Redesigning UNIX for Reliability

Contents

Overview

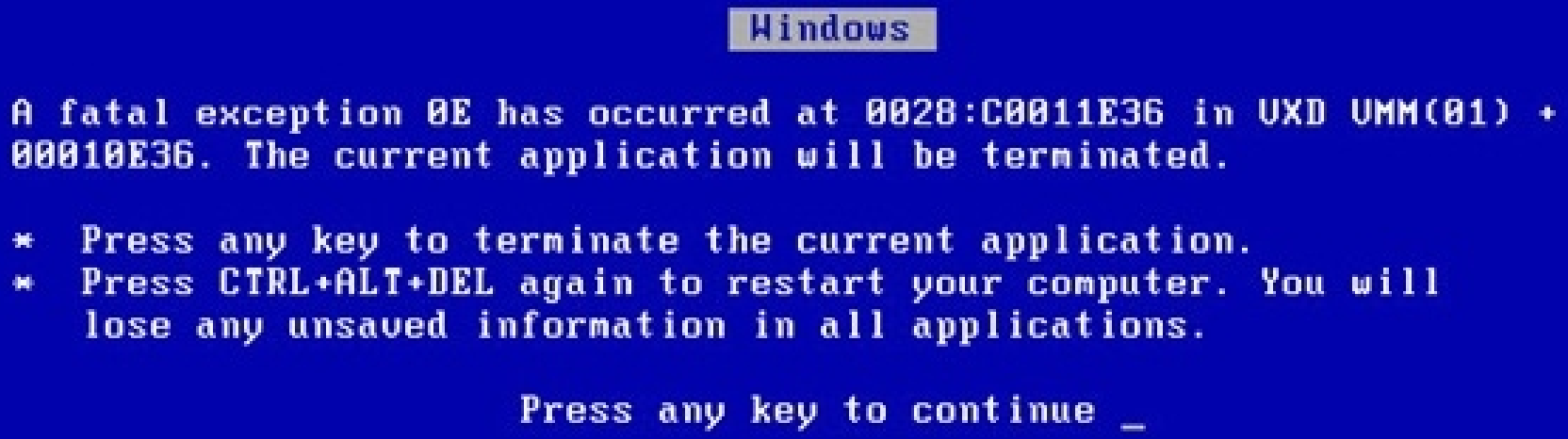
- Operating system are suppose to function flawlessly but OSes like MS Windows , Linux,Mac etc fail to do so.
- These OS have majority of their code in Kernel,and any bug can trash entire system.They violate Principle of Least Authorization.

Proposed System

- It dive into a UNIX-like system called MINIX 3 , with entire operating system running as tightly restricted, independent,user mode processes.

- This structure with mechanism for transparent recovery from crashes results in highly reliable,multiserver OS which looks and feels like most successful UNIX.

Why do Operating System Crash ?



Why do Operating System Crash ?

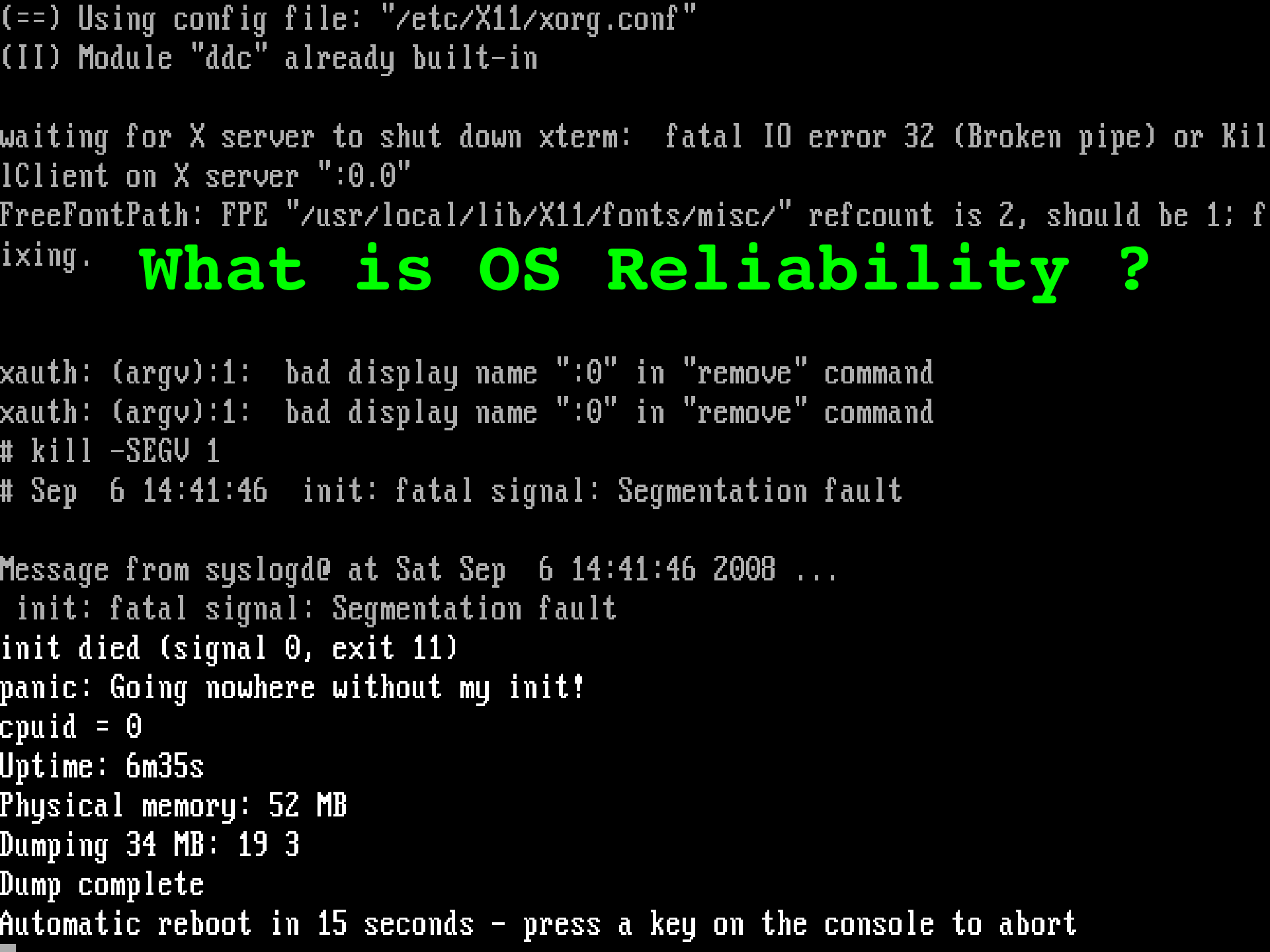
Reason for Operating System Crashes can be track backed to two basic design flaw:

- OS core modules shares too much privileges.
- Lack of fault isolation among OS core modules.

Other factors->

Other factors

- Most users allow third party drivers to be loaded into the OS core.
- Device drivers are written by programmers working for the peripheral manufacturer.
- Even in an open-source effort,it is not necessarily coded experienced volunteer.
- Most errors are due to programming bugs ,studies have shown there are 1-16bugs per 1000 lines of code.


(==) Using config file: "/etc/X11/xorg.conf"
(II) Module "ddc" already built-in
waiting for X server to shut down xterm: fatal IO error 32 (Broken pipe) or Kil
lClient on X server ":0.0"
FreeFontPath: FPE "/usr/local/lib/X11/fonts/misc/" refcount is 2, should be 1; f
ixing.
What is OS Reliability ?
xauth: (argv):1: bad display name ":0" in "remove" command
xauth: (argv):1: bad display name ":0" in "remove" command
kill -SEGV 1
Sep 6 14:41:46 init: fatal signal: Segmentation fault
Message from syslogd@ at Sat Sep 6 14:41:46 2008 ...
 init: fatal signal: Segmentation fault
init died (signal 0, exit 11)
panic: Going nowhere without my init!
cpuid = 0
Uptime: 6m35s
Physical memory: 52 MB
Dumping 34 MB: 19 3
Dump complete
Automatic reboot in 15 seconds - press a key on the console to abort

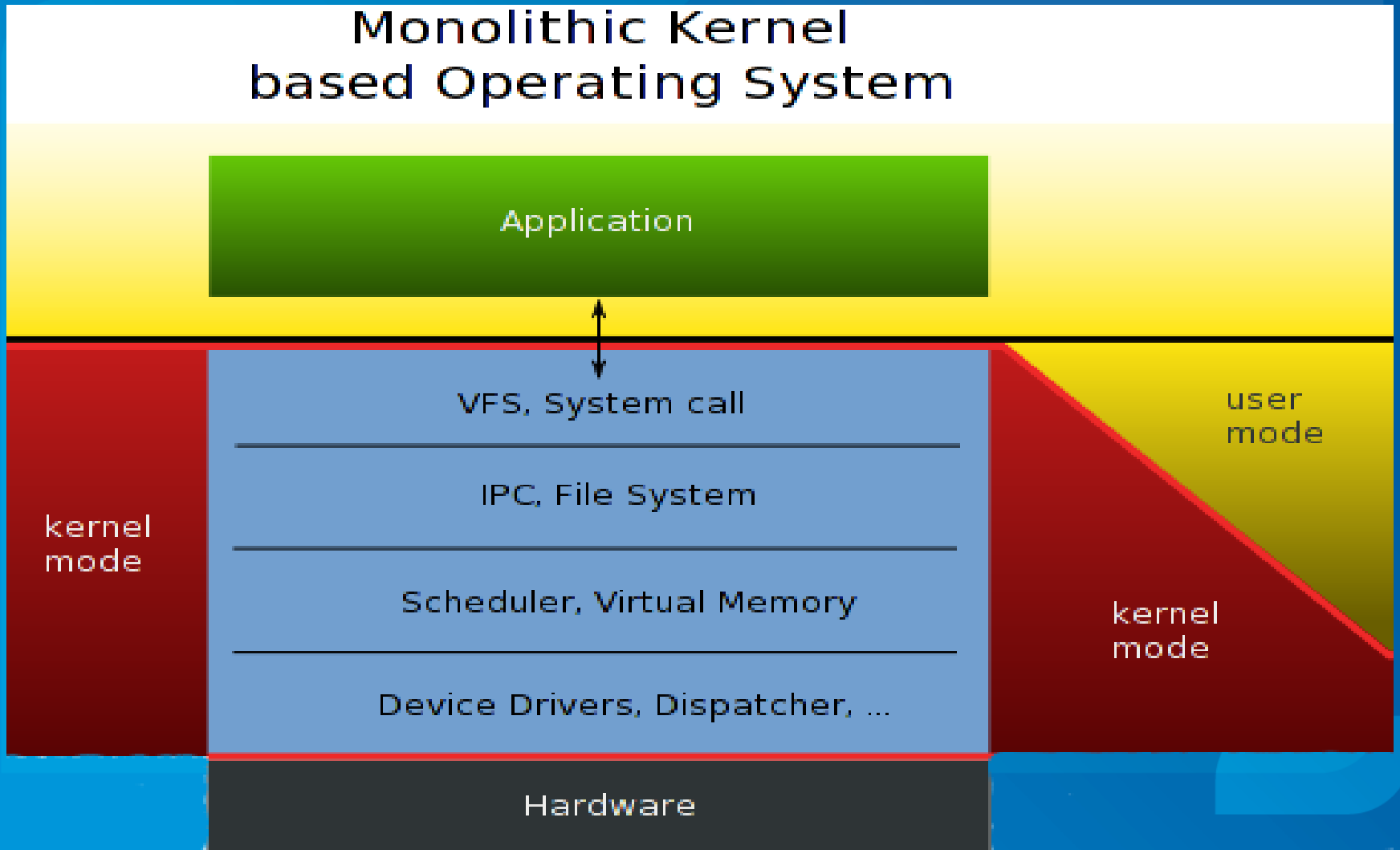
Operating System Design

We aim at reliability,lets look at how the choice of OS kernel design affect reliability.

Two typical kernel design options are:

- Monolithic Kernel
- Micro Kernel

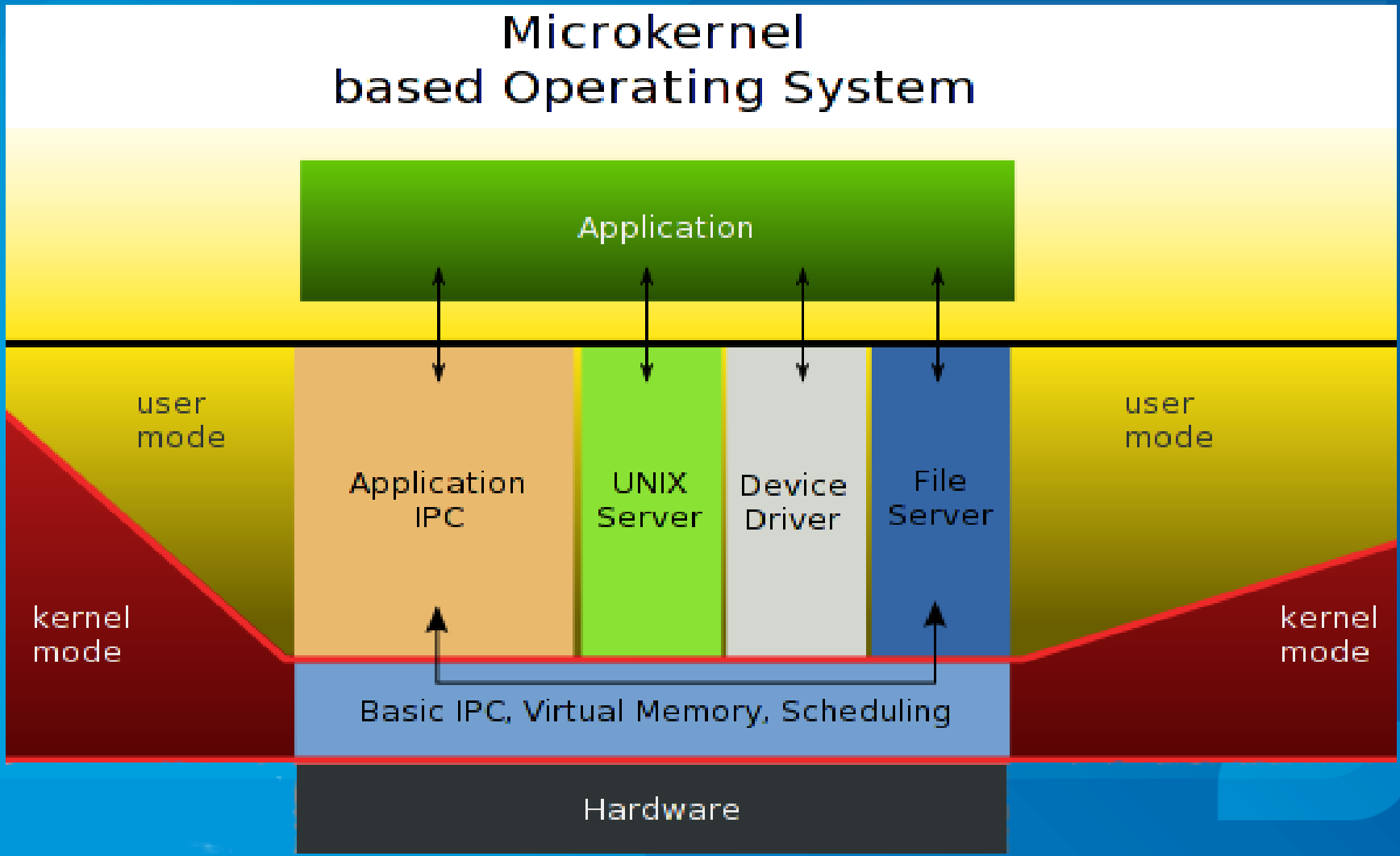
Monolithic Kernel



Problem with Monolithic Kernel

- No proper isolation of faults.
- All codes runs at highest privilege levels.
- Huge amount of code imply many bugs.
- Untrusted,third party code in Kernel.
- Hard to maintain due to complexity.
- Kernel is assumed to be perfect code.
- Fundamental desing requires 3rd party driver to be inserted into the kernel.

MicroKernel Design



Why a Microkernel Design?

- It consist of bare mechanism but no policy.
- Kernel just provides mechanism of interrupt handler ,scheduler,MMU,IPC.
- Std OS functionality like device drivers,file system,network server,high level memory management runs as separate process in a private address space.
- Functionality communicates among them with kernel IPC than shared virtual address space.

Microkernel is Reliable

- In userspace operating system are restricted to what they can do.
- Kernel exports kernel calls to only those OS task that are authorized to call.
- Device drivers have no privileges to perform I/O directly but can request kernel.
- All IPC is done by exchanging small fixed size messages,kernel checks the call being requested to see if its authorized.

Proposed Design Principles

- Simplicity.
- Modularity.
- Least Authorization.
- Fault tolerance.

Simplicity

- We keep our system as simple as possible so that is easy to understand.
- This design avoids problems of resource exhaustion.
- If needed we compromise performance for reliability.
- In kernel we statically declare all the datastructres even though it waste memory,it is much simpler to manage.
- We deliberately not implement multi-threading,so that we can avoid race-conditions,may with cost of performance.

Modularity

- We split our system into collection of independent modules.
- This builds a firewall across modules so that errors can't propagate to other modules.
- We have reduced interdependencies among modules so that failure of one doesn't affect others.
- eg:File System is depends on device drivers but its designed in such a way that its prepared to handle driver failure.

Least Authorization

- A fault in powerful module can bring down entire system,so we reduce the privileges of all the process including servers and drivers.
- Kernel maintains a bit map that governs who can do what.
- For eg:The allowed kernel call map and list of permitted message destinations,this information is in each process table entry and initialized at boot time set by sys admin.

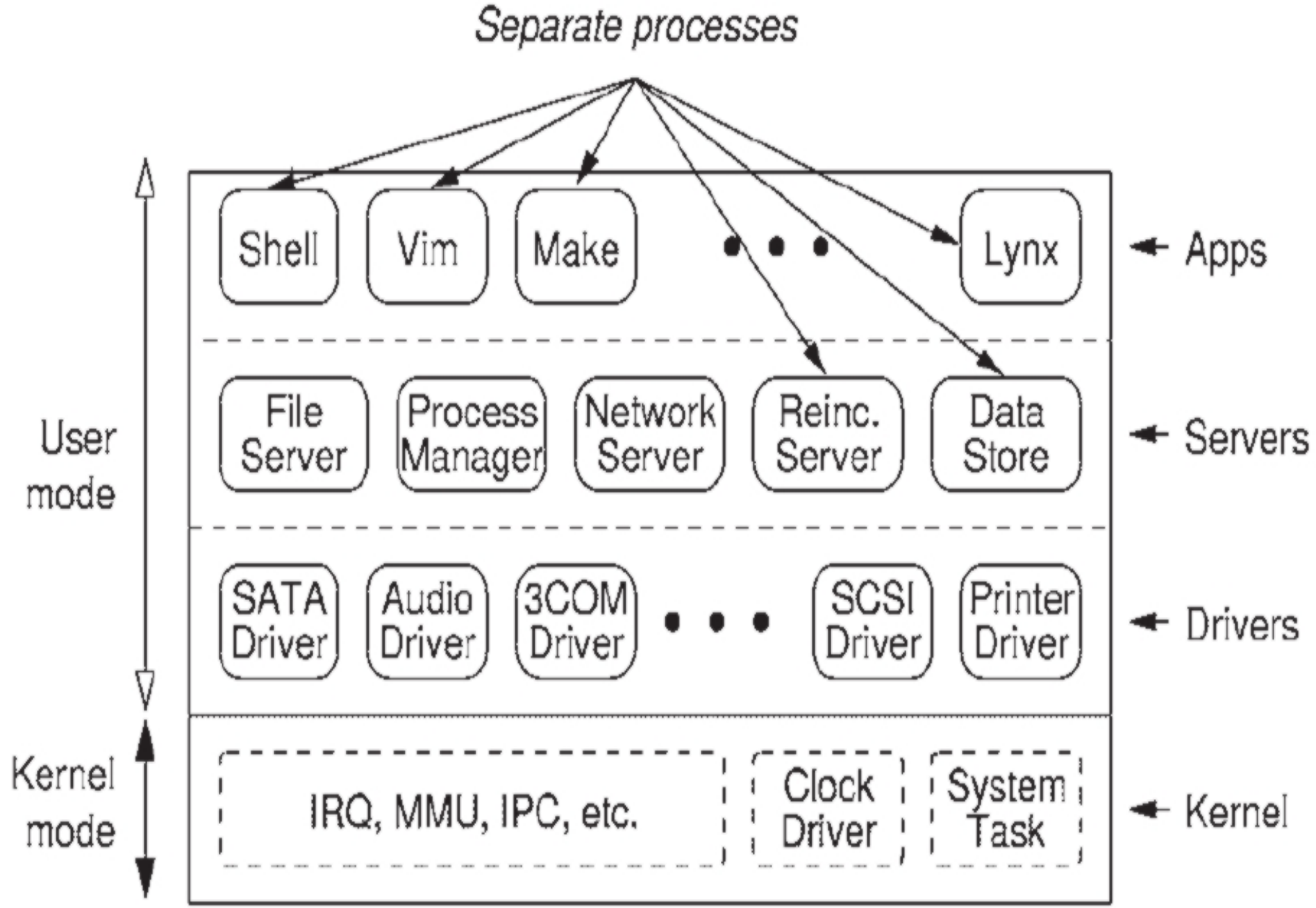
Fault Tolerance

- We have explicitly designed our system to withstnad failures.
- All server and drivers are monitered by special server called reincarnation server.
- If system process unexpectedly exits,this is found by RS and process is restarted.
- Status of each system process is periodically checked to see if its still working,if not malfuctioning server/driver is killed and restarted.
- Fault is detected,rectified on the fly.

MINIX 3 our Reliable UNIX

- *Andrew S. Tanebaum and R&D students.*
- *UNIX Clone.*
- *POSIX Compliant.*
- *Microkernel based.*
- *Full Multiuser.*
- *Multiprograming.*
- *Fault tolerant.*
- *Open Sourced GPL/BSD.*
- *Single chip/low powered devices.*
- *Embbeded Systems.*
- *Education.*

Architecture of MINIX 3



The Kernel

- The kernel consist of under 4000 lines of execu-table code (LoC),which makes easy for us to understand and use verification tools.
- The kernel is responsible for low level operations such as programming the CPU,MMU,interrupt handling and IPC.
- Kernel contains two tasks namely SYS and CLOCK to support the usermode part of the system.

The Kernel (continued)

- Kernel maintains list and bitmaps to restrict the power of all system process,like IPC destination,kernel calls allowed,I/O ports,IRQ lines,memory regions.

- Policies are set by the RS and enforced by kernel at runtime.

Kernel IPC

- Kernel IPC eliminates the need for dynamic allocation of resources.
- The standard request-reply is rendez-vous,if the destinations is not waiting IPC REQUEST blocks the sender until IPC REPLY has been sent,receiver is blocked on IPC SELECT,when no IPC is available.
- Messages are never buffered in kernel ,but copied from sender to receiver speeding up the IPC eliminating buffer overruns.

Kernel IPC (continued)

- For special events IPC NOTIFY primitive can be used to send non blocking notification messages.

- `int notify(endpoint_t dest);`
- `int receive(endpoint_t src, message *m_ptr);`
- `int send(endpoint_t dest, message *m_ptr);`

System Task (SYS)-kernel space

- It is the interface to the kernel for all usermode servers and drivers that requried lowlevel kernel operation.
- All kernel calls in system library are transformed into request message sent to SYS,which handles if caller authorized.
- Kernel calls handled by SYS can be grouped into process ,memory and interrupt management,device I/O and clock services.

System call	Purpose
PROCESS MANAGEMENT	
SYS_EXEC	Execute a process (initialize process)
SYS_EXIT	Exit system service (clean up process)
SYS_FORK	Fork a process (create new process)
SYS_KILL	Kill a process (send a signal)
SYS_NEWMAP	Install new or updated memory map
SYS_XIT	Exit a user process (clean up process)
SYS_SIGCTL	Signal handling (get and process it)
SYS_TRACE	Tracing (control process execution)
COPYING DATA	
SYS_COPY	General copying (virtual and physical)
SYS_PHYSCOPY	Physical copying (arbitrary memory)
SYS_VCOPY	General copying (vector with requests)
SYS_VIRCOPY	Virtual copying (local, remote, BIOS)
DEVICE I/O	
SYS_DEVIO	Read or write a single device register
SYS_SDEVIO	Input or output an entire data buffer
SYS_VDEVIO	Process a vector with multiple requests

Clock Task (CLOCK)-kernel space

- It is responsible for accounting CPU usage,scheduling when quantum expires, managing watchdog timers and interacting with hardware clock.
- Clock registers an interrupt handler ,that is run on every clock tick.
- ISR for clock only increments the CPU usage and decrements the scheduling quantum.
- SYS provides interface from CLOCK to servers and drivers.

User Space Servers

- The size of the servers approximately range from 1000-3000 LoC /server,which helps us in understanding.
- This extends the UNIX philosophy of limited responsibility and power to OS desing level.
- We here implement POSIX comformant,multi server operating system.
- Servers and and drivers cooperate using kernel IPC to provide the functionality of ordinary UNIX.

User Space servers (continued)

- PM (Process manager)
- MM (Memory manger)
- FS (File System)
- DS (Data Store)
- RS (Reincarnation Server)

Process Manager

- Together with FS,PM implements the POSIX interface.
- PM is responsible for process mana-ment,such as creating and removing processes,assig ning process ID and priorities and flow of execution,also responsible for POSIX signaling.
- PM implements all the process management policies,for kernel all process are the same,all it does is to schedule the highest priority ready process.

Memory Manager

- To allowing porting,we make use of hardware-independent segmented memory.
- Each process has text that can be shared with process that execute the same program.
- System process can be granded to acess aditional memory like video memory.
- Text segments of all process are RDONLY.
- Stack and Data segments are not executable.
- MM maintains the list of free memory region.

File Server

- It is an ordinary file server that handles the POSIX calls like read(),open(),write(),etc.
- File system blocks are buffered into FS's buffer cache.
- Caches are periodically written to disk.
- Current MINIX 3 support only one File system,VFS (Virtual File system),is on the way,that supports multiple file systems.

Data Store

- Small data base server with publish subscribe functionality.
- System process use DS to store data privately.
- A restarting system server can use DS to restore its state back.
- It is glue between OS components ,producer can publish data with an ID,and consumer can subscribe to events by producer with the ID.

Reincarnation Server

- It is the central component responsible for managing all OS server and drivers.
- There is policy script associated with each driver and servers.
- A utility called service.
- RS adopts all process in boot image as child.
- RS does a periodic check on driver/server
- Status request from drivers/server are nonblocking.
- RS replace malfunctioning task with fresh one.

Device drivers

- In MINIX 3 we have ATA,SATA,floppy and RAM disk ,keyboard,display drivers etc.
- Each driver in MINIX 3 runs as a user process ,preventing the fault from spreading.
- Not all bugs can be cured by restarting the failed drivers,but certain bugs only requires a restart.
- Well more can be set in the policy script,to be set up in the event of driver crash.

MINIX 3 Reliability Features

- Reducing the Number of kernel bugs.
- Reducing the bug power.
- Recovering from failures.
- Limited Buffer Overruns.
- Ensuring Reliable IPC
- Restricting IPC
- Avoiding Deadlocks.
- Restricting Driver functionality.
- Denying Access to I/O ports.
- Parameter Checking
- Catching bad pointers.
- Taming Infinite Loop.

Reducing the Number of kernel bugs

- We have tiny kernel,with almost 4000LoC ,which is well understood,when compared to 2.5 million LoC of Linux.
- More chances of finding bugs.
- More code mean more bugs,1 16bugs per 1000lines.

Reducing the Bug power

- When bug is triggered the effect would be less devastating by converting it into usermode bug than kernel mode bug.
- eg:User mode sound driver that tries to dereference a bad pointer is killed by RS server,causing sound to stop while rest of the system running.
- A similar kernel mode sound driver,the dereferencing is allowed which may over write the stack return address,causing entire system to crash.

Recovering From failures

- Servers and drivers are forked by system process called reincarnation server.
- If at all server/driver terminates this is notified to RS,as it is the parent.
- RS polls the children periodically.
- Monolithic systems doesn't have method to detect faulty drivers on fly.

Limiting Buffer Overruns

- Buffer overflow is heavily exploited by viruses and worms.
- Our system provides a prevention for this than cure.
- Since our kernel is static allocation,problem doesn't affect kernel.
- Our system only allows execution of code in RDONLY text segment.
- Our stack and datasegment is not executable.
- Worst case is overwriting return address in stack.

Ensuring Reliable IPC

- Our synchronous message passing mechanism is redezvous which enables buffering and buffer management.
- If receiver is not waiting SEND blocks the caller,same with receiver.
- Asynchronous method is NOTIFY.
- We make use of short fixed sized messages,thus buffer management is easy.

Restricting IPC

- IPC must be tightly controlled,our re-dezvous mechanism can be used hang the system with deadlock.
- IPC primitive called SENDREC,which combines send and receive in and single call.It blocks the caller until reply is received.
- SENDREC is the only method that ordinary process can use.
- Kernel maintains a bit map per process.

Avoiding Deadlocks

- IPC is synchronous SEND and RECEIVE done simultaneously deadlock.
- User process only SENDREC() to servers which provides POSIX service.
- We make use of NOTIFY() to send message from kernel space to userspace which is non blocking.
- Notification is stored in destination process table entry until RECEIVE.

Restricting Driver Functionality

- Kernel exports limited functions.
- Kernel maintains a bit map of calls a driver can call.
- Some drivers need only read() call,and would never need a write(),allowing write() to these drivers is a risk.
- In monolithic kernel any one is allowed to call any procedure,even write into other ports.

Denying Acess to IO ports

- For each driver kernel maintains list of ports,the driver is privileged to use.
- Read and write access is protected separately,process with rdonly try to do write is returned an error.
- Only kernel is allowed to do IO operations.
- We do compromise reliability over performance.

Parameter Checking

- Well a complete parameter checking is not possible.
- But kernel can block when a driver tries to write a block of data using physical addressing.
- But with a virtual addressing kernel can't tell if its a valid address,but atleast it can check if the address is valid within users address space.
- Monolithic any thing is possible.

Catching Bad Pointers

- C/C++ programs suffer from great pointer errors.
- Dereferencing a bad pointer leads to segmentation fault.
- In our design a bad pointer issue done by a driver / server would be killed and core would be dumped.
- Thus RS restarts the Killed driver / servers.

Taming Infinite Loop

- When a driver is stuck in an infinite loop it consumes too much CPU time.
- Scheduler notice this and reduce the priority until it becomes idle.
- Since driver would stop responding to RS,RS reads the policy script and either restarts it or do as in policy script.
- In monolithic entire system hangs.

MINIX 3 Performance

- Measuring the performance is bit tricky.
- Running benchmarks on MINIX 3 other clones like BSD or Linux would have helped.
- But these kernels have been compiled with different compilers and follow different algorithms for memory / filesystem etc.
- So benchmarks were conducted with two systems that differ only in one parameter.

MINIX 3 Performance (continued)

- We compared the benchmarks run on MINIX 2 which is a hybrid microkernel with drivers in the kernel.
- All the test were conducted in 2.2GHz.

System Call performance

Call	MINIX 2	MINIX 3	Δ	Ratio
getpid	0.831	1.011	0.180	1.22
lseek	0.721	0.797	0.076	1.11
open+close	3.048	3.315	0.267	1.09
read 64k+lseek	81.207	87.999	6.792	1.08
write 64k+lseek	80.165	86.832	6.667	1.08
creat+wr+del	12.465	13.465	1.000	1.08
fork	10.499	12.399	1.900	1.18
fork+exec	38.832	43.365	4.533	1.12
mkdir+rmdir	13.357	14.265	0.908	1.07
rename	5.852	6.812	0.960	1.16
Average				**1.12**

Table 1:System call times for kernel call vs user mode drivers ,unit of μs.

Application test Results

Program	2.0.4	3.0.1	Δ	Ratio
Build image	3.630	3.878	0.248	1.07
Build POSIX tests	1.455	1.577	0.122	1.08
Sort	99.2	103.4	4.2	1.04
Sed	17.7	18.8	1.1	1.06
Grep	13.7	13.9	0.2	1.01
Prep	145.6	159.3	13.7	1.09
Uuencode	19.6	21.2	1.6	1.08
Average				**1.06**

Table 2:Run times in seconds for various test programs first two test where run repeatedly in a loop while others once to exclude the effect of cache.

Summery of Performance

- Reading and writing are 8% slower in MINIX 3 due to extra two messages and two context switches for user mode drivers.
- The getpid() call a simple call from user process to process manager takes 180 ns more in MINIX 3 than MINIX 2.
- Average performance hit in application testing were 6%.
- For reliability we pay a performace loss of 5-10%.We consider it a price paying.

Conclusion

To achieve reliability design was guided by simplicity,modularity,POLA and fault tolerance.An understandable kernel means fewer kernel bugs,most part of operating system works as a isolated process in userspace.Servers and drivers are monitored by a special server called RS,for the reduction of operating system crashes we may 5-10% performance loss.Drivers and other components are not rendered bug free but the effect bugs is reduced.

Concluding,we have seen operating system reliablitly can be improved with careful desigin even at cost of performance.

References

[1] T.J. Ostrand and E.J. Weyuker. The Distribution of Faults in a Large Industrial Software System. In Proc.of the 2002 ACM SIGSOFT Int'l Symp. on Software Testing and Analysis, Pages 55–64,86-96. ACM, 2002.

[2] V. Basili and B. Perricone. Software Errors and Complexity: An Empirical Investigation.Commun. of the ACM, 21(1):42–52, Jan. 1984.

[3] A. Chou, J. Yang, B. Chelf, S. Hallem, and D. Engler. An Empirical Study of Operating System Errors. InProc. 18th ACM Symp. on Oper. Syst. Prin., pages 73–88, 2001.

[4] A. Bricker, M. Gien, M. Guillemont, J. Lipkis, D. Orr, and M. Rozier. A New Look at Microkernel-Based UNIX Operating Systems: Lessons in Performance and Compatibility. In Proc. EurOpen Spring 1991 Conf., pages 13–32, May 1991.

[5] D. Cheriton. The V Kernel: A Software Base for Distributed Systems. IEEE Software, 1(2):19–42, Apr 1984. 6th Symp. on Oper. Syst. Design and Impl., pages 17–30, Dec. 2004.

[6] Herder, J.N., Bos, H., Tanenbaum, A.S.: A Lightweight Method for Building Reliable Operating Systems Despite Unreliable Device Drivers. In: Technical Report IR-CS-018[www.cs.vu.nl/»jnherder/ir-cs- 018.pdf], Vrije Universiteit (2006).

[7] Jorrit N. Herder,Herber Bos,Ben Gras,Philip Homburg and Andrew S. Tanenbaum: MINIX 3 A highly reliable,Self Repa ring Operating System ,Vrije University.

[8] Jorrit N. Herder,Herber Bos,Ben Gras,Philip Homburg and Andrew S. Tanenbaum.Reorganizing UNIX for Reliability,Computer Science Dept.,Vrije University Amsterdam

[9] Andrew S. Tanenbaum,Albert S. Woodhull, The MINIX book,Operating System Design and Implimentation Third edition

Computers are like air conditioners, they stop working once you

Open

Windows

A fatal exception 0E has occurred at 0028:C0011E36 in VXD VMM(01) +
00010E36. The current application will be terminated.

* Press any key to terminate the current application.
* Press CTRL+ALT+DEL again to restart your computer. You will
  lose any unsaved information in all applications.

Press any key to continue _

Thank you ;-)