System Security

Introduction to Minix
(Lecture 3)

Roberto Guanciale

Outline of today

- Minix architecture
- Working with Minix
- Your first Minix Application
- Your first Minix Device Driver

Minix Architexture

0 (NASA)/ 1 (MS)/ 15 (Avg)

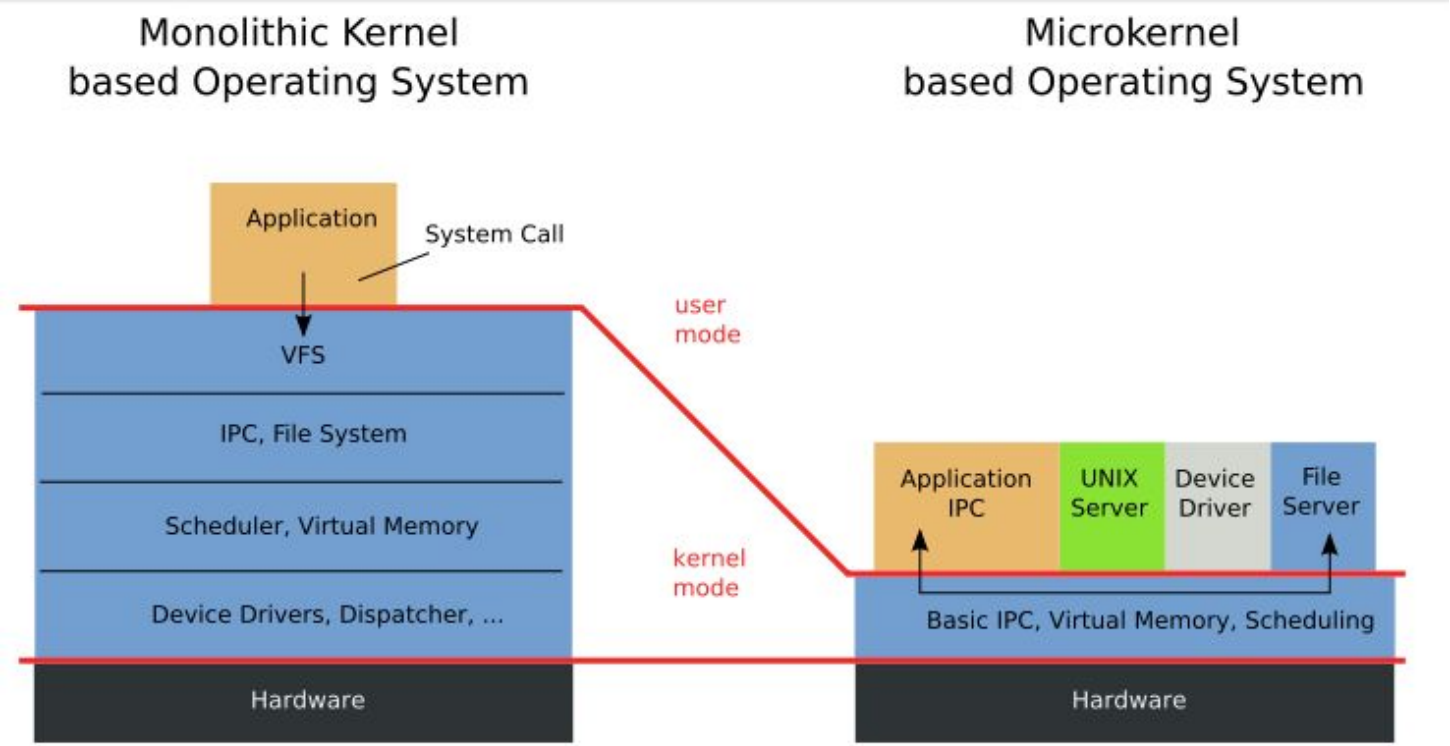
Bugs per 1000 LOC

https://www.vidarholen.net/contents/wordcount/

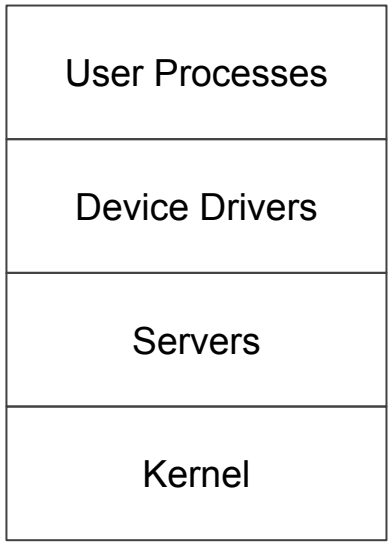
About Minix

- Minix - Mini Unix (Minix) basically, a UNIX - compatible operating system.
- Minix is small in size, with microkernel-based design.
- Minix has been kept (relatively) small and simple.
- A POSIX-compliant, preemptive multitasking operating system
- Support for TCP/IP
- Source code: C language
- BSD licence
- Support for x86, ARMv7

Architecture

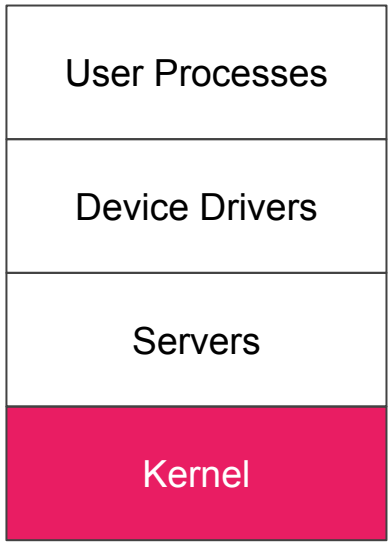


Architecture



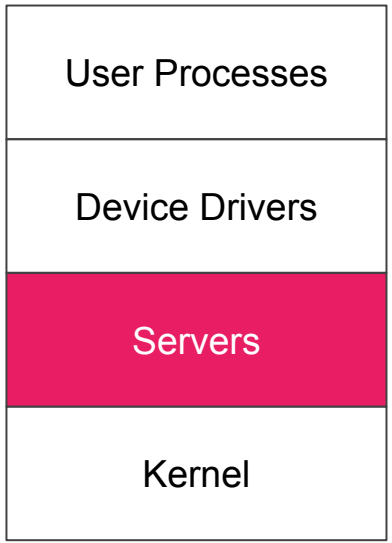
https://wiki.minix3.org/doku.php?id=developersguide:overviewofminixarchitecture

Architecture



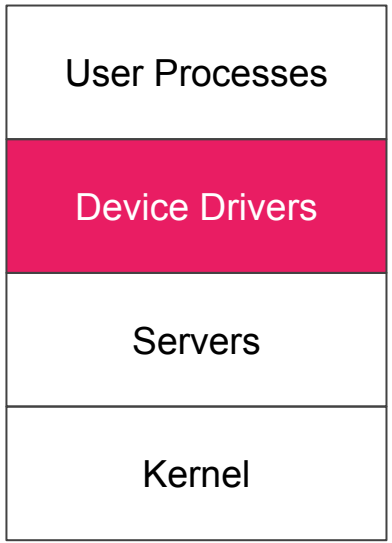
- central component of the OS
- manages the system's resources
  - interface between hardware and software
- lowest-level of abstraction
- only component executed in kernel mode
  - Interrupts
  - Scheduling
  - Processes
  - Inter Process Communication (IPC)

Architecture



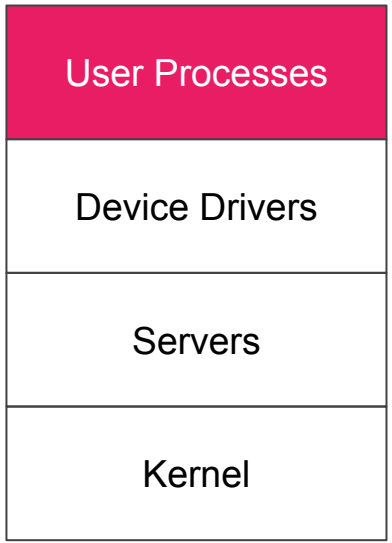
- Offer services to the rest of the system
- Interact with drivers/servers/kernel
- Virtual Memory
  - Allocates physical memory
  - Changes page tables
- Virtual File System
  - Abstraction from of the actual file systems
  - Has special privileges to allows device drivers to access memory of other processes
- Actual File System
- Process Manager
  - Handles process informations
- Reincarnation Server
  - Start/ping/restart other servers
- Network stack

Architecture



- handle a specific hardware component.
- each driver has access through the micro-kernel to its hardware resources (input/output ports, interrupts)
- mass storage (floppy disk, hard disk, CD/DVD/Blu-Ray/tape drive…)
- network card
- printer and serial port drivers.

Architecture



- User-land processes are processes just like drivers and servers
- they do not have specific privileges
- to access facilities such as the file system or the network, they perform requests to servers
- application programming interface (API) is POSIX, a set of common system calls (not the Minix micro-kernel ones) and libraries functions all Unix-like operating systems share.
  - Mkdir, fork, execl, pipe, kill, networking, mmap

Reliability

https://wiki.minix3.org/doku.php?id=www:documentation:reliability

---

1. Reduce kernel size
2. Cage the bugs
3. Limit drivers' memory access
4. Survive bad pointers (reincarnation)
5. Tame infinite loops (reincarnation)
6. Limit damage from buffer overruns (partially true)
7. Restrict access to kernel functions
8. Restrict access to I/O ports
9. Restrict communication with OS components
10. Reincarnate dead or sick drivers
11. Integrate interrupts and messages

Message Passing

https://wiki.minix3.org/doku.php?id=developersguide:messagepassing

---

- fixed-length 64 bytes of data
  - The endpoint: a 4-byte identifier of who receives the message
  - Message type : a 4-byte message type identifier
  - Payload : 56 bytes of data
    - If more that 56 bytes, use memory grants
- each endpoint has a 1 message receive buffer inside the micro-kernel
- API
  - SEND: the sender is blocked until the message is delivered
  - RECEIVE: the process is blocked until a message is delivered to them
  - SENDREC: the sender is blocked until it receives a reply from the receiver.

Memory Grants

https://wiki.minix3.org/doku.php?id=developersguide:memorygrants

---

- allow processes to transfer large amounts of data
- processes grant access to regions of memory to another process
- attributes:
  - Access mode: read/write/read-write
  - Virtual memory region (byte granularity)
- cpf_* functions wrap system calls to manage grants
- types:
  - Direct
  - Indirect
  - Magic (only VFS)
- grant-id communicated via IPC
- grantee uses safecopies syscall specifying grant-id

Developing Minix

Developing for Minix

https://wiki.minix3.org/doku.php?id=developersguide:crosscompiling

---

- Cross compiling:
  - building MINIX from another operating system
  - suggestion use Ubuntu 18.04
    - There is a VM, but is slow
- Native compiling:
  - building MINIX from within MINIX
  - High probability that you break everything
    - you change order of arguments of a system call, and you change accordingly the useland libs
    - You rebuild the system, the kernel and userland are recompiled and installed
    - You are still running the old kernel, but new processes (i.e. GCC) start to use the new userland

Crosscompiling

https://wiki.minix3.org/doku.php?id=developersguide:crosscompiling

---

- install dependencies
  a. apt-get install build-essential curl git zlibc zlib1g zlib1g-dev g++
- fetch sources
  a. https://gits-15.sys.kth.se/robertog/system-sec-minix
- bash ./releasetools/x86_hdimage.sh
  a. Build the toolchain (i.e. clang)
  b. Build kernel, server, driver, libs, applications (i.e. vi), games
  c. Package everything and create a disk image
- test with qemu (or your own computer if you are really brave)

Crosscompiling

https://wiki.minix3.org/doku.php?id=developersguide:crosscompiling

---

- fetch sources
  a. `git clone git://git.minix3.org/minix`
  b. `git clone https://gits-15.sys.kth.se/robertog/system-sec-minix.git`
- use “master” (other branches are examples)

Crosscompiling

https://wiki.minix3.org/doku.php?id=developersguide:crosscompiling

---

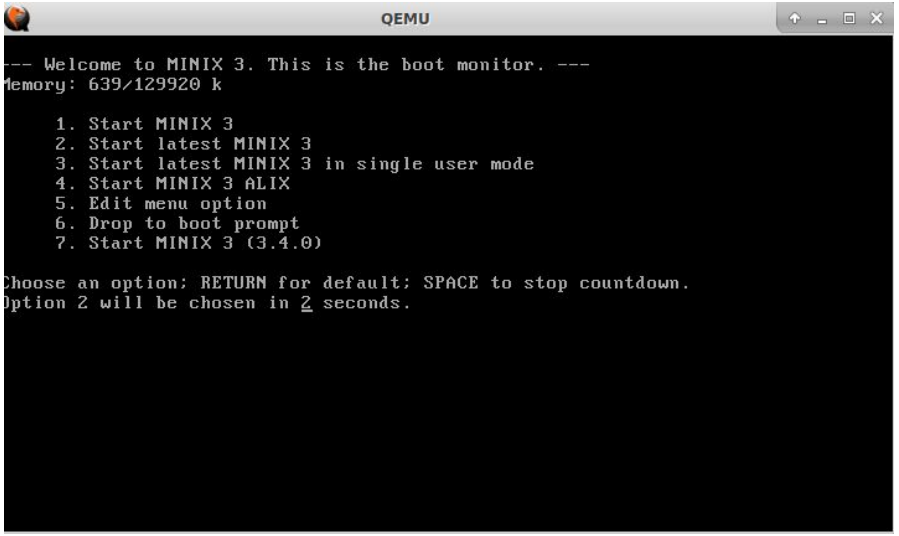
- bash ./releasetools/x86_hdimage.sh
  a. Build the toolchain (i.e. clang)
     - First time takes several hours
     - ~~There are problems to link LLVM, either fix the link script or compile adding -l termcap~~
     - It is recompiled if you update your Linux kernel
  b. Build kernel, server, driver, libs, applications (i.e. vi), games
  c. Package everything and create a disk image
- It is not needed to use release tool every time, but until you are confident it highly recommended
- To speed up set ${JOBS=3} in ./releasetools/image.defaults

Running

https://wiki.minix3.org/doku.php?id=usersguide:runningonqemu

- ./releasetools/x86_hdimage.sh prints the qemu command to run the vm
- there is little to no support for external debuggers
  a. You can debug the microkernel, but difficult to debug processes
- Install KVM, it makes the emulation faster
- qemu-system-i386 --enable-kvm -m 256 -hda minix_x86.img

Demo Session



You first Minix Program

“Hello world!”

Developing for Minix

https://wiki.minix3.org/doku.php?id=developersguide:programmingminix

---

- Crosscompilation or In Minix compilation
- Good text about secure programming
  - https://dwheeler.com/secure-programs/Secure-Programs-HOWTO/index.html

The Minix Hello World

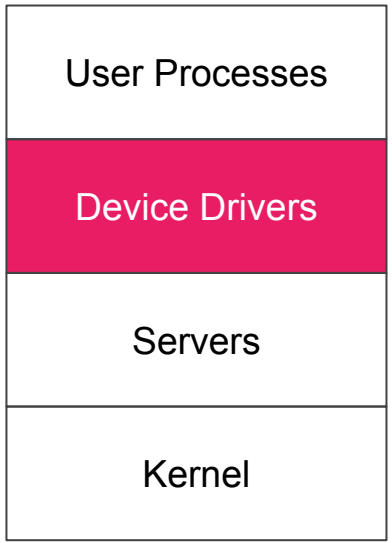
- games/demo01/Makefile
  - BSD Makefile
  - https://wiki.netbsd.org/bsd_make/
- games/Makefile
  - Add demo01 here
- distrib/sets/lists/games/mi and distrib/sets/lists/minix-games/mi
  - Add ./usr/games/demo01
- games/demo01/demo01.c
  - Your hello world app

The Minix Hello World

- Compile and deploy:
  - ./releasetools/x86_hdimage.sh
- Only compile:
  - PATH=$PATH:<???>/obj.i386/tooldir.Linux-<???>/bin/
  - nbmake-i386 clean
  - nbmake-i386

Your first Device Driver

Architecture



- handle a specific hardware component.
- each driver has access through the micro-kernel to its hardware resources (input/output ports, interrupts)
- mass storage (floppy disk, hard disk, CD/DVD/Blu-Ray/tape drive…)
- network card
- printer and serial port drivers.

https://wiki.minix3.org/doku.php?id=developersguide:overviewofminixdrivers

Driver Architecture

- The drivers source code are located in minix/drivers
- The binary executables are located under the /service/ directory

Character Based

- data as a stream of bytes
- do not support the seek
- Printer / Serial Port / TTY / Random/

Block Based

- data in chunks called blocks of a constant number of bytes
- Support seek
  - access to any block at any moment
- hard disk and CD drives

https://wiki.minix3.org/doku.php?id=developersguide:overviewofminixdrivers

Driver Architecture

- Interact with Kernel/VFS/Reincarnation server using specialized Low level protocol
  - Using message passing, memory grants and system calls
- High-level abstraction libraries
  - provide a higher-level abstraction interface
  - shield the drivers from low-level protocols changes

  - blockdriver : for block-oriented drivers
  - chardriver : for generic character-oriented drivers
  - inputdriver : for drivers communicating with the input server
  - netdriver : for network device drivers

https://wiki.minix3.org/doku.php?id=developersguide:driverprogramming

A simple character based device

- Driver source code
  - minix/drivers/mydriver/mydriver.c
  - minix/drivers/mydriver/mydriver.h
- Configuration:
  - minix/drivers/mydriver/mydriver.conf
  - Allows driver to communicate to other services and invoke syscalls
- Makefile
  - minix/drivers/mydriver/Makefile
  - It copies the driver binary in /service and conf in /etc/system.conf.d
  - minix/drivers/Makefile
    - Add mydriver here
- Distribution List
  - distrib/sets/lists/minix-base/mi
  - add /etc/system.conf.d/mydriver and /service/mydriver

https://wiki.minix3.org/doku.php?id=developersguide:driverprogramming

The System Event Framework (SEF)

- System library to deal with system events
- sef_startup()
  - RS will send the driver an initialization message with information on how to initialize properly
  - The driver is expected to reply back with OK or error

1. Driver registers callback
2. Driver calls sef_startup()
3. SEF do the rest and invokes callbacks

https://wiki.minix3.org/doku.php?id=developersguide:driverprogramming

A simple character based device

- To start the driver
  - minix-service up /service/mydriver
- To stop the driver
  - minix-service down mydriver
- Reincarnation Server (RS)
  - Start and stop services
  - minix-service sends a message to RS
  - RS pings the driver
    - If the driver aborted or exited
      - Restart the driver
    - If the driver never replies since is is stuck in a loop
      - Stop and restart
- Driver should reply to keep-a-live messages
  - library libchardriver takes care of various common tasks

https://wiki.minix3.org/doku.php?id=developersguide:driverprogramming

The System Event Framework (SEF)

- sef_setcb_init_fresh
  - fresh start initialization
- sef_setcb_init_lu
  - init after live update
  - there are specific call backs to inform RS that we can be updated and prepare for update
- sef_setcb_init_restart
  - init after restart

https://wiki.minix3.org/doku.php?id=developersguide:driverprogramming

The Chardriver

- struct chardriver
  - Call backs for char based device
  - chardriver library has a main-loop to fetch messages, invoke callbacks, send reply messages
- open/close/read/write
- sys_safecopyto(endpt, grant, 0, (vir_bytes) ptr, size)
  - endpt: id of process requesting read (actually is the process that granted the memory access)
  - grant: grant to a region of memory of another process
  - 0: offset in the receiver buffer
  - ptr: address in driver's virtual memory
  - size: amount of bytes
  - actual copy done by the kernel

https://wiki.minix3.org/doku.php?id=developersguide:driverprogramming

The Chardriver final notes

- minix/include/minix/dmap.h
  - Allocate major 18 to mydriver
- minix/commands/MAKEDEV/MAKEDEV.sh
  - Make device file /dev/mydriver at boot
- etc/usr/rc
  - Start driver at boot

https://wiki.minix3.org/doku.php?id=developersguide:driverprogramming

The Chardriver

```
> mknod /dev/mydriver c 18 0
```

- major ID 18 (look at minix/dmap.h). Identifies device driver
- character device with major number 18 and minor number 0

```
> minix-service up /service/mydriver -dev /dev/mydriver

> cat /dev/mydriver

> minix-service refresh mydriver

> minix-service down mydriver
```

https://wiki.minix3.org/doku.php?id=developersguide:driverprogramming

Example projects

- Anti-code injection (2018)
  - Monitor exec to only execute signed code / prevent bad programs
  - Inspect process text area to detect alterations / check signatures
- Control flow integrity (2018)
  - Monitor syscalls and enforce caller PC integrity
- Firewall (2018)
- Run-time patching (2018)
- Malware monitor and analysis
- Anti DOS
  - Monitor memory allocation / fork / fopen to limit used resources
- Anti buffer overflows
  - Inspect stack to identify stack corruptions
- Extend GDB support
- Re-randomization

Questions?