Linux Kernel Module Driver for Keyboard LEDs

Bridging the Gap Between the Kernel Space and the User Space

Connor Reeder

Problem Statement:

The driver which Linux currently uses to activate and deactivate the Caps Lock and Num Lock LED lights on a Toshiba Satellite C55-A5286 currently does not allow for control from user space applications. The functionality of those two lights is bound to the standard functions of Caps Lock mode and Num Lock mode, respectively. Thus, there is no way to repurpose the lights to serve other functions in the event that the user does not use those lights for their current function.

Idea

- Write a Linux kernel module which containing a driver for the Toshiba Satellite C55-A5286 keyboard LED lights which will replace the one currently in use.
- The driver will mount each of the two LED lights as a linux special file node in the /dev directory so as to allow any user space application to read and write to it like any other device.
- It will be mounted as a character device, thus requiring applications to read and write to it in block-aligned sizes.
- Create a simple user space program which will use the caps lock light as a notification for some type of event.

Issues and Challenges

- Lack of documentation for particular C55-A5286 hardware.
- Most feasible way to control lights ended up being using LED subsystem
- Creating and accessing user space files safely from the kernel space
- Had to maintain multiple device files from the same driver, meaning that the two lights share the same code and variables.

Intel Transactional Synchronization Extensions

Robert Guthrie

An Introduction to Transactional Memory

Problem: Determining possible resource contention statically results in unnecessary loss of concurrency dynamically.

```
int arr[4096];
int get(int i) {
    lock();
    int ret = arr[i];
    unlock();
    return ret;
}
```

Thread A:
get(a);

Thread B:
get(b);

Result?

An Introduction to Transactional Memory

Do Thread A and Thread B need to synchronize? Sometimes.

If a != b, there is no contention between Thread A and Thread B. But this can only be determined dynamically.

Transactional Memory is a way to dynamically determine if serialization on a lock is necessary.

If unnecessary (like when a != b), threads can continue with the critical section without acquiring a lock.

An Introduction to Transactional Memory

What happens when there is a **data conflict** between threads in their **transactional regions**?

The threads might have already executed instructions in their critical regions before the conflict is determined.

An Introduction to Transactional Memory

Solution: While executing in transactional regions, memory writes are stored in a local state invisible to other threads of execution.

Upon reaching the end of a transactional region, if no data conflict was detected, the entire sequence of memory writes from the transactional region is committed to main memory and visible to the rest of the threads.

If conflict is detected during execution, the processor state reverts to just before entering the region, and execution continues serially.

Intel Transactional Synchronization Extensions

An API for using Transactional Memory in concurrent applications on Intel Skylake (the most recent generation) and later

Brand new technology receiving a lot of current research

Provides a backwards-compatible Hardware Lock Elision (HLE) interface and more powerful Restricted Transactional Memory (RTM) interface

Example RTM Code

New instructions

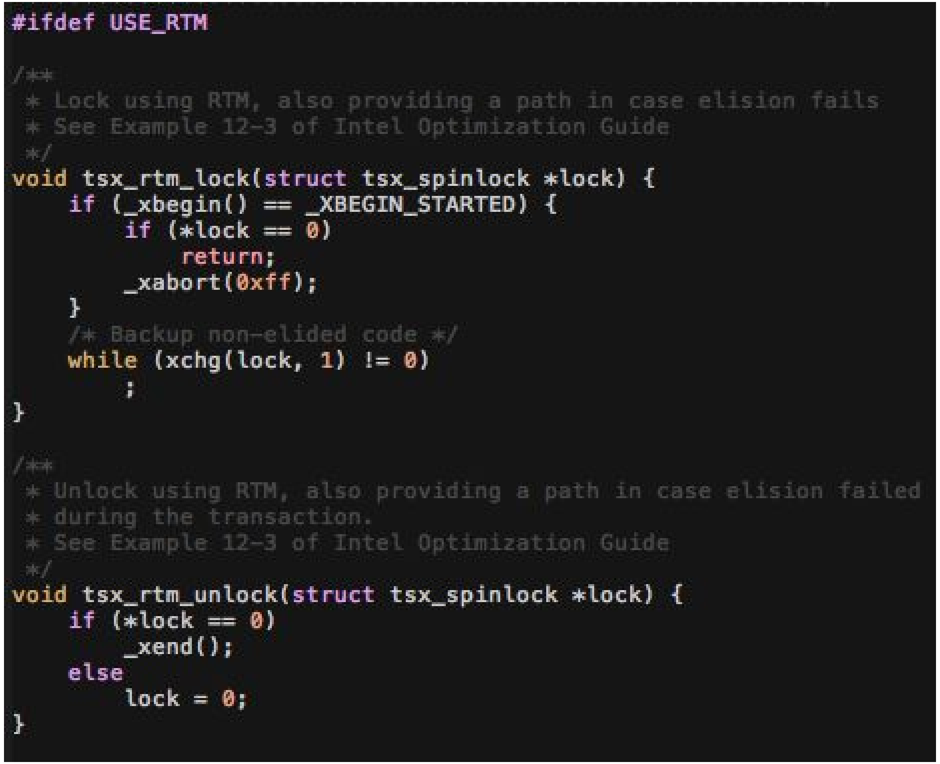
XBEGIN alt_path_addr

movl (%ebx), %eax

addl $1, %eax

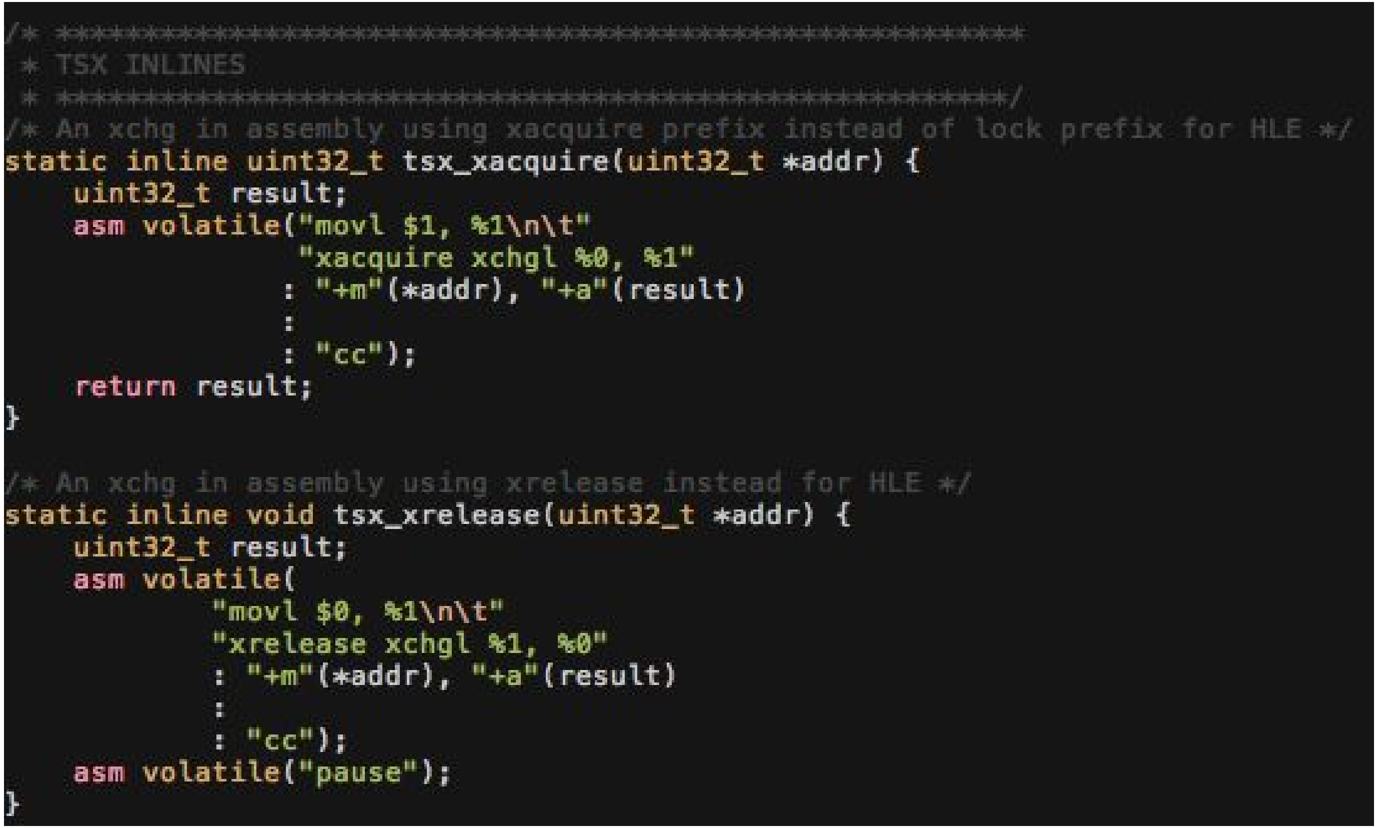
movl %eax, (%ebx)

XEND



```
#ifdef USE_RTM

/**
 * Lock using RTM, also providing a path in case elision fails
 * See Example 12-3 of Intel Optimization Guide
 */
void tsx_rtm_lock(struct tsx_spinlock *lock) {
    if (_xbegin() == _XBEGIN_STARTED) {
        if (*lock == 0)
            return;
        _xabort(0xff);
    }
    /* Backup non-elided code */
    while (xchg(lock, 1) != 0)
        ;
}

/**
 * Unlock using RTM, also providing a path in case elision failed
 * during the transaction.
 * See Example 12-3 of Intel Optimization Guide
 */
void tsx_rtm_unlock(struct tsx_spinlock *lock) {
    if (*lock == 0)
        _xend();
    else
        lock = 0;
}
```

Example HLE Code



```
/* ********************************************************
 * TSX INLINES
 * *******************************************************/
/* An xchg in assembly using xacquire prefix instead of lock prefix for HLE */
static inline uint32_t tsx_xacquire(uint32_t *addr) {
    uint32_t result;
    asm volatile("movl $1, %1\n\t"
                 "xacquire xchgl %0, %1"
                : "+m"(*addr), "+a"(result)
                :
                : "cc");
    return result;
}

/* An xchg in assembly using xrelease instead for HLE */
static inline void tsx_xrelease(uint32_t *addr) {
    uint32_t result;
    asm volatile(
            "movl $0, %1\n\t"
            "xrelease xchgl %1, %0"
            : "+m"(*addr), "+a"(result)
            :
            : "cc");
    asm volatile("pause");
}
```

TSX in JOS

Wanted to do optimization and benchmarking: not possible without booting

What I did:

- Finer-grained locking
- TSX protection of random access structures (envs array, pages, etc.)

Possible Optimizations:

- Allocate environment ID's so that the envs structs are spread out as much as possible (better: store in a hash map based on the environment ID)

TSX in User Programs

Wrote a hash map in C++, similar to tutorial 8

Demo

Design decisions using TSX

PAGING TO DISK

Sneh Munshi

Bhavani Jaladanki

Problem Statement

- **Problem:** Memory space in JOS is limited!

- **Solution:**
  - *Page Swapping:*
    - Program will be able to use any page it wants regardless of whether the page is already in memory or not.
    - Swapping should replace a page in memory that will not be used in the near future, with the page in the disk that the program wants.

- **Goal:** make sure that the page switched out is one that is rarely used since permanent storage is slower than memory

Paging Server (disk)

- Uses in-memory bitmap – show which blocks are used in the partition of paged out pages
- Similar to File System server
- 4 IPCs – handled constantly in loop
  - Page in
  - Page out
  - Discard Page
  - Get Page Stats

Paging Library

- Uses a type of LRU to find page to swap out
- Page Map – Used w/ Page in
- Page Un-map – Used w/ Page out
- Page Allocation
- Page Fault handler

Demo Time!

- Tries to allocate more memory than the amount of physical memory that system actually has, so paging out

- Tries to get pages that system allocated previously, so paging in

- Will breakpoint in code w/o paging, & work in code w/ paging

Test #1: Normal Paging

- Goes from va 0x10000000 to 0x14000000, and allocates the pages.
- Stores a number in sequence from 1, in each page
- Goes in loop and checks that each page has the right number that represents the page number (linear)

Test #2: Random Paging

- Goes from va 0x10000000 to 0x18000000, and allocates the pages.
- Stores a number in sequence from 1, in each page
- Randomly checks that each page has the right number that represents the page number

Test #3: Page Eviction

- Goes from va 0x10000000 to 0x18000000, and allocates the pages.

- Stores a number in sequence from 1, in each page

- Goes through pages from second half of memory, proving that LRU is a good algorithm to use

Efficiency
LRU vs Linear

Page ins/Page Outs	Normal Paging	Random Paging	Page Eviction
Linear	20317/25369 = .80	5302/22573 = .23	5989/23272 = .68
LRU	6477/11300 = .57	5256/22469 = .23	13963/31284 = .45

LRU generally produces a lower page in/page out, especially with big programs like Normal Paging, which have many page ins to page outs

Conclusion

- Used exo-kernel style to give user programs permission to have their own paging server and library for paging in and out

- We made sure the ratio of page ins to page outs was not very high

- Ensures that number of disk access are as low as possible

Thank you!

WE HOPE YOU ENJOYED IT!

Extending JOS by Implementing mmap( )

Daniel Carnauba, Nicolette Fink, Thomas Coe

Problem Statement

- read() and write() can be inefficient for non-sequential file access
    - Lots of system call overhead (seeking)
- Multiple processes accessing the same file can be inefficient
    - Each process reads the file into a buffer in its individual memory space
- Context switching can be costly when making many system calls for reading or writing files

Implementation: mmap( ), munmap( ), and msync()

- **mmap**() creates a new mapping in the virtual address space of the calling process.

- The **munmap**() system call deletes the mappings for the specified address range, and causes further references to addresses within the range to generate invalid memory references.

- **msync**() flushes changes made to a file that was mapped into memory, ensuring that changes are written back before munmap() is called.

Implementation: Mapping Modes

MAP_SHARED

Share this mapping. Updates to the mapping are visible to other processes that map this file, and are carried through to the underlying file.

MAP_PRIVATE

Create a private copy-on-write mapping. Updates to the mapping are not visible to other processes mapping the same file, and are not carried through to the underlying file. It is unspecified whether changes made to the file after the **mmap**() call are visible in the mapped region.

Demo

- MAP_PRIVATE and MAP_SHARED functionality
  - Read and write to files opened with mmap()
  - Access MAP_SHARED files from multiple processes
  - Cause page faults by accessing unmapped files

- Benchmark mmap() vs. read()
  - Compare sequential and non-sequential accesses

Demo

- Sample results of benchmarking test:

	Sequential	Random
mmap()	10 ms	<10 ms
read()	1850 ms	2050 ms

Questions?

Paging to Disk

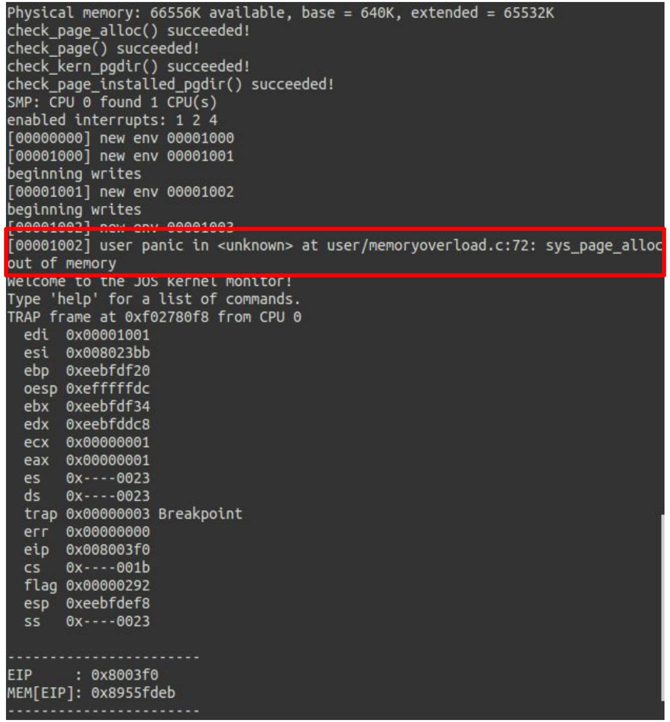
Henry Peteet, Millad Asgharneya, Premkumar Saravanan

Problem statement revisited

Our configuration of JOS has 64M of physical memory.

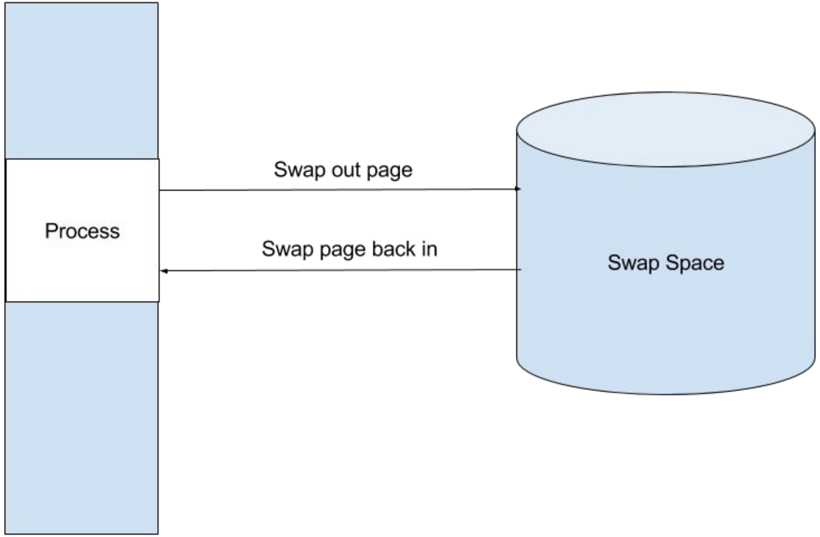
If you use more than 64M the OS will kill the environment.

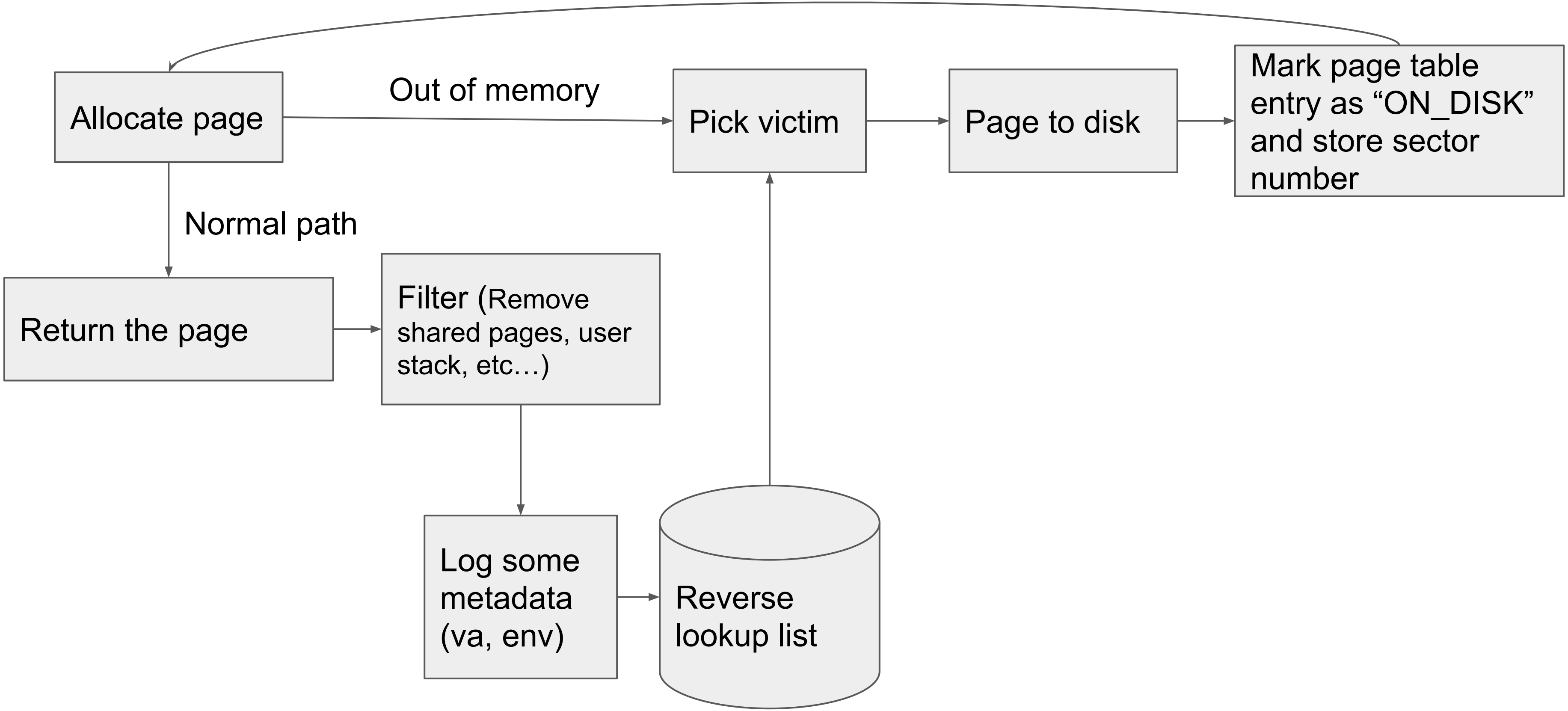
We added a panic just to highlight the issue.



```
Physical memory: 66556K available, base = 640K, extended = 65532K
check_page_alloc() succeeded!
check_page() succeeded!
check_kern_pgdir() succeeded!
check_page_installed_pgdir() succeeded!
SMP: CPU 0 found 1 CPU(s)
enabled interrupts: 1 2 4
[00000000] new env 00001000
[00001000] new env 00001001
beginning writes
[00001001] new env 00001002
beginning writes
[00001002] new env 00001003
[00001002] user panic in <unknown> at user/memoryoverload.c:72: sys_page_alloc
out of memory
Welcome to the JOS kernel monitor!
Type 'help' for a list of commands.
TRAP frame at 0xf02780f8 from CPU 0
  edi  0x00001001
  esi  0x008023bb
  ebp  0xeebfdf20
  oesp 0xefffffdc
  ebx  0xeebfdf34
  edx  0xeebfddc8
  ecx  0x00000001
  eax  0x00000001
  es   0x----0023
  ds   0x----0023
  trap 0x00000003 Breakpoint
  err  0x00000000
  eip  0x008003f0
  cs   0x----001b
  flag 0x00000292
  esp  0xeebfdef8
  ss   0x----0023

----------------------
EIP     : 0x8003f0
MEM[EIP]: 0x8955fdeb
----------------------
```

How did we address it?





Allocate page

Out of memory

Pick victim

Page to disk

Mark page table entry as “ON_DISK” and store sector number

Normal path

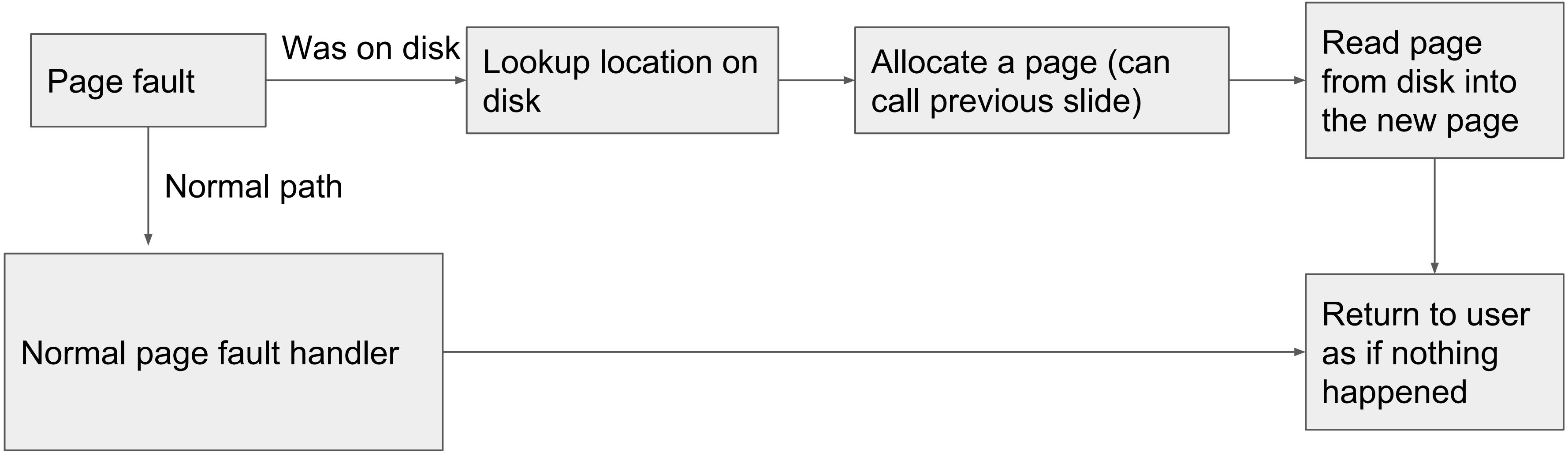
Return the page

Filter (Remove shared pages, user stack, etc…)

Log some metadata (va, env)

Reverse lookup list

Implementation



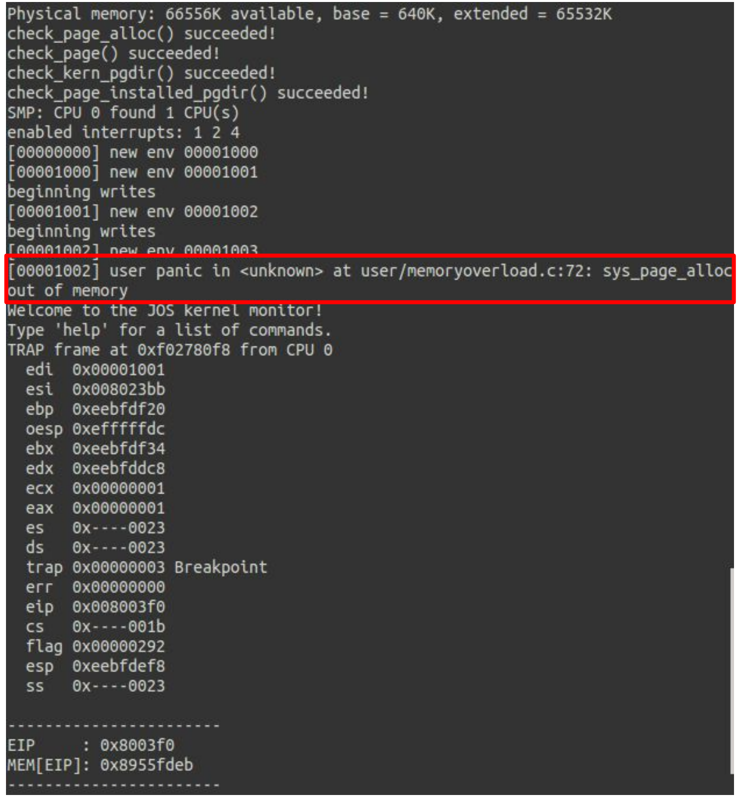
Testing

We wrote a test program that mimics dumbfork and writes/reads a bunch of memory guaranteeing that all environments stay active the entire time.

With this we were able to break the old version of JOS, and see a successful run on our modified version when we try to use 64M of memory (since the kernel uses some of it as well)

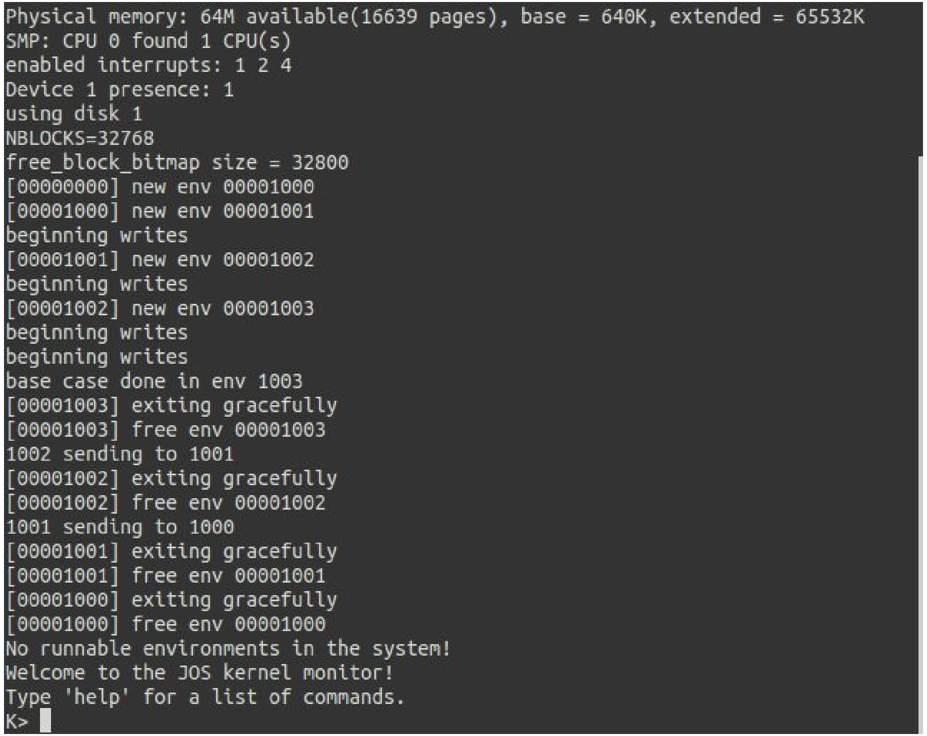
Testing results

Original



```
Physical memory: 66556K available, base = 640K, extended = 65532K
check_page_alloc() succeeded!
check_page() succeeded!
check_kern_pgdir() succeeded!
check_page_installed_pgdir() succeeded!
SMP: CPU 0 found 1 CPU(s)
enabled interrupts: 1 2 4
[00000000] new env 00001000
[00001000] new env 00001001
beginning writes
[00001001] new env 00001002
beginning writes
[00001002] new env 00001003
[00001002] user panic in <unknown> at user/memoryoverload.c:72: sys_page_alloc
out of memory
Welcome to the JOS kernel monitor!
Type 'help' for a list of commands.
TRAP frame at 0xf02780f8 from CPU 0
  edi  0x00001001
  esi  0x008023bb
  ebp  0xeebfdf20
  oesp 0xeffffdc
  ebx  0xeebfdf34
  edx  0xeebfddc8
  ecx  0x00000001
  eax  0x00000001
  es   0x----0023
  ds   0x----0023
  trap 0x00000003 Breakpoint
  err  0x00000000
  eip  0x008003f0
  cs   0x----001b
  flag 0x00000292
  esp  0xeebfdef8
  ss   0x----0023

----------------------
EIP     : 0x8003f0
MEM[EIP]: 0x8955fdeb
----------------------
```

Ours



```
Physical memory: 64M available(16639 pages), base = 640K, extended = 65532K
SMP: CPU 0 found 1 CPU(s)
enabled interrupts: 1 2 4
Device 1 presence: 1
using disk 1
NBLOCKS=32768
free_block_bitmap size = 32800
[00000000] new env 00001000
[00001000] new env 00001001
beginning writes
[00001001] new env 00001002
beginning writes
[00001002] new env 00001003
beginning writes
beginning writes
base case done in env 1003
[00001003] exiting gracefully
[00001003] free env 00001003
1002 sending to 1001
[00001002] exiting gracefully
[00001002] free env 00001002
1001 sending to 1000
[00001001] exiting gracefully
[00001001] free env 00001001
[00001000] exiting gracefully
[00001000] free env 00001000
No runnable environments in the system!
Welcome to the JOS kernel monitor!
Type 'help' for a list of commands.
K> 
```

Results

1. We can run environments that use more than 64M
2. We can evict from other environments (so if we can start new environments even when memory is full)
3. We use shared swap space

Limitations

1. Disk space
2. FIFO limitations
    a. Commonly pages out pages that are used heavily
    b. Can easily lead to more and more page faults
    c. With more processes we can have more and more faults since working sets take up more of memory

Questions?

Intel 80386 Emulator

•••

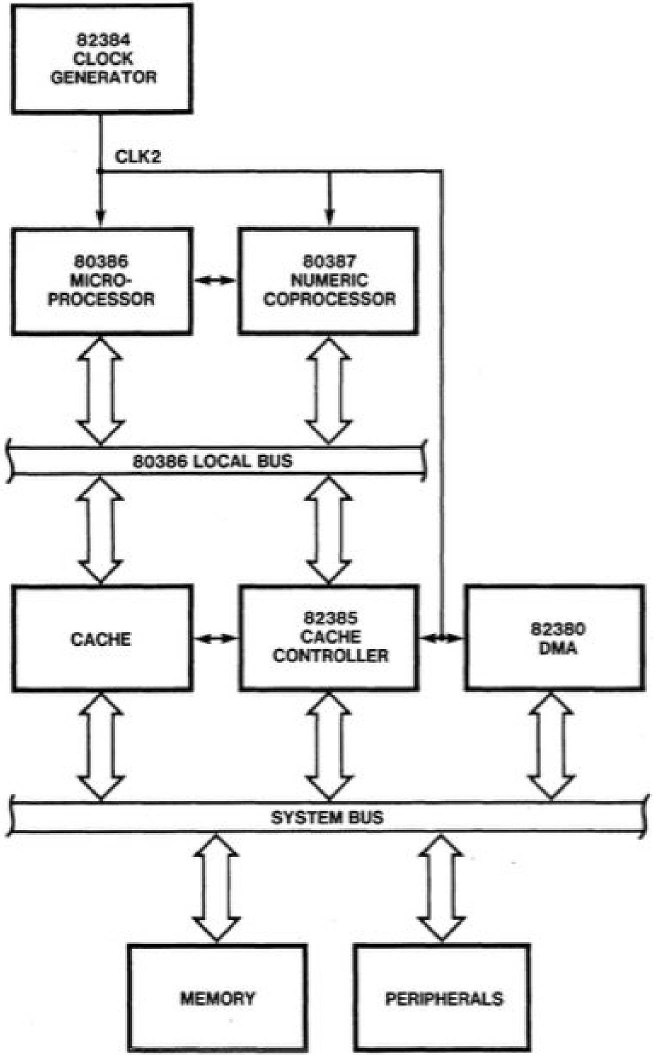
Stephan Williams

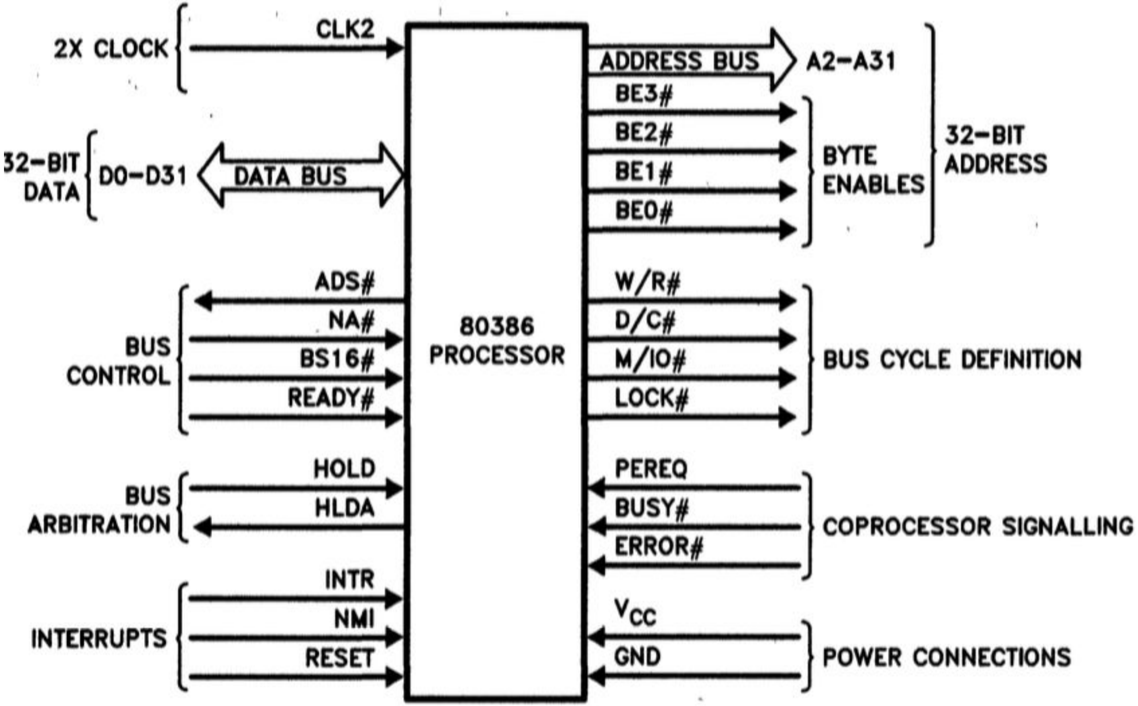
Features

- Written in Rust
- Trivial BIOS
- Disk I/O over bus (READ SECTOR)
- RAM over bus
- Text Display
- Virtual Memory
- >200 opcodes
- Can boot JOS

Things Lab 1 JOS can live without:

- Protected-mode segments
- Memory Protection
- Interrupts


82384 CLOCK GENERATOR
CLK2
80386 MICRO-PROCESSOR
80387 NUMERIC COPROCESSOR
80386 LOCAL BUS
CACHE
82385 CACHE CONTROLLER
82380 DMA
SYSTEM BUS
MEMORY
PERIPHERALS


2X CLOCK
CLK2
32-BIT DATA
D0-D31
DATA BUS
BUS CONTROL
ADS#
NA#
BS16#
READY#
BUS ARBITRATION
HOLD
HLDA
INTERRUPTS
INTR
NMI
RESET
80386 PROCESSOR
ADDRESS BUS
A2-A31
BE3#
BE2#
BE1#
BE0#
BYTE ENABLES
32-BIT ADDRESS
W/R#
D/C#
M/IO#
LOCK#
BUS CYCLE DEFINITION
PEREQ
BUSY#
ERROR#
COPROCESSOR SIGNALLING
V_{CC}
GND
POWER CONNECTIONS

INSTRUCTION PREFIX	ADDRESS-SIZE PREFIX	OPERAND-SIZE PREFIX	SEGMENT OVERRIDE
0 OR 1	0 OR 1	0 OR 1	0 OR 1
NUMBER OF BYTES			

OPCODE	MODR/M	SIB	DISPLACEMENT	IMMEDIATE
1 OR 2	0 OR 1	0 OR 1	0,1,2 OR 4	0,1,2 OR 4
NUMBER OF BYTES				

MODR/M BYTE

7	6	5	4	3	2	1	0
MOD		REG/OPCODE			R/M		

SIB (SCALE INDEX BASE) BYTE

7	6	5	4	3	2	1	0
SS		INDEX			BASE		

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	ADD						PUSH	POP	OR						PUSH	2-byte
	Eb,Gb	Ev,Gv	Gb,Eb	Gv,Ev	AL,Ib	eAX,Iv	ES	ES	Eb,Gb	Ev,Gv	Gb,Eb	Gv,Ev	AL,Ib	eAX,Iv	CS	escape
1	ADC						PUSH	POP	SBB						PUSH	POP
	Eb,Gb	Ev,Gv	Gb,Eb	Gv,Ev	AL,Ib	eAX,Iv	SS	SS	Eb,Gb	Ev,Gv	Gb,Eb	Gv,Ev	AL,Ib	eAX,Iv	DS	DS
2	AND						SEG	DAA	SUB						SEG	DAS
	Eb,Gb	Ev,Gv	Gb,Eb	Gv,Ev	AL,Ib	eAX,Iv	=ES		Eb,Gb	Ev,Gv	Gb,Eb	Gv,Ev	AL,Ib	eAX,Iv	=CS	
3	XOR						SEG	AAA	CMP						SEG	AAS
	Eb,Gb	Ev,Gv	Gb,Eb	Gv,Ev	AL,Ib	eAX,Iv	=SS		Eb,Gb	Ev,Gv	Gb,Eb	Gv,Ev	AL,Ib	eAX,Iv	=CS	
4	INC general register								DEC general register							
	eAX	eCX	eDX	eBX	eSP	eBP	eSI	eDI	eAX	eCX	eDX	eBX	eSP	eBP	eSI	eDI
5	PUSH general register								POP into general register							
	eAX	eCX	eDX	eBX	eSP	eBP	eSI	eDI	eAX	eCX	eDX	eBX	eSP	eBP	eSI	eDI
6	PUSHA	POPA	BOUND	ARPL	SEG	SEG	Operand	Address	PUSH	IMUL	PUSH	IMUL	INSB	INSW/D	OUTSB	OUTSW/D
			Gv,Ma	Ew,Rw	=FS	=GS	Size	Size	Ib	GvEvIv	Ib	GvEvIv	Yb,DX	Yb,DX	Dx,Xb	DX,Xv
7	Short displacement jump of condition (Jb)								Short-displacement jump on condition(Jb)							
	JO	JNO	JB	JNB	JZ	JNZ	JBE	JNBE	JS	JNS	JP	JNP	JL	JNL	JLE	JNLE
8	Immediate Grpl			Grpl	TEST		XCNG		MOV				MOV	LEA	MOV	POP
	Eb,Ib	Ev,Iv		Ev,Iv	Eb,Gb	Ev,Gv	Eb,Gb	Ev,Gv	Eb,Gb	Ev,Gv	Gb,Eb	Gv,Ev	Ew,Sw	Gv,M	Sw,Ew	Ev
9	NOP	XCHG word or double-word register with eAX							CBW	CWD	CALL	WAIT	PUSHF	POPF	SAHF	LAHF
		eCX	eDX	eBX	eSP	eBP	eSI	eDI			Ap		Fv	Fv		
A	MOV				MOVSB	MOVSW/D	CMPSB	CMPSW/D	TEST		STOSB	STOSW/D	LODSB	LODSW/D	SCASB	SCASW/D
	AL,Ob	eAX,Ov	Ob,AL	Ov,eAX	Xb,Yb	Xv,Yv	Xb,Yb	Xv,Yv	AL,Ib	eAX,Iv	Yb,AL	Yv,eAX	AL,Xb	eAX,Xv	AL,Xb	eAX,Xv
B	MOV immediate byte into byte register								MOV immediate word or double into word or double register							
	AL	CL	DL	BL	AH	CH	DH	BH	eAX	eCX	eDX	eBX	eSP	eBP	eSI	eDI
C	Shift Grp2		RET near		LES	LDS	MOV		ENTER	LEAVE	RET far		INT	INT	INTO	IRET
	Eb,Ib	Ev,Iv	Iw		Gv,Mp	Gv,Mp	Eb,Ib	Ev,Iv	Iw,Ib		Iw		3	Ib		
D	Shift Grp2				AAM	AAD		XLAT	ESC(Escape to coprocessor instruction set)							
	Eb,1	Ev,1	Eb,CL	Ev,CL												
E	LOOPNE	LOOPE	LOOP	JCXZ	IN		OUT		CALL	JNP			IN		OUT	
	Jb	Jb	Jb	Jb	AL,Ib	eAX,Ib	Ib,AL	Ib,eAX	Av	Jv	Ap	Jb	AL,DX	eAX,DX	DX,AL	DX,eAX
F	LOCK		REPNE	REP	HLT	CMC	Unary Grp3		CLC	STC	CLI	STI	CLD	STD	INC/DEC	Indirct
				REPE			Eb	Ev							Grp4	Grp5

Boot Process

- Execution begins at 0xFFFFFFF0 (Reset Vector)
  - 0xF000:0xFFF0 with 0xFFF00000 asserted by CPU
  - Jump to BIOS at 0xF0000
- BIOS loads first sector of disk at 0x1F0 (bootloader) to 0x7C00
  - Jumps to 0x7C00
- Bootloader loads kernel from disk to address 0x100000
  - Enters protected mode, etc.
  - Jumps to 0x10000C
- Kernel sets up virtual memory, Serial/Keyboard/CGA I/O

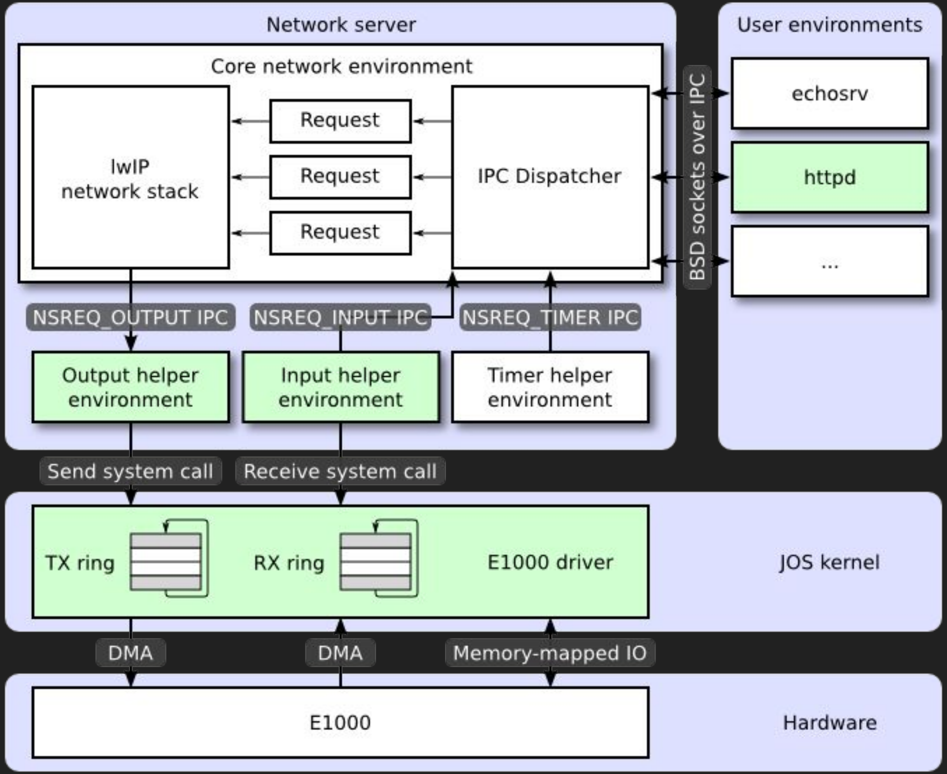
Extending JOS to Include Networking Capabilities

Zain Rehmani and Adithya Nott

First…..let’s run the Lab 6 scripts! :D

- We’ll let them run in the background
- We’ll show that all tests passed later on.

So… What is Lab 6 about?

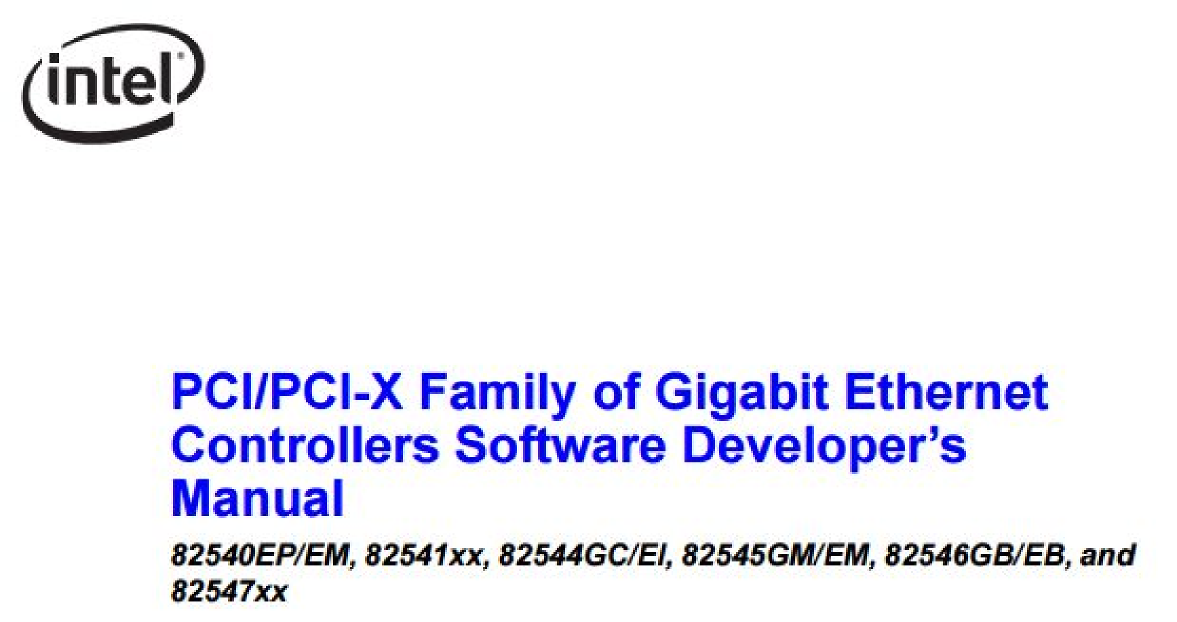


Exercise 1

- Yet another system/trap call. Just handling a clock interrupt to give JOS a sense of time.
- Allows JOS to have the ability to have the notion of network timeouts for the purposes of retransmission
- Clock interrupt that is generated by hardware every 10 ms
    - Just advance a variable each time the interrupt occurs to represent a timer

Exercise 2

- Literally just an RTFM exercise with reading Intel's guide on the E1000 driver.



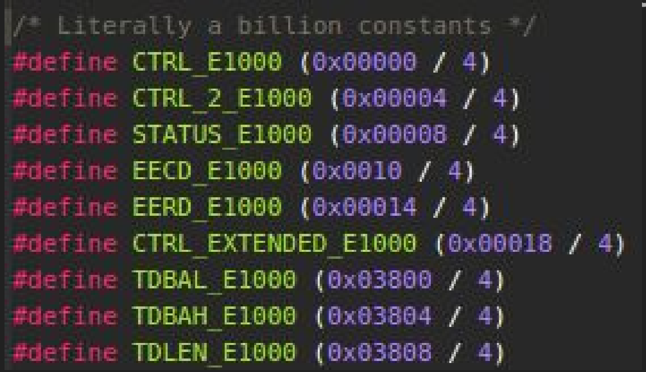
intel

PCI/PCI-X Family of Gigabit Ethernet Controllers Software Developer's Manual

82540EP/EM, 82541xx, 82544GC/EI, 82545GM/EM, 82546GB/EB, and 82547xx

Exercise 3-6, 9-10

- Now apply the manual’s sections to make the E1000 driver
- Literally writing e1000.c from scratch
    - “We have provided the kern/e1000.c and kern/e1000.h files for you so that you do not need to mess with the build system. They are currently blank; you need to fill them in for this exercise. You may also need to include the e1000.h file in other places in the kernel.”
- Not too hard to come across bugs when you write stuff from scratch



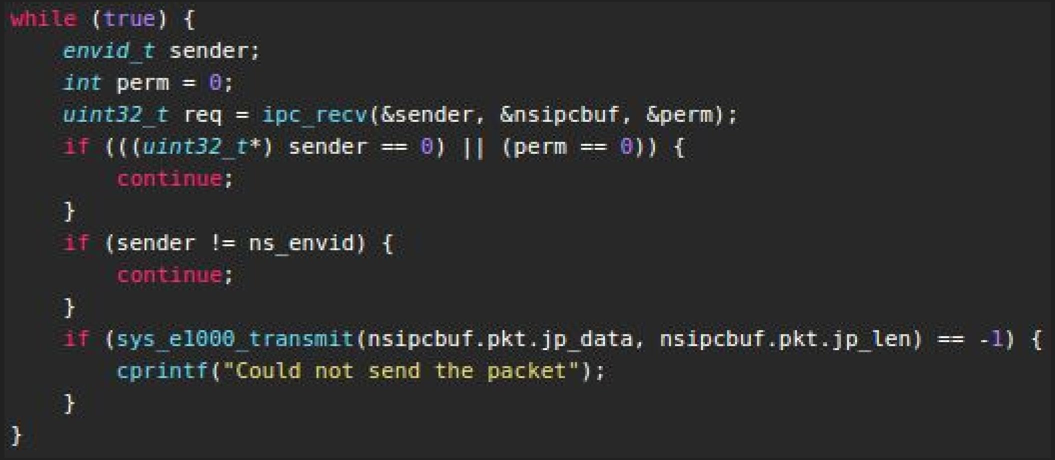
```c
/* Literally a billion constants */
#define CTRL_E1000 (0x00000 / 4)
#define CTRL_2_E1000 (0x00004 / 4)
#define STATUS_E1000 (0x00008 / 4)
#define EECD_E1000 (0x0010 / 4)
#define EERD_E1000 (0x00014 / 4)
#define CTRL_EXTENDED_E1000 (0x00018 / 4)
#define TDBAL_E1000 (0x03800 / 4)
#define TDBAH_E1000 (0x03804 / 4)
#define TDLEN_E1000 (0x03808 / 4)
```

Exercise 7

- Another syscall to transmit_packets from a userspace program.
    - The TXD_DD_E1000 flag is used to determine if there is space to transmit a packet on the tx_queue
    - When attempting to send a packet, if the tx_queue is full, it will drop the packet and attempt to transmit it another 10 times before giving up

Exercise 8

- Implementing net/output.c
    - Reads a packet from the network server
    - Sends the packet to the device driver



```
while (true) {
    envid_t sender;
    int perm = 0;
    uint32_t req = ipc_recv(&sender, &nsipcbuf, &perm);
    if (((uint32_t*) sender == 0) || (perm == 0)) {
        continue;
    }
    if (sender != ns_envid) {
        continue;
    }
    if (sys_e1000_transmit(nsipcbuf.pkt.jp_data, nsipcbuf.pkt.jp_len) == -1) {
        cprintf("Could not send the packet");
    }
}
```

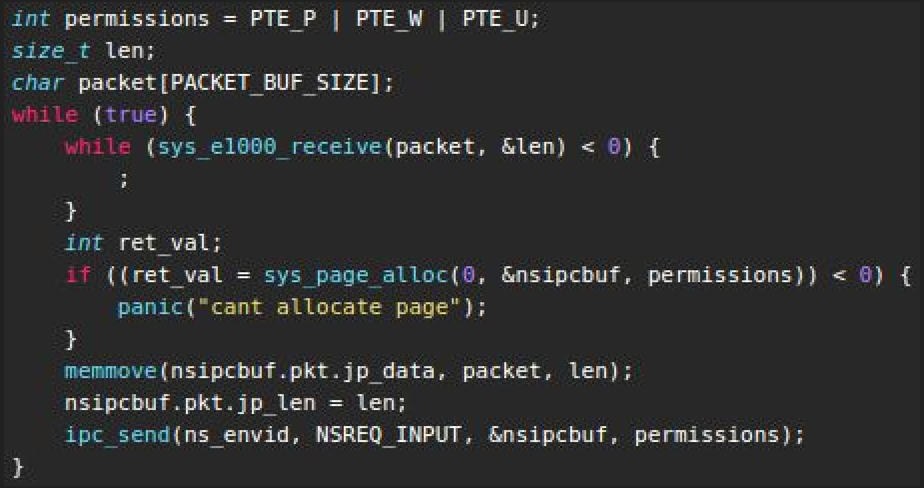
Exercise 11

- Function to receive packets as well as another system call
    - For the receive side, if the TXD_DD_E1000 flag is not set, then no packet has been received
    - If the receiving side is expecting a packet but nothing has yet been received, what should it do?
        - Option 1: Keep trying again
            - This is wasteful because the receive queue may be empty for a long stretch of time
        - Option 2: Suspend the calling environment until there are packets in the receive queue.

          Allow E1000 to generate interrupts on receive. Resume the environment that is blocked waiting for a packet.
            - More involved, but better. We chose to do this.

Exercise 12

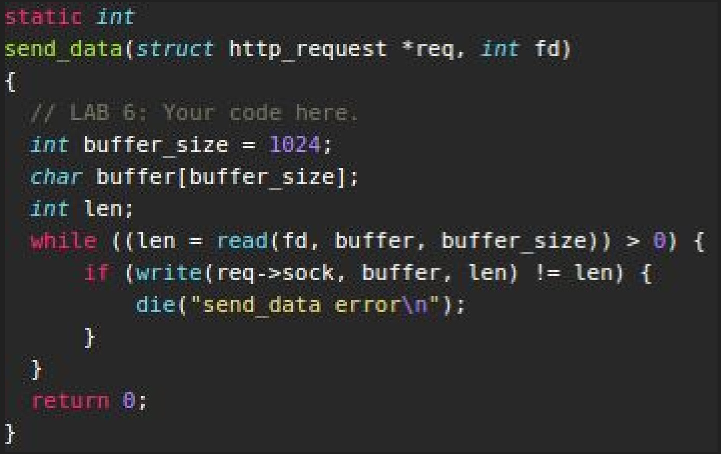
- Implementing net/input.c
  - Read a packet from the device driver
  - Send the packet to the network server



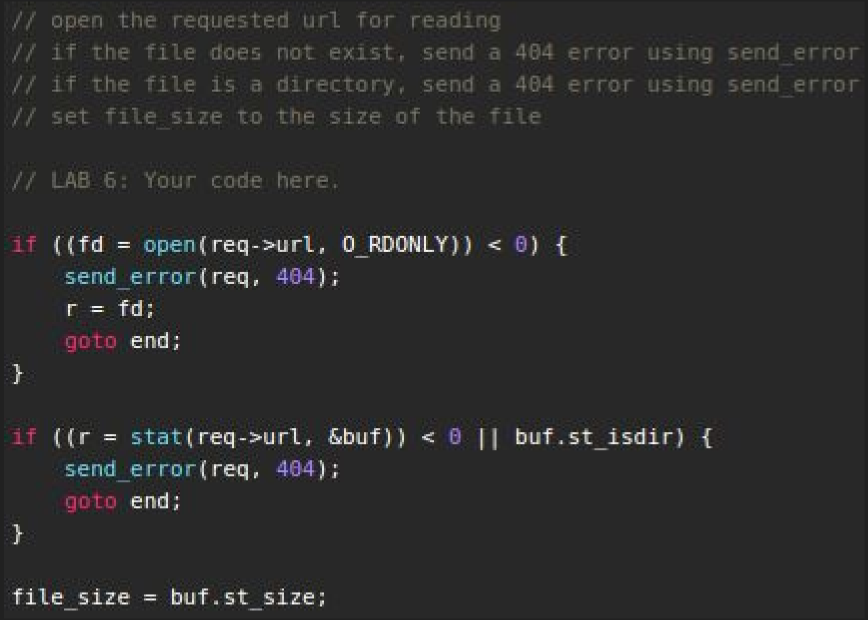
```
int permissions = PTE_P | PTE_W | PTE_U;
size_t len;
char packet[PACKET_BUF_SIZE];
while (true) {
    while (sys_e1000_receive(packet, &len) < 0) {
        ;
    }
    int ret_val;
    if ((ret_val = sys_page_alloc(0, &nsipcbuf, permissions)) < 0) {
        panic("cant allocate page");
    }
    memmove(nsipcbuf.pkt.jp_data, packet, len);
    nsipcbuf.pkt.jp_len = len;
    ipc_send(ns_envid, NSREQ_INPUT, &nsipcbuf, permissions);
}
```

Exercise 13

- Basic web server implementation, which can send the contents of a file to a requesting client
- Implement send_file and send_data



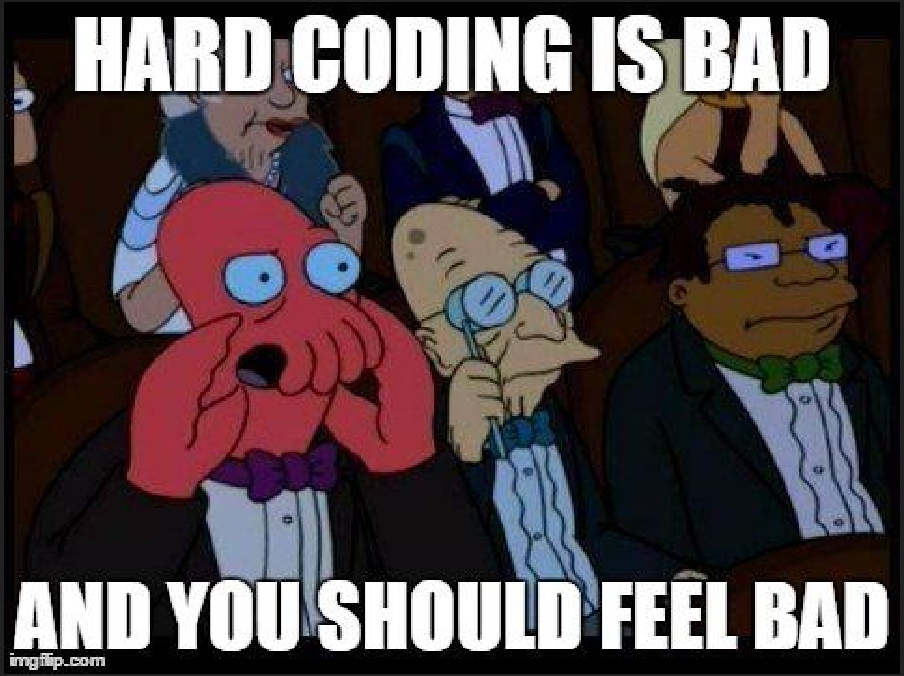
```c
static int
send_data(struct http_request *req, int fd)
{
  // LAB 6: Your code here.
  int buffer_size = 1024;
  char buffer[buffer_size];
  int len;
  while ((len = read(fd, buffer, buffer_size)) > 0) {
      if (write(req->sock, buffer, len) != len) {
          die("send_data error\n");
      }
  }
  return 0;
}
```



```c
// open the requested url for reading
// if the file does not exist, send a 404 error using send_error
// if the file is a directory, send a 404 error using send_error
// set file_size to the size of the file

// LAB 6: Your code here.

if ((fd = open(req->url, O_RDONLY)) < 0) {
    send_error(req, 404);
    r = fd;
    goto end;
}

if ((r = stat(req->url, &buf)) < 0 || buf.st_isdir) {
    send_error(req, 404);
    goto end;
}

file_size = buf.st_size;
```

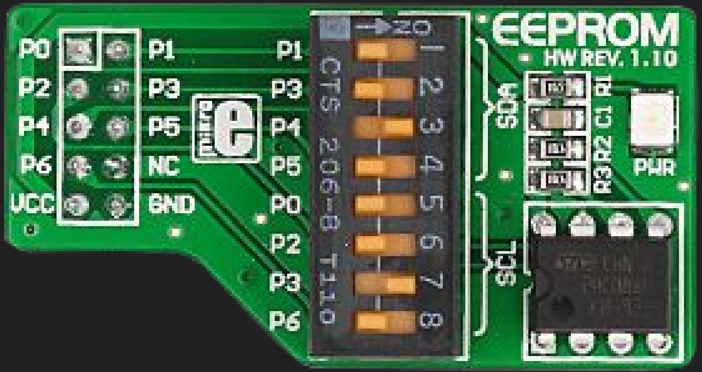
Beyond core Lab 6

- QEMU’s default MAC address is hardcoded in the code to work...



Electrically Erasable Programmable Read-Only Memory (EEPROM)

- Why not use the EEPROM to handle the MAC address initialization for us?
- Allows us to use multiple different MAC Address values with QEMU, with it being handled dynamically as opposed to only one preset value.



Implementation Details

- Loading MAC address out of EEPROM
- Yet another syscall…..this time to get the MAC address for lwIP
- lwIP modified to use this syscall instead of just some hardcoded value

- And now, a brief demo :D

What we would do with a bit more time

- Revisit DSM concept. A lot of it is just locking pages and sending page data over a network.
- More challenge problems
  - We have some code here and there going for the web chat server, but it’s not in a demoable state.
    - Idea in theory would have been to have multiple instances of a python script all being used to send messages and the web chat server would receive these messages and display them (with user customization with colors and the like :D)
    - Issues getting the right socket configuration with QEMU to interact with the script.

Difficulties we faced

- Got behind schedule due to uncaught bugs from Lab 4 impacting completion of Lab 5 (hindered progress on Lab 6)
- Very elusive bug in e1000.c that caused testinput and onward to fail for Lab 6
    - Took a long time to figure out why packets were missing
- Connectivity among different programs, let alone different computers
    - Interacting with QEMU via sockets for the web chat server problem
    - QEMU virtualization
- Now we see why MIT OCW descriptions of Lab 6 always seem to mention that it’s is the “default” final project.
    - It became clear that Lab 6 was more feasible than hardcore attempting DSM
    - Version of Lab 6 we saw via Google was different (E100 driver vs E1000 driver)
    - We didn’t get full details of Lab 6 until well after proposal

Any Questions?