Creating an Operating System from Scratch

Koshan Dawlaty 3/7/26 SCALE 23x

OSDev wiki

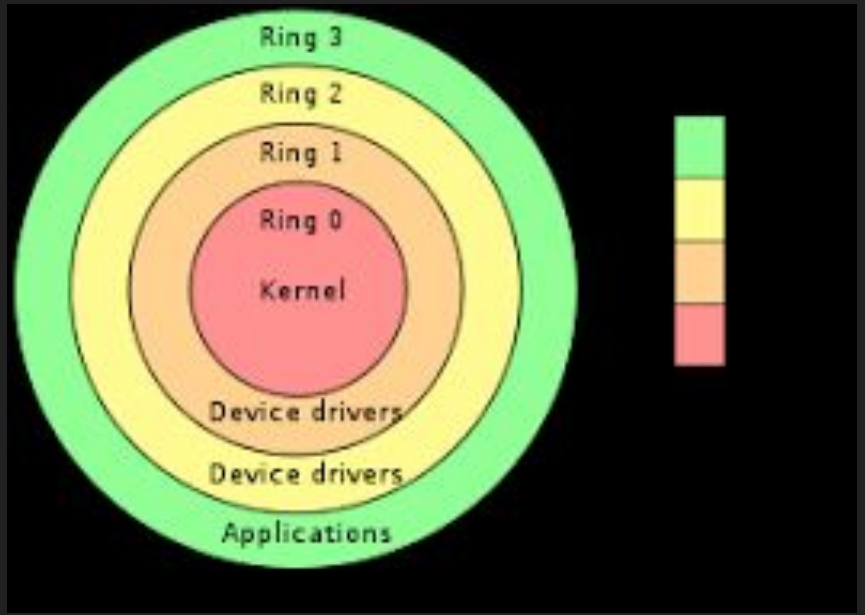
Has a lot of very useful information

https://wiki.osdev.org/

What is a kernel

Manages access to hardware

Userspace interacts with kernel through system calls



Bootloader - GRUB

Main purpose of the bootloader is to jump to 32 bit mode from 16 bit mode, and load the kernel code in to memory

Also provides pointers to some memory mapped devices

Printing to the screen

(char)(0xB8000) = ‘a’;

Write directly to a memory location to display a character

Debugging

Had to connect to gdb server to debug

Only works for one userspace executable (or two if they are loaded at different addresses)

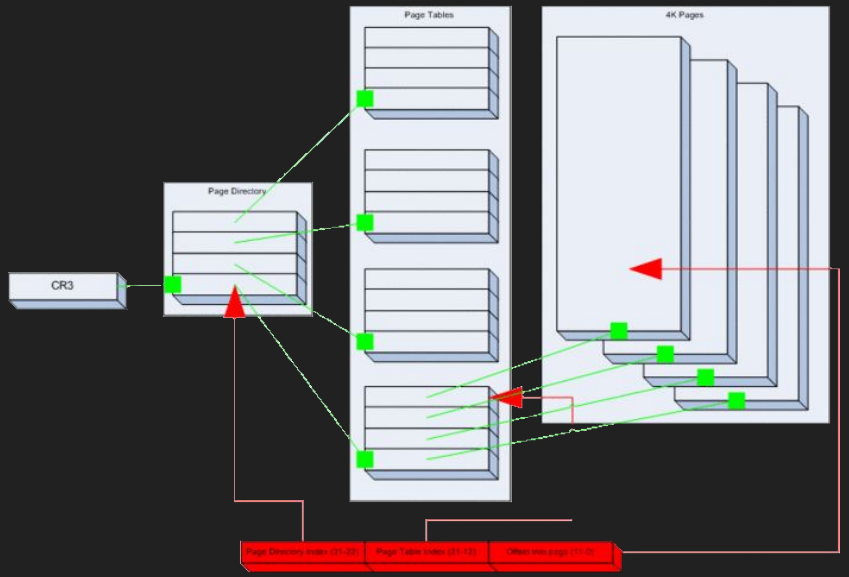
I recommend doing this very early on

Paging

Paging is the primary method for memory management on x86

Address space is split into tables, then pages, with directories containing page tables

Changing page directories will change what is mapped



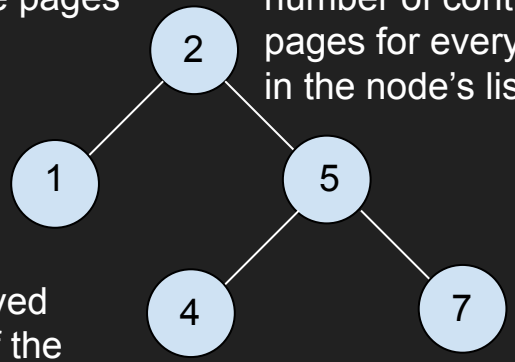
Binary tree virtual page allocation

Faster

More difficult to initialize

Each node contains a list of free pages

The node's key is the number of contiguous pages for every element in the node's list



A page is removed from the start of the node's list on allocation

Interrupts

Interrupts stop the currently executing code and run other code

Fault handlers

ATA PIO mode

```
for (size_t i = 0; i < …; i ++)
{
        wait_for_data(...);
        buffer[i] = inb(...);
}
```

Very slow

Bug

Incorrect

```
outb(lba_low, (u8)lba & 0xFF);
outb(lba_mid, (u8)(lba >> 8) & 0xFF);
outb(lba_high (u8)(lba >> 16) & 0xFF);
```

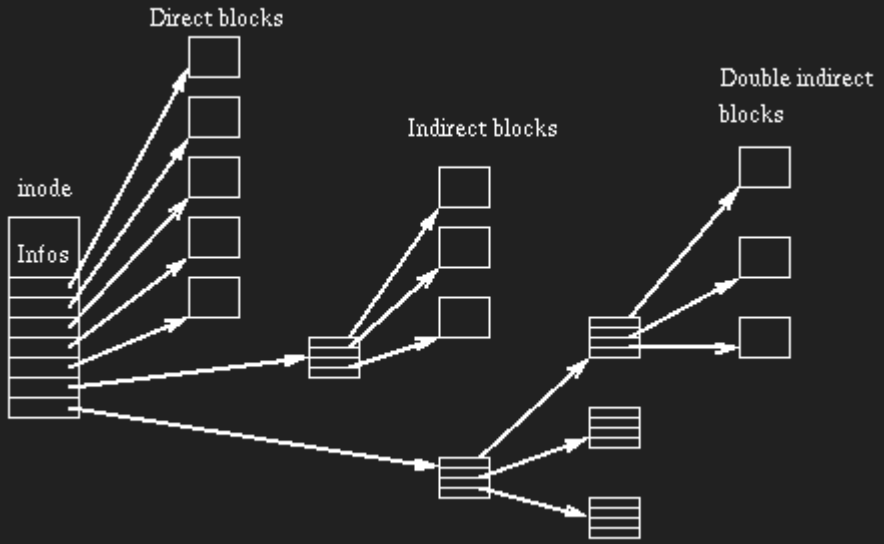
Correct

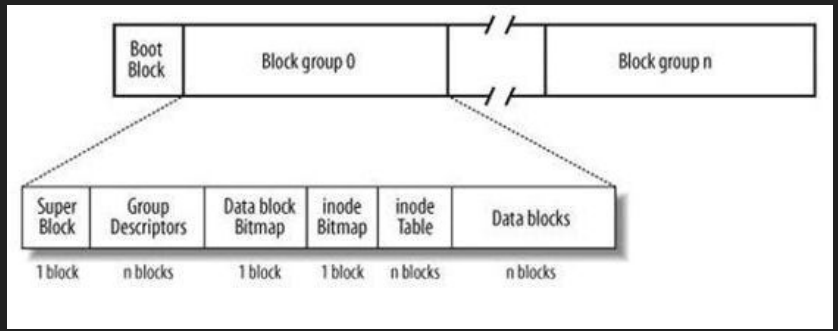
```
outb(lba_low, (u8)(lba & 0xFF));
outb(lba_mid, (u8)((lba >> 8) & 0xFF));
outb(lba_high, (u8)((lba >> 16) & 0xFF));
```

Filesystems

Ext2

Filesystem abstraction layer





File system abstraction

```
struct inode_functions {
        int (*read)(struct inode *node …)
        …
}
```

```
struct superblock {
        struct filesystem_type type;
}
```

```
superblock->type.inode_functions.read()
```

```
const struct filesystem_type ext2_filesystem = {
        .inode_functions = { … }
        …
}
```

readdir()

Old:

struct dentry ** get_dentries(struct dentry * dir);

Returns a heap pointer of an array of all of the directory entries

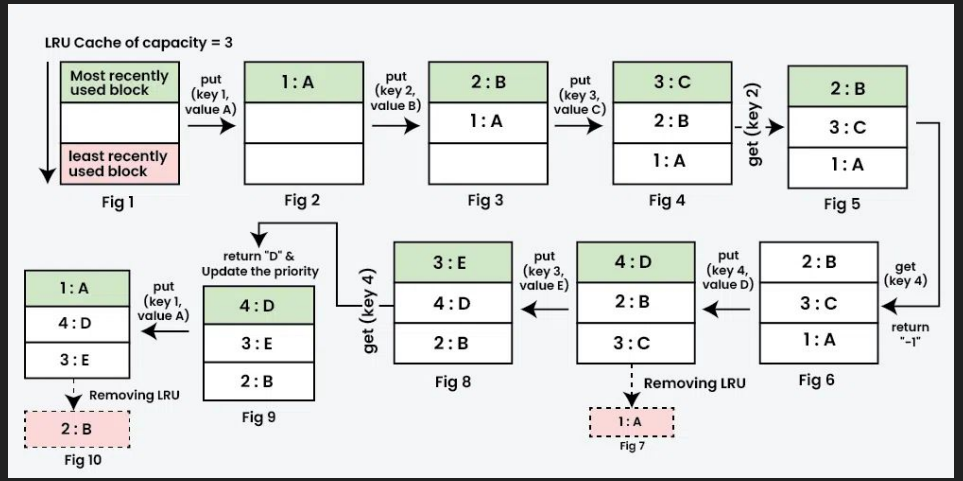
Current:
struct dentry * readdir(struct dentry * dir, struct dir_context context);

Returns one, then reads the next one on the next call
A filesystem specific offset is stored to further improve performance

Directory entry cache

Stores recently used directory entries

Permanently stores mount points and devices



Issue with cache

The cache can contain some of the entries for a directory, but all of the entries are contained in the filesystem

This means it is difficult to read from the cache when trying to read all entries in a directory

Initial Userspace

```
void user_function() { syscall(123, 456); }
```

(Declared in kernel)

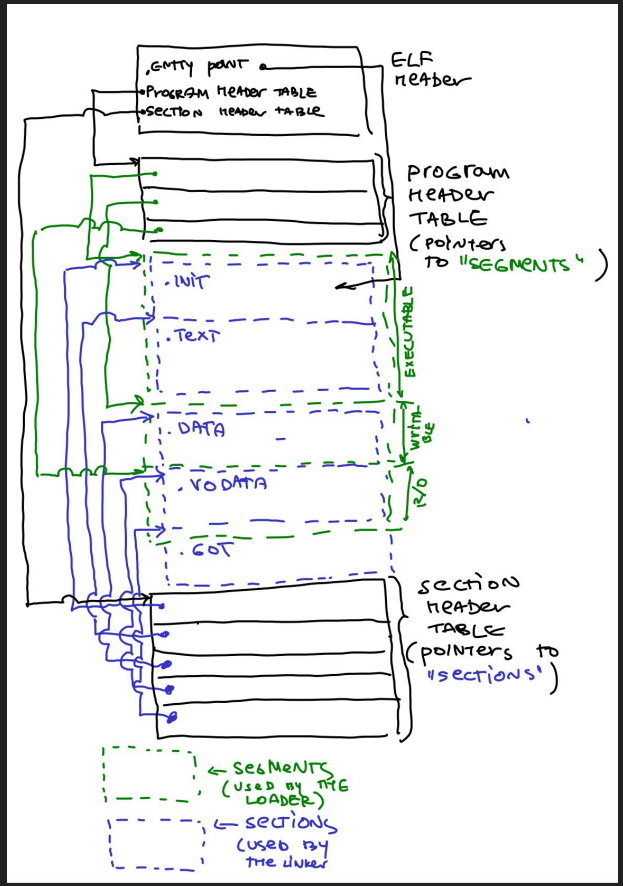
```
memcpy(user_addr, user_function, 100);

jump_to_usermode(user_addr);
```

Executables

ELF file format

Easy to compile to, and simple

Important for an operating system



System calls

How userspace interacts with the kernel

Implemented with a software interrupt

Scheduler

Called when a timer interrupt occurs

Circular linked list for queue of tasks

Results in a lot of race conditions if you had not prepared the kernel for multitasking

Context Switch - Incorrect implementation

```
struct task * current_task = …;

if (current_task->was_in_usermode) {
        jump_to_usermode(current_task);
} else {
        jump_to_kernel(current_task);
}
```

Context switch - Correct implementation

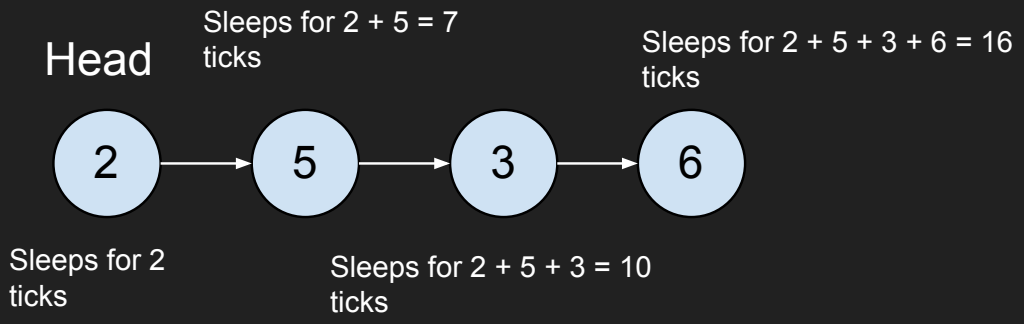
```
save_registers(old_task);

swap_tasks();

struct task * current_task = …;

load_registers(current_task->registers);
```

```
load_registers:
        # load the registers from the struct
        …
        mov task_esp, %esp
        ret
save_registers:
        # save the registers
        …
        mov %esp, task_esp
        ret
```

Sleeping processes

Processes are added to a sleeping queue

Only decrement the head of the queue



x86-64

Very similar to 32 bit x86

I was able to reuse the drivers for disks and the keyboard

Only had to change paging, interrupts, and scheduler assembly functions

x86-64 bug

```
struct tss
{
u64 rsp0;
(padding added by compiler)
u64 rsp1;
…
};
```

```
struct tss
{
u64 rsp0;
u64 rsp1;
…
} __attribute__((packed));
```

Graphics

Can be very difficult

GRUB simplifies the process

I wanted to implement unix domain sockets and shared memory for the window manager

Window manager

Needed to implement IPC for it to work

Wanted a server socket that would accept connections from clients, then map the client’s buffer in the server address space

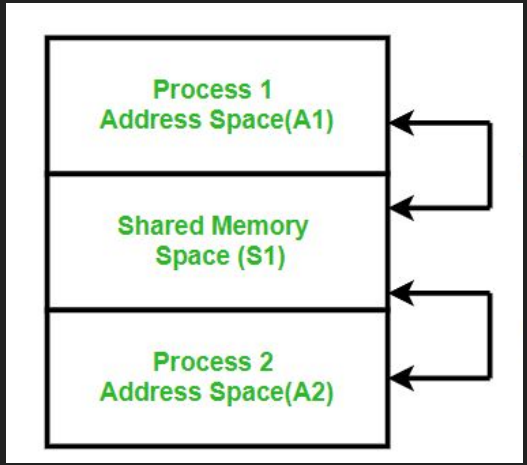
Device abstraction

Keyboard and mouse became accessed through device files

Like /dev on linux

Shared memory

Wanted to use shared memory to map the window contents into the window manager process

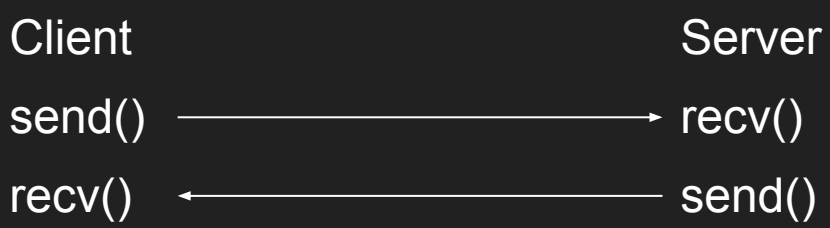


Unix domain sockets

Very similar to tcp sockets (they share the same (send/recv) api)

Only runs on local machine

Used by real display servers



Socket temporary fix

```
int connection_socket = socket_create();

int client_socket;

send(connection_socket, &client_socket, sizeof(int));
```

Socket bug

Incorrect

```
send(client, &render_code, sizeof(int));

send(client, &key_request_code,
sizeof(int));

recv(client, &key, 1);
```

Correct

```
send(client, &render_code, sizeof(int));

recv(client, &acknowledge_code,
sizeof(int));

send(client, &key_request_code,
sizeof(int));

recv(client, &key, 1);
```

User threads

Very useful when implementing window manager

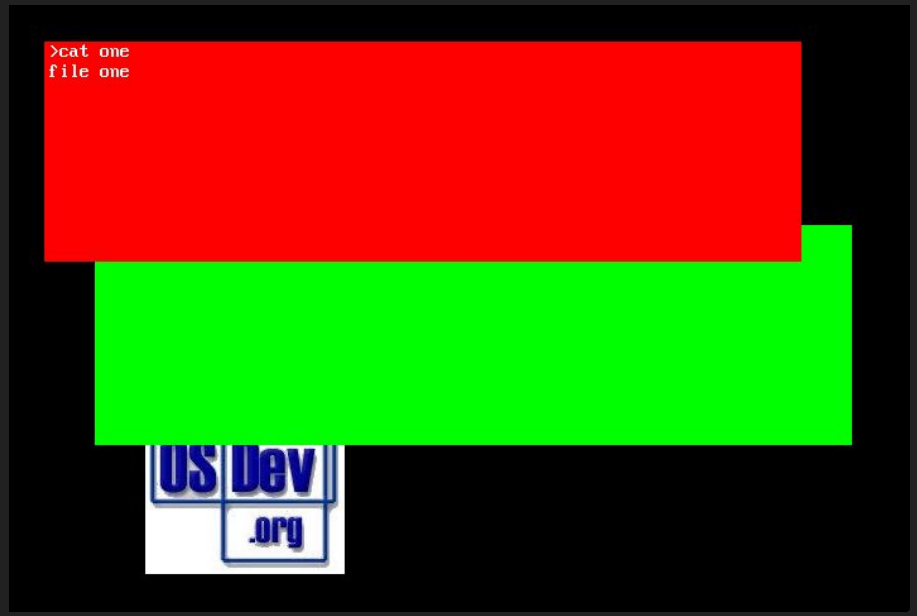
My socket implementation would block, which would make taking information from other clients impossible

I could have also fixed this by making more asynchronous functions

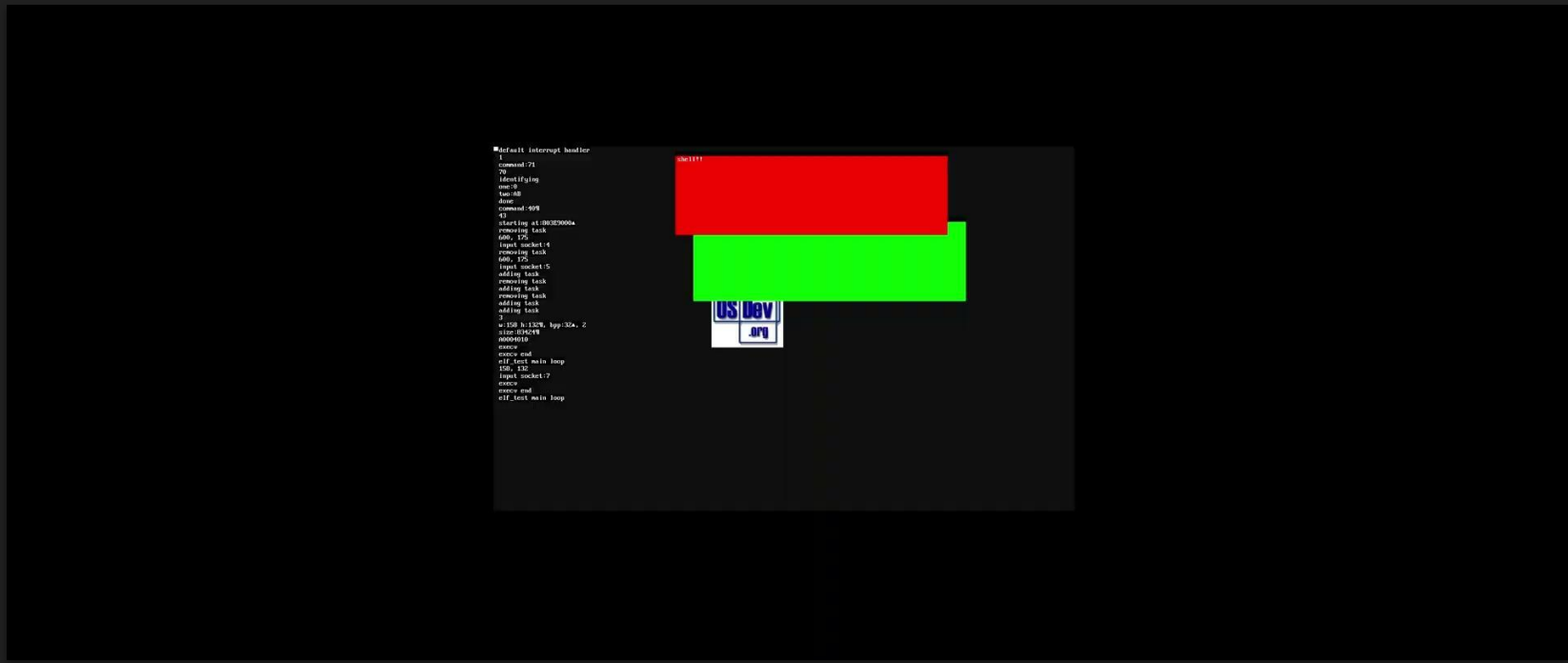
Window compositing

Stores windows in a stack

Goes through stack and renders relevant part of windows



Window compositing video



Terminal

pipe() syscall

Child processes (like the ones created from the shell) need to inherit file descriptors

Other things

Unit test framework

In-memory block device

Running on real hardware

Thank You!

Q&A

Github

https://github.com/koshan-dawlaty/scale-os-public