Escape

Nils Asmussen

MKC, 07/09/2015
Outline

1. Introduction
2. Tasks
3. Memory
4. VFS
5. IPC
6. UI
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 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
- ...

- libc
- libc++
- libesc
- libgui
- libfs
- ...

μ-kernel
- Tasks
- Memory
- VFS

Hardware
Outline

1. Introduction
2. Tasks
3. Memory
4. VFS
5. IPC
6. UI
Processes and Threads

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

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

...
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
  - “User-semaphores” as a combination of atomic operations and process-local semaphores
  - Readers-writer-lock
  - ...

Priority Management

- Kernel adjusts thread priorities dynamically based on compute-intensity
- High CPU usage $\rightarrow$ downgrade, low CPU usage $\rightarrow$ upgrade
Outline

1. Introduction
2. Tasks
3. Memory
4. VFS
5. IPC
6. UI
Memory Management

Physical Memory
- Most of the memory is managed by a stack for fast alloc/free of 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

 libre.so (text)

MMIO

dynlink (text)

stack1

stack2

data

text

/lib/libc.so

flags=shared,exec
size=16K, procs=1,2

flags=write,grow,stack
size=12K, procs=2

flags=write,grow
size=16K, procs=1

flags=shared,exec
size=20K, procs=1,2

/bin/hello
Outline

1. Introduction
2. Tasks
3. Memory
4. VFS
5. IPC
6. UI
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. Unique names for synchronization
3. Access userspace filesystems
4. Access devices
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/fs/$fs`)
- The communication with devices works via asynchronous message passing
Message Passing

- **Client**
- **Channel**
  - inbox
  - outbox
- **Driver**

Messages are sent and received using the following functions:

- `send(id, msg)`
- `recv(id | 42, msg)`
- `recv(id | 42, msg)`
- `send(id | 42, msg)`

The channel stores messages in its inbox and outbox, with identifiers `id | 42` and `id | 43` respectively.
Devices Can Behave Like Files

- As in UNIX: Devices should be accessible like files
- Messages: FILE_OPEN, FILE_READ, FILE_WRITE, FILE_CLOSE
- Devices may support a subset of these messages
- If using open/read/write/close, the kernel handles the communication
- Transparent for apps whether it is a virtual file, file in userspace fs or device
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
- If using the system calls with the same names, the kernel handles the communication
- Filesystems are mounted using the `mount` system call
Mounting

**Concept**

- Mountspace is a special file
- Every process has a mountspace, inherited to childs
- Mountspace can be cloned and joined
- Mountspace contains list of \((\text{path},\text{fs-con})\) pairs
- Kernel translates fs-system-calls into messages to \textit{fs-con}
Mounting

Concept
- Mountspace is a special file
- Every process has a mountspace, inherited to childs
- Mountspace can be cloned and joined
- Mountspace contains list of \((\text{path}, \text{fs-con})\) pairs
- Kernel translates fs-system-calls into messages to \(\text{fs-con}\)

Example

// assuming that ext2 has created /dev/ext2-hda1
int fd = open("/dev/ext2-hda1", ...);
int ms = open("/sys/proc/self/ms", O_WRITE);
mount(ms, fd, "/mnt/hda1");
// open("/mnt/hda1/a/b", ...) -> FS_OPEN("/a/b")
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 just react on a specific message, do an `mmap` and check in `read/write` whether the shared memory should be used
Achieving Higher Throughput – Code Example

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

static char buf[SIZE];

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

close(fd);
```
Achieving Higher Throughput – Code Example

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

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

int fd = open("/dev/zero", IO_READ);
ulong shname;
void *buf;
if (sharebuf(fd, SIZE, &buf, &shname, 0) < 0) {
    if (buf == NULL)
        error("Unable to mmap buf");
}
while (read(fd, buf, SIZE) > 0) {
    // ...
}
destroybuf(buf, shname);
close(fd);
```
Achieving Higher Throughput – Usage Example

- cp
- ext2
- ata
Achieving Higher Throughput – Usage Example

- cp
- ext2
- ata
- ftpfs
- tcpip
- e1000
Canceling Operations

Problem

- What if we want to SIGTERM a process during a read?
- An already sent read-request can’t be taken back
- Channels might be shared (shared state ... )
Canceling Operations

**Problem**
- What if we want to SIGTERM a process during a read?
- An already sent read-request can’t be taken back
- Channels might be shared (shared state . . .)

**Solution**
- Introduce a cancel syscall and message
- If a thread gets a signal, it wakes up and sends the cancel message to the driver
- The driver cancels the currently pending request, if necessary
- Race-condition: the driver might have already responded
Sibling Channels

Problem

- Suppose, you want to have a control channel and event channel per client
- Suppose, you want to implement socket’s accept
- How do you do that?

Solution

Introduce a creatsibl syscall and message.

The kernel creates a new channel and sends a DEV CREATSIBL message to the driver over the current channel.

The driver knows both and can then e.g. attach client-specific state to the new channel based and associate it with the old one.
Sibling Channels

Problem

- Suppose, you want to have a control channel and event channel per client
- Suppose, you want to implement socket’s accept
- How do you do that?

Solution

- Introduce a creatsibl syscall and message
- The kernel creates a new channel and sends a DEV_CREATSIBL message to the driver over the current channel
- The driver knows both and can then e.g. attach client-specific state to the new channel based and associate it with the old one
Integrating Networking

- Network services should be accessible like files or filesystems
- To support URLs:
  "XYZ://foo/bar" is translated to "/dev/XYZ/foo/bar"
Integrating Networking

- Network services should be accessible like files or filesystems
- To support URLs:
  "XYZ://foo/bar" is translated to "/dev/XYZ/foo/bar"

Demo!
Outline

1. Introduction
2. Tasks
3. Memory
4. VFS
5. IPC
6. UI
IPC between Client and Driver (Low Level)

```c
int id = createdev("/dev/foo",...);
```

- **dev**
  - `createdev` creates a device node
  - `/dev/foo`

- **driver**
  - `creates`
IPC between Client and Driver (Low Level)

```c
int id = createdev("/dev/foo",...);

int fd = open("/dev/foo",IO_MSGS);

creates

points to

creates

inbox

client

channel

outbox

driver

client

channel

outbox

inbox

creates

dev

foo

int id = createdev("/dev/foo",...);

driver

creates

int fd = open("/dev/foo",IO_MSGS);
```
IPC between Client and Driver (Low Level)

```c
int id = createdev("/dev/foo", ...);

driver

int fd = open("/dev/foo", IO_MSGS);
points to
creates
inbox
client
channel
outbox
msg.arg1 = 10;
mid = send(fd, 42, &msg, sizeof(msg));
receive(fd, &mid, &msg, sizeof(msg));
```
IPC between Client and Driver (Low Level)

```c
int id = createdev("/dev/foo",...);

int fd = open("/dev/foo",IO_MSGS);
points to

int fd = getwork(id,&mid,&msg,sizeof(msg),0);
```

```
createdev("/dev/foo",...);

creates

int fd = open("/dev/foo",IO_MSGS);
points to

int fd = getwork(id,&mid,&msg,sizeof(msg),0);
```

Channel:
- **inbox**
- **outbox**

Client:
- `msg.arg1 = 10;`
- `mid = send(fd,42,&msg,sizeof(msg));`
- `receive(fd,&mid,&msg,sizeof(msg));`

Driver:
IPC between Client and Driver (Low Level)

```
int id = createdev("/dev/foo",...);

int fd = open("/dev/foo",IO_MSGS);
points to creates

int fd = getwork(id,&mid,&msg,sizeof(msg),0);
msg.arg1 = 1;
send(fd,mid,&msg,sizeof(msg));
```

```
driver
creates

int fd = open("/dev/foo",IO_MSGS);
points to inbox

msg.arg1 = 10;
mid = send(fd,42,&msg,sizeof(msg));
receive(fd,&mid,&msg,sizeof(msg));

int fd = getwork(id,&mid,&msg,sizeof(msg),0);
msg.arg1 = 1;
send(fd,mid,&msg,sizeof(msg));
```
Driver Example: /dev/zero

```c
struct ZeroDevice : public ClientDevice {
    explicit ZeroDevice(const char *name, mode_t mode) :
        ClientDevice(name, mode, DEV_TYPE_BLOCK, DEV_OPEN | DEV_SHFILE |
        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", 0444);
    dev.loop();
    return EXIT_SUCCESS;
}
```
Client Example: vterm

```cpp
// 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

1. Introduction
2. Tasks
3. Memory
4. VFS
5. IPC
6. UI
UI Concept

- uimng
- keyb
- mouse
- vga
- vesa
UI Concept

shell

ls

vterm

uimng

keyb

mouse

vga

vesa

vterm

...
UI Concept

- shell
- ls
- desktop
- fileman
- vterm
- mouse
- vga
- vesa
- uimng
- keyb
- winmng
- fileman
- desktop
- shell
- vterm
- vesa
- keyb
- mouse
- vga
UI Concept

- shell
- ls
- desktop
- fileman
- vterm
- winmng
- uimng
- keyb
- mouse
- vga
- vesa
Get the code, ISO images, etc. on: https://github.com/Nils-TUD/Escape

Any questions to Escape?