# Escape

Nils Asmussen

MKC, 06/11/2020

# Outline

# Outline

# Motivation

## Beginning

- Writing an OS alone? That's way too much work!
- Port of UNIX32V to ECO32 during my studies
- Started with Escape in October 2008

## Goals

- Learn about operating systems and related topics
- Experiment: What works well and what doesn't?
- What problems occur and how can they be solved?

# Overview

## Basic Properties

- UNIX-like microkernel-based OS
- Open source, available on `github.com/Nils-TUD/Escape`
- Mostly written in `C++`, some parts in `C`
- Runs on x86, x86_64, ECO32 and MMIX
- Only third-party code: `libgcc`, `libsupc++`, x86emu, inflate

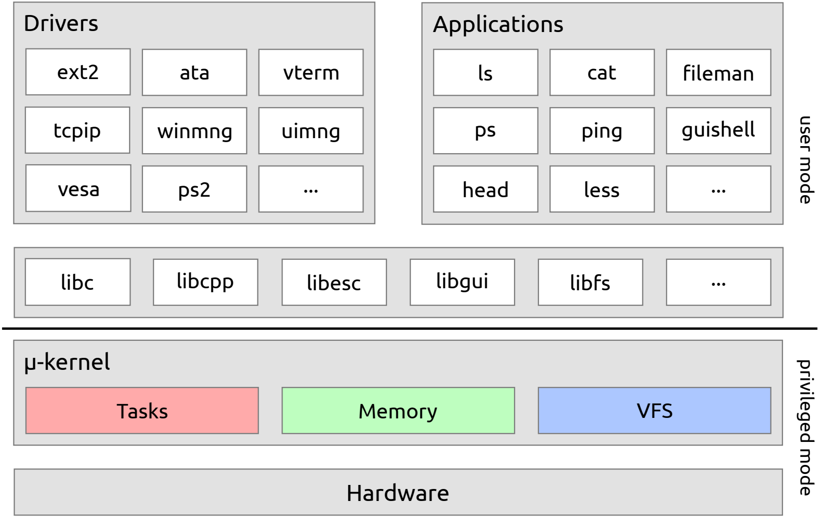
## ECO32

MIPS-like, 32-bit big-endian RISC architecture, developed by Prof. Geisse for lectures and research

## MMIX

64-bit big-endian RISC architecture of Donald Knuth as a successor for MIX (the abstract machine from TAOCP)

# Overview


Drivers
ext2
ata
vterm
tcpip
winmng
uimng
vesa
ps2
...
Applications
ls
cat
fileman
ps
ping
guishell
head
less
...
user mode
libc
libcpp
libesc
libgui
libfs
...
µ-kernel
Tasks
Memory
VFS
privileged mode
Hardware

# Outline

# Processes and Threads

## Process

- Virtual address space
- File descriptors
- Mountspace
- Threads (at least one)
- ...

## Thread

- User and kernel stack
- State (running, ready, blocked, ...)
- Scheduled by a round-robin scheduler with priorities
- Signals
- ...

# Processes and Threads

## Synchronization

- Process-local semaphores (can also be created for interrupts)
- Global semaphores, named by a path to a file
- Userspace builds other synchronization primitives on top
    - Combination of atomic ops and process-local semaphores
    - Readers writer lock
    - ...

## Priority Management

- Priorities are dynamically adjusted based on compute intensity
- High CPU usage → downgrade, low CPU usage → upgrade

# Outline

# Memory Management

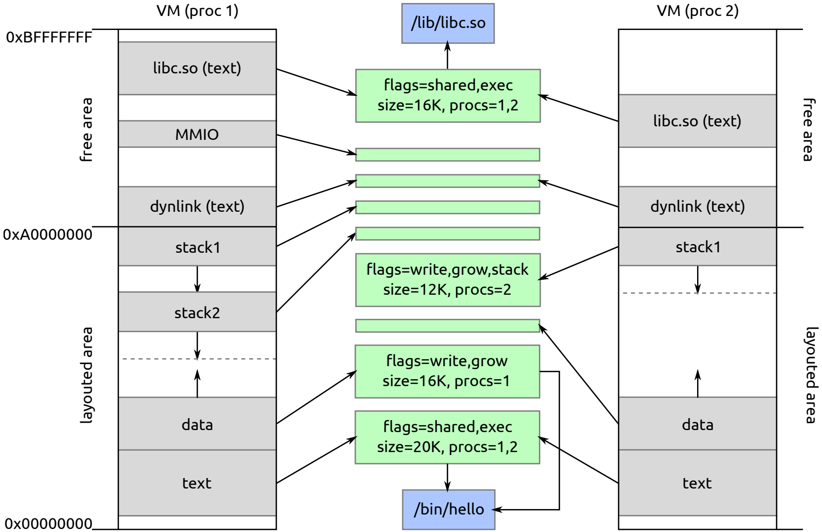
## Physical Memory

- Mostly, memory is managed by a stack (fast for single frames)
- A small part handled by a bitmap for contiguous phys. memory

## Virtual Memory

- Kernel part is shared among all processes
- User part is managed by a region-based concept
- `mmap`-like interface for the userspace

# Virtual Memory Management



# Outline

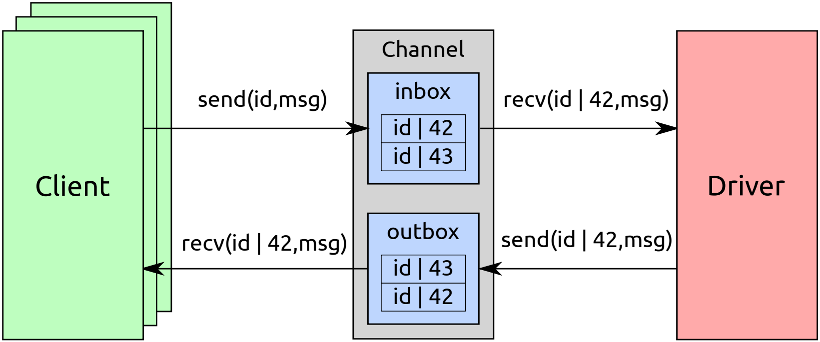
## Basics

- The kernel provides the virtual file system
- System-calls: `open`, `read`, `mkdir`, `mount`, ...
- It's used for:
  1. Provide information about the state of the system
  2. Access userspace filesystems
  3. Access devices
  4. Access interrupts

# Drivers and Devices

- Drivers are ordinary user programs
- They create devices via the system call `createdev`
- These are usually put into /dev
- Devices can also be used to implement on-demand-generated files (such as /sys/net/sockets)
- Communication is based on asynchronous message passing

# Message Passing



# Devices Can Behave Like Files

- As in UNIX: Devices should be accessable like files
- Messages: `FILE_OPEN`, `FILE_READ`, `FILE_WRITE`, `FILE_CLOSE`
- Devices may support a subset of these message
- Kernel handles communication for `open`/`read`/`write`/`close`
- Type of file transparent for applications

# Devices Can Behave Like Filesystems

- Messages: `FS_OPEN`, `FS_READ`, `FS_WRITE`, `FS_CLOSE`, `FS_STAT`, `FS_SYNC`, `FS_LINK`, `FS_UNLINK`, `FS_RENAME`, `FS_MKDIR`, `FS_RMDIR`, `FS_CHMOD`, `FS_CHOWN`
- Kernel handles communication, if syscall refers to userspace fs
- Filesystems are mounted using the `mount` system call

## Achieving Higher Throughput

- Copying everything twice hurts for large amounts of data
- `sharebuf` establishes shmem between client and driver
- Easy to use: just call `sharebuf` once and use this as the buffer
- Clients don't need to care whether a driver supports it or not
- Drivers need to handle `DEV_SHFILE` to support it
- In `read`/`write`, they check if SHM should be used

# Achieving Higher Throughput – Code Example

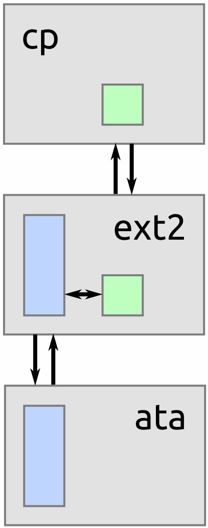
```
int fd = open("/dev/zero",O_READ);
static char buf[SIZE];



while(read(fd,buf,SIZE)) > 0) {
    // ...
}

close(fd);
```

# Achieving Higher Throughput – Code Example

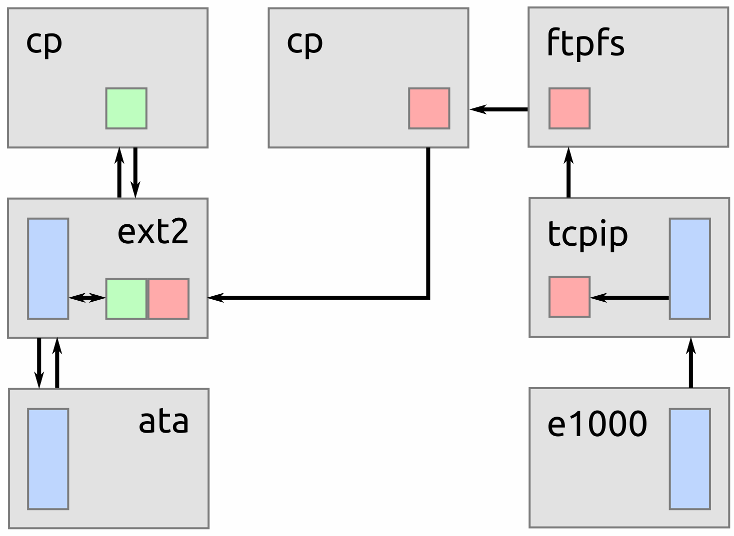
```c
int fd = open("/dev/zero",O_READ);

static char buf[SIZE];




while(read(fd,buf,SIZE)) > 0) {
    // ...
}


close(fd);
```

```c
int fd = open("/dev/zero",O_READ);

void *buf;
if(sharebuf(fd,SIZE,&buf,0) < 0) {
    if(buf == NULL)
        error("Unable to mmap buf");
}

while(read(fd,buf,SIZE) > 0) {
    // ...
}

destroybuf(buf);
close(fd);
```

# Achieving Higher Throughput – Usage Example



# Achieving Higher Throughput – Usage Example



# File Exchange

- Files (=capabilities) can be exchanged via channel
- Client can delegate/obtain files from driver:
  - `int delegate(int chan,int fd,uint perm,int arg)`
  - `int obtain(int chan,int arg)`
- Used for:
  - Establishing shared memory
  - Connecting control and event channel of uimng
  - Accepting incoming network connections (`accept`)
  - ...

# File Descriptors For Everything

## Interrupts

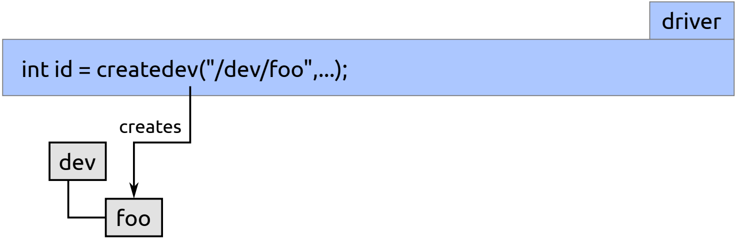
- Escape uses semaphores for interrupts
- For each interrupt, Escape creates a file /sys/irq/$irq
- Syscall `semirqcrt` expects fd for IRQ file
- On an IRQ, all semaphores in the list are up'ed

## Signals

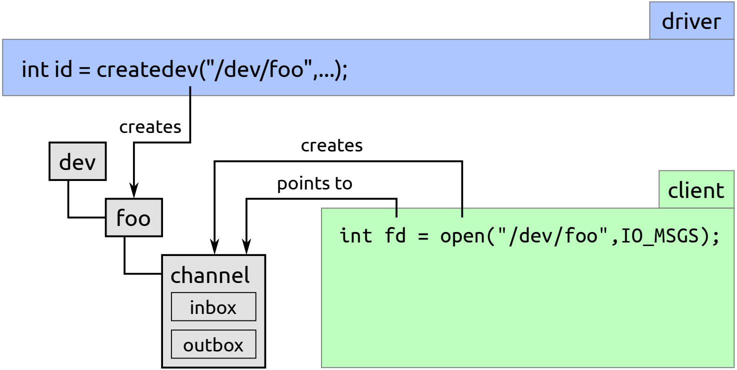
- The `kill` syscall expects fd for process directory
- Only if it has write permission, the signal can be sent

# Outline

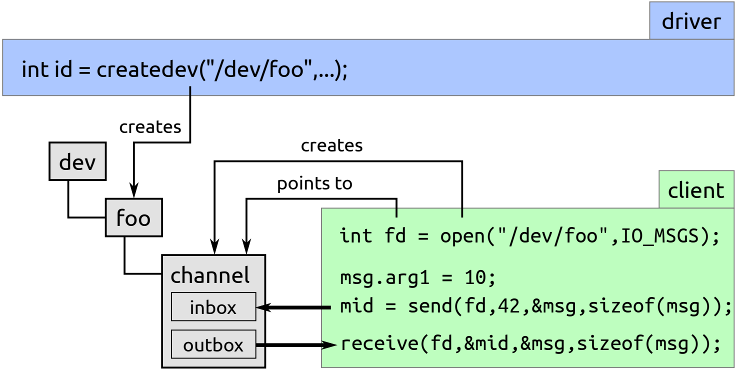
# IPC between Client and Driver (Low Level)



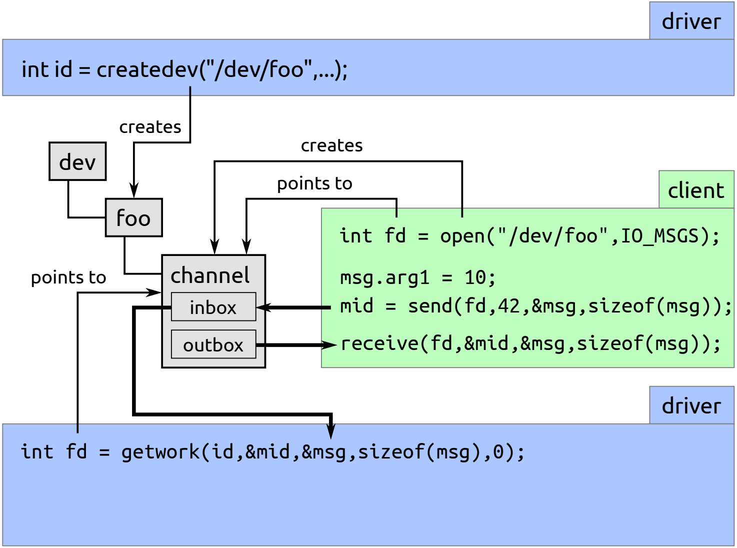
# IPC between Client and Driver (Low Level)



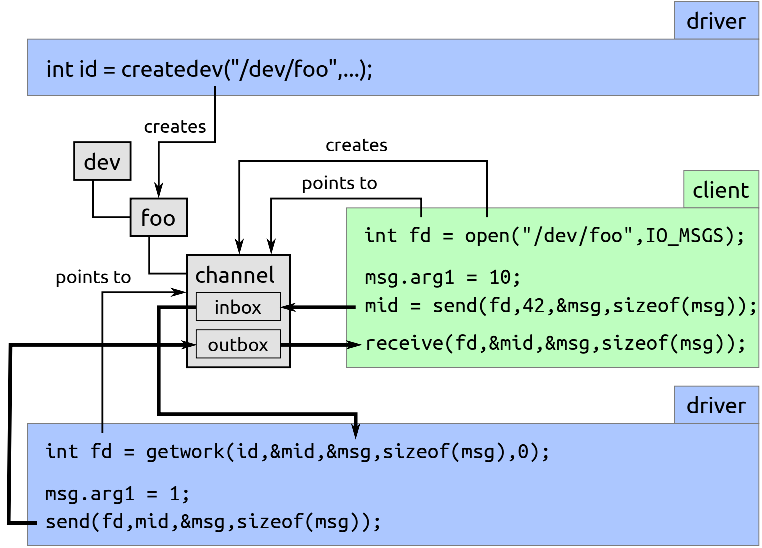
# IPC between Client and Driver (Low Level)



# IPC between Client and Driver (Low Level)



# IPC between Client and Driver (Low Level)



# Driver Example: /dev/zero

```
struct ZeroDevice : public ClientDevice<> {
    explicit ZeroDevice(const char *name, mode_t mode)
        : ClientDevice(name, mode, DEV_TYPE_BLOCK, DEV_OPEN | DEV_DELEGATE |
            DEV_READ | DEV_CLOSE) {
        set(MSG_FILE_READ, std::make_memfun(this, &ZeroDevice::read));
    }

    void read(IPCStream &is) {
        static char zeros[BUF_SIZE];
        Client *c = get(is.fd());
        FileRead::Request r;
        is >> r;

        if(r.shmemoff != -1)
            memset(c->shm() + r.shmemoff, 0, r.count);
        is << FileRead::Response(r.count) << Reply();
        if(r.shmemoff == -1 && r.count)
            is << ReplyData(zeros, r.count);
    }
};

int main() {
    ZeroDevice dev("/dev/zero", 0400);
    dev.loop();
    return EXIT_SUCCESS;
}
```

# Client Example: vterm

```
// get console-size
ipc::VTerm vterm(std::env::get("TERM").c_str());
ipc::Screen::Mode mode = vterm.getMode();

// implementation of vterm.getMode():
Mode getMode() {
    Mode mode;
    int res;
    _is << SendReceive(MSG_SCR_GETMODE) >> res >> mode;
    if(res < 0)
        VTHROWE("getMode()", res);
    return mode;
}
```

# Outline

# General Idea

## Goals

- Keep the powerful and convenient UNIX concepts
- Improve the security, reliability and maintainability

## Approach

- Structure it as a microkernel-based system
- Permissions can only be downgraded (e.g., no setuid)
- Mountspace as a first layer: control entire subtrees
- ACL as a second layer: control at file-level

# Mountspaces

- Every process has a mountspace, inherited to childs
- Mountspace is represented as a directory
- Child mountspaces become child directories
- Changing a mountspace requires write permission
- Mountspace translates: `path` $\rightarrow$ (`FS, perm, subpath`)
- `perm` defines upperbound for files in `subpath`
- Can be done by unprivileged users
  - Filesystems and drivers run in userspace
  - ...with the user+group of the mounter
  - Overmounting system directories is no security issue

# Mounting for the User

## Tools

- mount creates a new FS for a device and makes it visible
  - $ mount /dev/hda1 /mnt /sbin/ext2
- bind makes an existing FS visible at a different place
  - $ bind /dev/ext2-hda1 /home/me/mnt

## What does bind do?

```
int fs = open("/dev/ext2-hda1", ...);
int ms = open("/sys/pid/self/ms", O_WRITE);
mount(ms, fs, "/home/me/mnt");
// open("/home/me/mnt/a/b", ...) -> FS_OPEN("/a/b")
```

# Sandbox

## Reasoning

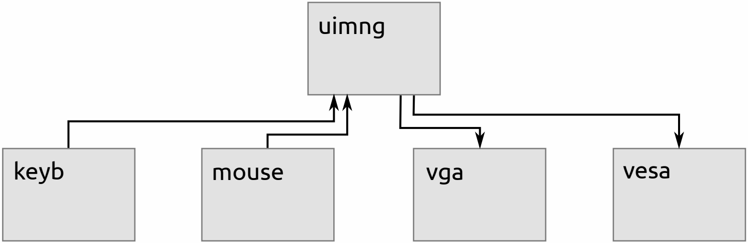
- Some applications are not trusted
- Running them as a different user is inconvenient
- Instead: run with same user, but less permissions

## The `sandbox` tool

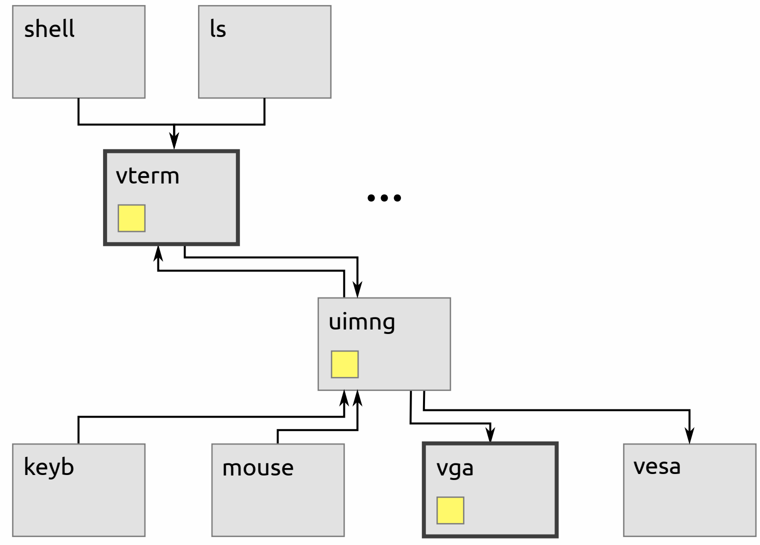
- Allows to leave groups
- Allows to reduce permissions to entire subtrees
- Example: `sandbox -g netuser -m /home:r app`
- Sandboxes can be nested and used by unprivileged users

# Outline

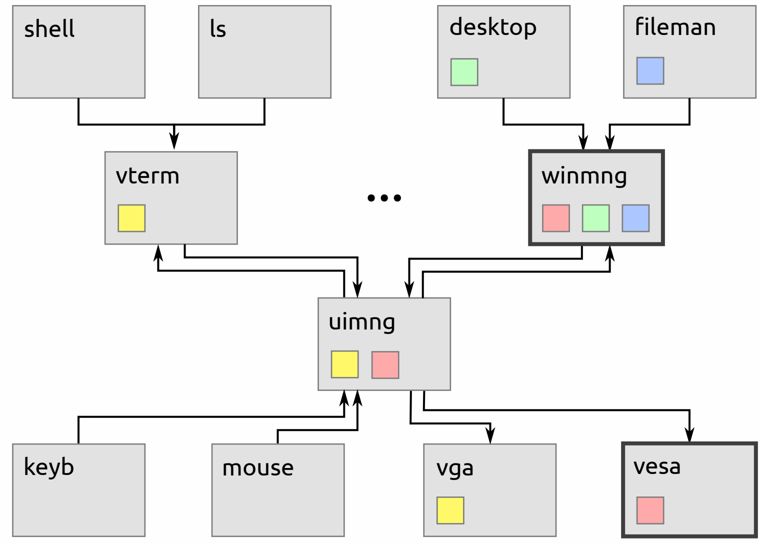
# UI Concept



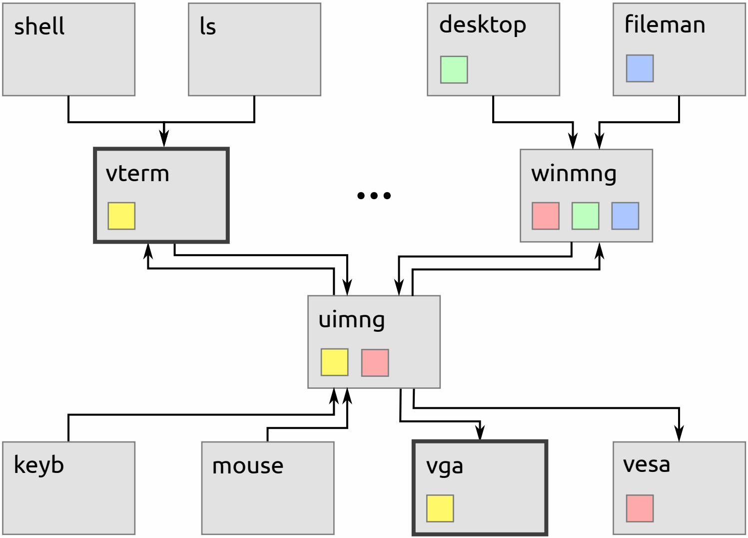
# UI Concept



# UI Concept


shell
ls
desktop
fileman
vterm
...
winmng
uimng
keyb
mouse
vga
vesa

# UI Concept


shell
ls
desktop
fileman
vterm
...
winmng
uimng
keyb
mouse
vga
vesa

# Questions

Get the code, ISO images, etc. on:

`https://github.com/Nils-TUD/Escape`

Questions?