams found the same sorts of results, the early Mach enthusiasm quickly disappeared. After a short time many in the development community seemed to conclude that the entire concept of using IPC as the basis of an operating system was inherently flawed.”
- `http://en.wikipedia.org/wiki/Mach_kernel`

Monolithic kernels (1992)

- I also question the extent to which Tanenbaum may have been blowing smoke. He had to be aware of the performance problems microkernels were experiencing.
- If he could persuade all developers that “LINUX is obsolete” it could turn into a self fulfilling prophecy and save him from embarrassment.

2006, the second wave of the debate, Linus blasts microkernels.

- On May 9, 2006 in a post on Real World Technologies discussion forum Linus begins the second round of the debate. This post was not specifically addressed to Tanenbaum. It made a number of claims:
- Microkernels are not simpler.
- Microkernels have performance problems.
- SW development is slower for mircokernels.

Linus claims continued (2006)

- Microkernels force designers to use distributed algorithms which are more difficult to write and maintain.
- Microkernels are not more secure or stable.
- A failed service often takes the whole system down contrary to what MK advocates claims.
- Microkernels make it more difficult to handle coherency issues, which are common in OS design.
- hybrid kernels are a marketing ploy.

Linus substantiates the claims

- Linus makes these claims in a no holds barred non equivocal fashion.
- These claims are interspersed with a detailed analysis of why the problems flow from a prime microkernel feature, namely separate address spaces; Or as Linus puts it “separate access spaces”.
- In further posts Linus amplifies and expands on his position.

“Many small tools”

- Linus accuses the microkernel advocates of taking the Unix “many small tools” idea and attempting to use it in a problem space where it totally breaks down. They are senselessly trying to fit a round peg into a square hole and it refuses to fit.

The concept sounds good.

- “The final (and I think deciding) argument is that the real argument for microkernels has nothing to do with speed, ease of implementation, security, or anything else.
- The real reason people do microkernels (and probably will continue to do them, and make up new reasons for why they are better) is that the concept sounds so good. It sounded good to me too!”

I did not know how hard it would be.

- “The whole 'make small independent modules' thing just sounds like manna from heaven when you're faced with creating an OS, and you realize how daunting a task that is. At that point, you can either sit back and enjoy the ride (which I did - partly because I didn't initially really realize how daunting it would be), or you can seek mental solace in an idea that makes it sound easier than it is.”

Wish for a simpler world

- “So that 'microkernels are wonderful' mantra really comes from that desperate wish that the world should be simpler than it really is. It's why microkernels have obviously been very popular in academia, where often (you) basically cannot afford to put a commercial-quality big development team on the issue, so you absolutely require that the problem is simpler.”

Escape from reality “OS design is hard.”

- “So reality has nothing to do with microkernels. Exactly the reverse. The whole point of microkernels is to try to escape the reality that OS design and implementation is hard, and a lot of work.
- It's an appealing notion.”

Linus mentions Tanenbaum.

- Only at one point in the thread does Linus actually mention Tanenbaum:
- “Shapiro (and to some degree Tanenbaum) also makes the mistake of equating different address spaces with the notion of modularity. They have nothing to do with each other. You can be modular without using hardware to enforce it, and you can generate a horribly messy system where two processes are intimately aware of how each other works even if they are separated by MMU boundaries.”

Linus plugs Tanenbaum's book.

- “That said, I still think Tanenbaum's book on OS design is one of the best ones around. So I'll happily disagree with him, and I can still give the man credit for being a big reason for getting involved and interested in UNIX in the first place!”

My oversimplification

- The following slides are my over simplification of some of the things the critics of microkernels are saying. (2006)
- I hope I have not distorted them too much.

When all you've got is a hammer, everything looks like a nail.

- Since the beginning of Operating Systems, when dinosaurs roamed the earth, OS designers have used process address separation to prevent LUSERS from stomping on each other.

interprocess address separation is the hammer that ...

- microkernel advocates want to use interprocess address separation to reduce OS complexity and errors.
- Interprocess (thread) communication is limited to message passing. This is essentially “pass by value”.
- This is a form of “bondage and discipline programming.”

Bondage and Discipline programming.

- seeks to prevent errors by creating a “system” where errors of a certain type are impossible.
- “system” creators are the “smart” people on the central committee. They don't trust ordinary developers to “do it right”, hence this program to control them.
- In the case of µkernels, the disk driver (hopefully) can not interfere with the video driver, because they are in separate processes with separate address spaces.
- The Java approach to memory leaks is another example of “bondage and discipline programming”.
- Pascal is the canonical Bondage and Discipline language.
- Bondage and Discipline programming seems to be favored in academia

Flaws of Bondage and Discipline programming

- (Flaw 1) bondage and discipline programming causes overhead and reduces your performance.
- (Flaw 2) bondage and discipline programming won't let you choose the best method to achieve your goal, so your design becomes more difficult.
- (Flaw 3) The “smart” people on the central committee, the creators of the B&D system, are not as smart as they think they are.

Microkernels have both B&D flaws!

- The overhead associated with context switching and message checking reduces system performance. (The MK folks have done a **lot** of work on this without completely solving it.) (Flaw 1)
- “The fundamental result of access space separation is that you can't share data structures. That means that you can't share locking, it means that you must copy any shared data, and that in turn means that you have a much harder time handling coherency. All your algorithms basically end up being distributed algorithms.” (Flaw 2)

Dissident voices.

- The TUNES people also have a harsh criticism of microkernels. They accuse the microkernel design of “abstraction inversion”.
- http://tunes.org/wiki/Microkernel
- According to them, criticism of microkernels is almost unknown in the academic world, where it might be a career limiting move (CLM).
- Their criticisms are very similar to Linus' except expressed in a more polished academic language.

Linus proposes an academic research project!

- In a later post, Linus suggests that a special computer language be developed for OS design. Compilers for this language would have the ability to check statically at design, compile, link time, the very things that µkernels are trying to check at run time with the MMU!
- If this could be done, it would obviously be more efficient than doing it at run time with the MMU.
- This idea is perfect for academic research!

Tanenbaum replies

- Not on any discussion forum, but on his own web page entitled “Tanenbaum-Torvalds Debate: Part II”
- He begins with expressions of collegiality; Tanenbaum is not Linus' enemy

Minix 3

- Tanenbaum has released Minix 3 under a BSD like license, an improvement over the original Minix which was under a proprietary license. Perhaps Tanenbaum has learned the value of free software.
- Minix 3 uses microkernel. OS book is updated to reflect changes in Minix 3.
- Much of the reply is marketing hype for Minix 3.

Minix 3 target

- Minix 3 is initially targeted at:
  - Applications where very high reliability is required
  - Single-chip, small-RAM, low-power, $100 laptops for Third-World children
  - Embedded systems (e.g., cameras, DVD recorders, cell phones)
  - Applications where the GPL is too restrictive (MINIX 3 uses a BSD-type license)
  - Education (e.g., operating systems courses at universities)
- So Minix 3 does not compete with Linux.

Technical reply

- Tanenbaum does make some reply to some of Linus' arguments, some of which is unclear, some of which is clearly wrong, some of which is not on point.

synchronization?

- “Linus also made the point that shared data structures are a good idea. Here we disagree. If you ever took a course on operating systems, you no doubt remember how much time in the course and space in the textbook was devoted to mutual exclusion and synchronization of cooperating processes. When two or more processes can access the same data structures, you have to be very, very careful not to hang yourself. It is exceedingly hard to get this right, even with semaphores, monitors, mutexes, and all that good stuff.”

synchronization methods

- Tanenbaum seems to be suggesting that Microkernels do not need synchronization methods, i.e. semaphores, monitors, mutexes, and all that good stuff.
- But microkernels do have message queues.
- It is the use of message queues that prevents race conditions in microkernels.

synchronization methods

- But message queues are accessed from all over the system and they must be kept in a consistent state. How is this accomplished?
- Answer: synchronization methods, that is, semaphores, monitors, mutexes, and all that good stuff.
- So microkernels do not avoid synchronization methods.
- They simply use a one size fits all approach to race conditions called message queues.

Linus not object oriented?

- “My view is that you want to avoid shared data structures as much as possible. Systems should be composed of smallish modules that completely hide their internal data structures from everyone else. They should have well-defined 'thin' interfaces that other modules can call to get work done. That's what object-oriented programming is all about--hiding information--not sharing it. I think that hiding information (a la Dave Parnas) is a good idea. It means you can change the data structures, algorithms, and design of any module at will without affecting system correctness, as long as you keep the interface unchanged. Every course on software engineering teaches this. In effect, Linus is saying the past 20 years of work on object-oriented programming is misguided. I don't buy that.”

Object oriented?

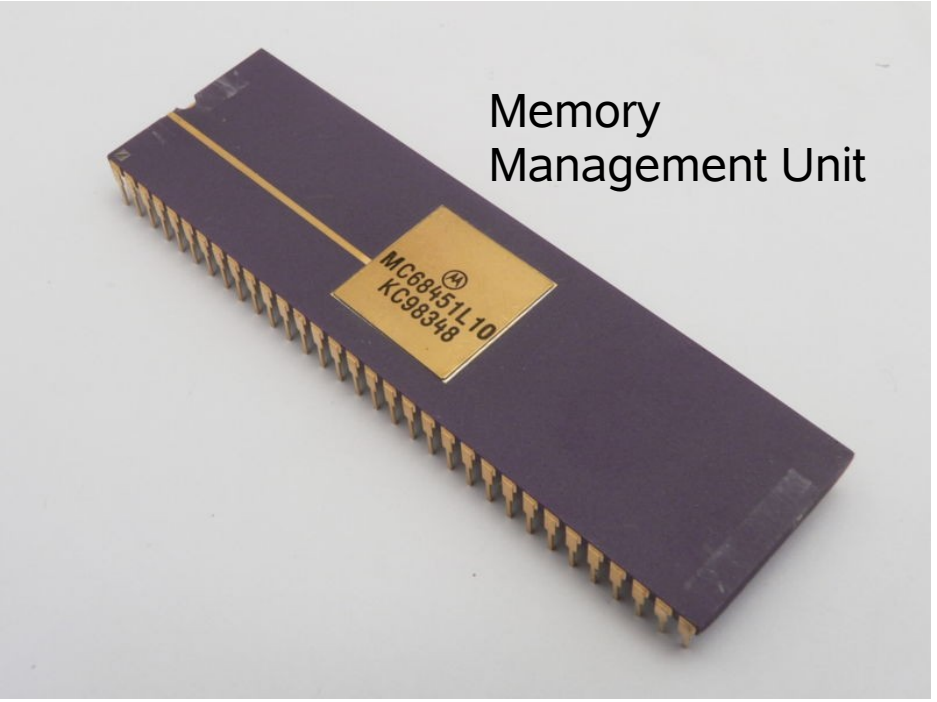
- “That's what object-oriented programming is all about--hiding information--not sharing it”
- Tanenbaum is giving the impression that data hiding prevents race conditions, but it is not true!
- Data hiding (encapsulation) does not mean that different threads of execution do not access that data at potentially the same time! It does not prevent race conditions! That is why Java has a synchronized keyword! What prevents race conditions in the µkernel design is not encapsulation, but rather a particular way of handling of messages! The Object Oriented paradigm does not define how messages are to be implemented.
-
-
- So the issue is not object orientation at all!

Kernel constraint compiler?

- Tanenbaum does not respond to Linus' idea for a OS compiling, constraint checking, computer language.

What is the MicroKernel really?

- What is the microkernel style exactly? It is basicly a way of using the MMU to do some kinds of checking. The extra code that does this checking runs at least 100 times per second on every CPU that runs the OS. To facilitate this checking, developers must reorganize the way their code is organized, breaking the flow of thought and understanding into a lot of small pieces.
- All to do some checking that at least in theory, could have been done at design, compile, link time once.
- This checking only checks for only some of the possible coding errors. Most OSes do not crash because of stray memory references. There are 49 other crash landings.



- Academic researchers should be thinking at a high level.
- They should not be trying to solve low level problems with low level solutions, that disrupt how OS designers can express their designs and implementations.

Big picture

- Microkernels are a form of Bondage and Discipline Programming.
- 14 years ago in 1992, Tanenbaum said: “Linux is obsolete” and “it's all over but the shoutin”.
- 14 years later Minix 3 does not compete with Linux in its own huge problem space. Neither does any other microkernel design.
- It is time for microkernel advocates to come up with something that works on real world problems or stop shouting!

Copyright