Introduction

CS238P: Operating systems - Winter’18

UC Irvine, California

Logistics

- Graduate (MCS)
    - ~100 students
- Instructor: Anton Burtsev
- Meeting time: 3:30-4:50pm (M, W, F)
- 2 TAs
    - Vikram Naranayan, Junjie Shen
- Web page: `http://www.ics.uci.edu/~aburtsev/238P`
- Piazza: `https://www.piazza.com/uci/winter2018/cs238p`
- Mailing list: TBD
- Office hours: TBD

Logistics

- 4-5 homeworks
  - Implement a shell
  - Explain what's on the stack
  - Implement a system call
  - Change file system layout
- Grading (curved)
  - Exams (40%)
    - Midterm - 15%
    - Final - 25%
  - Homeworks - 60%
- Late submission policy
  - You can submit homework 3 days after the deadline for 60% of your grade

This course

- Inspired by
  - MIT 6.828: Operating System Engineering
  - `https://pdos.csail.mit.edu/6.828/2016/`
- We will use xv6
  - Relatively simple (9K lines of code)
  - Reasonably complete UNIX kernel
  - `https://pdos.csail.mit.edu/6.828/2016/xv6.html`
- xv6 comes with a book
  - `https://pdos.csail.mit.edu/6.828/2016/xv6/book-rev9.pdf`
- And source code printout
  - `https://pdos.csail.mit.edu/6.828/2016/xv6/xv6-rev9.pdf`

Other references

“Operating Systems: Three Easy Pieces” (OSTEP) by Remzi H. Arpaci-Dusseau and Andrea C. Arpaci-Dusseau

- Free online version - `http://pages.cs.wisc.edu/~remzi/OSTEP/`

Course organization

- Lectures
  - High level concepts and abstractions
- Reading
  - xv6 book + source code
  - Bits of OSTEP book
- Homeworks
  - Coding real parts of the xv6 kernel
- Design riddles
  - Understanding design trade-offs, explaining parts of xv6

Prerequisites

- Solid C programming skills
    - xv6 is written in C
    - You need to read, code and debug
    - All homeworks are in C
    - Many questions will require explaining xv6 code
- Be able to work and code on Linux/UNIX environment
- Some assembly skills

C Programming

Conditional statements

- `if...else`

```
int pid = fork();
if (pid == -1) {
  perror("fork:");
} else {
  // do the needful
}
```

- `switch...case`

```
switch(cmd->type){
  case '>': ...; break;
  default: ...; break;
}
```

Loops

- for

```
for (i = 0; i < ncpu; ++i) {
  if (cpus[i].apicid == apicid)
    return i;
}
```

- while

```
while(*path == '/')
  path++;
```

- do...while

```
do {
  buf[i++] = digits[x % base];
} while((x /= base) != 0);
```

Functions

- Process creation (`fork, exec`)

```
pid = fork();
if(pid == 0)
  exec("sh", argv);
```

Functions

- Process creation (`fork, exec`)

  ```
  pid = fork();
  if(pid == 0)
    exec("sh", argv);
  ```

- File I/O (`open, close, read, write`)

  ```
  fd = open(rcmd->file, rcmd->mode);
  read(fd, ...);
  close(fd);
  ```

Arrays

- Collection of objects of the same data type

Arrays

- Collection of objects of the same data type
- Accessed by index (`0 ... size - 1`)

Arrays

- Collection of objects of the same data type
- Accessed by index (`0 ... size - 1`)
- String is an array of characters

Arrays

- Collection of objects of the same data type
- Accessed by index (`0 ... size - 1`)
- String is an array of characters
- No reference operator

```
printf("Address of a \%p | \%p\n", a, &a);
>> Address of a 0x7aff07024060 | 0x7aff07024060
```

Array Intialization

Designated Initializers[1]

```
#define CAPSLOCK (1<<3)
#define NUMLOCK (1<<4)
#define SCROLLLOCK (1<<5)
static uchar togglecode[256] = {
  [0x3A] CAPSLOCK,
  [0x45] NUMLOCK,
  [0x46] SCROLLLOCK
};
/* equivalent to */
togglecode[0x3A] = CAPSLOCK;
togglecode[0x45] = NUMLOCK;
togglecode[0x46] = SCROLLLOCK;
```

Initialize the array elements 0x3A, 0x45, 0x46 only [2]

[1] http://gcc.gnu.org/onlinedocs/gcc-4.0.4/gcc/Designated-Inits.html

[2] sheet 77, xv6-rev9.pdf

Structures

```c
struct execcmd {
  int type;
  char *argv[MAXARGS];
};
```

Structures

```
struct execcmd {
  int type;
  char *argv[MAXARGS];
};
```

- Collection of objects of different data type

Structures

```
struct execcmd {
  int type;
  char *argv[MAXARGS];
};
```

- Collection of objects of different data type
- Declare a variable (`struct execcmd e`) or a pointer (`struct execcmd *ep`)

Structures

```
struct execcmd {
  int type;
  char *argv[MAXARGS];
};
```

- Collection of objects of different data type
- Declare a variable (`struct execcmd e`) or a pointer (`struct execcmd *ep`)
- Accessed elements by dot (`e.type`) or arrow operator (`ep->elem`)

Structures

```
struct execcmd {
  int type;
  char *argv[MAXARGS];
};
```

- Collection of objects of different data type
- Declare a variable (`struct execcmd e`) or a pointer (`struct execcmd *ep`)
- Accessed elements by dot (`e.type`) or arrow operator (`ep->elem`)
- Compiler generates the appropriate offset in the assembly code

Typecasting

- Change the data type of a variable/object for a single operation

  ```
  var = (dest_type) source;
  ```

Typecasting

- Change the data type of a variable/object for a single operation

  ```
  var = (dest_type) source;
  ```

- Pass generic datatypes/objects

  ```
  struct cmd { int type; };
  struct execcmd {
    int type;
    char *argv[MAXARGS];
  };
  void runcmd(struct cmd *cmd) {
      ...
      ecmd = (struct execcmd*)cmd;
  }
  struct cmd* execcmd(void) {
    struct execcmd *cmd;
    ...
    return (struct cmd*)cmd;
  }
  ```

Typecasting

- Change the data type of a variable/object for a single operation

  ```
  var = (dest_type) source;
  ```

- Pass generic datatypes/objects

  ```
  struct cmd { int type; };
  struct execcmd {
    int type;
    char *argv[MAXARGS];
  };
  void runcmd(struct cmd *cmd) {
      ...
      ecmd = (struct execcmd*)cmd;
  }
  struct cmd* execcmd(void) {
    struct execcmd *cmd;
    ...
    return (struct cmd*)cmd;
  }
  ```

- Beware of strings!

Bit fields[3]

```
// Gate descriptors for interrupts and traps
struct gatedesc {
  uint off_15_0 : 16; // low 16 bits of offset in segment
  uint cs : 16; // code segment selector
  uint args : 5; // # args, 0 for interrupt/trap gates
  uint rsv1 : 3; // reserved(should be zero I guess)
  uint type : 4; // type(STS_{TG,IG32,TG32})
  uint s : 1; // must be 0 (system)
  uint dpl : 2; // descriptor(meaning new) privilege level
  uint p : 1; // Present
  uint off_31_16 : 16; // high bits of offset in segment
};

struct gatedesc d;
d.s = 0; d.args = 0;
```

[3] sheet 09 xv6-rev9.pdf

Access low-level data

32 36	37 39	40 43	44	45 46	47	48 63
args	rsvd	type	s	dpl	p	offset high

0 15	16 31
offset	segment

Access low-level data

32 36	37 39	40 43	44	45 46	47	48 63
args	rsvd	type	s	dpl	p	offset high

0 15	16 31
offset	segment

- Set bit 44 (s) - Or (|) it

  ```
  /* on a 64-bit data type */
  data = data | (1 << 44);
  data |= (1 << 44);
  ```

Access low-level data

32 36	37 39	40 43	44	45 46	47	48 63
args	rsvd	type	s	dpl	p	offset high

0 15	16 31
offset	segment

- Set bit 44 (s) - Or (|) it

```
/* on a 64-bit data type */
data = data | (1 << 44);
data |= (1 << 44);
```

- Clear a bit (s) - And (&) and Not (∼)

```
/* on a 64-bit data type */
data = data & ~(1 << 44);
data &= ~(1 << 44);
```

Dynamic registration

- Declare a struct to hold function pointers [4]

```
#define NDEV  10
#define CONSOLE 1
struct devsw {
  int (*read)(struct inode*, char*, int);
  int (*write)(struct inode*, char*, int);
};
struct devsw devsw[NDEV]; /* global data structure */
```

[4] sheet 40 xv6-rev9.pdf

[5] sheet 82 xv6-rev9.pdf

Dynamic registration

- Declare a struct to hold function pointers [4]

```
#define NDEV  10
#define CONSOLE 1
struct devsw {
  int (*read)(struct inode*, char*, int);
  int (*write)(struct inode*, char*, int);
};
struct devsw devsw[NDEV]; /* global data structure */
```

- Register function pointer [5]

```
int consolewrite(struct inode *ip, char *buf, int n);
int consoleread(struct inode *ip, char *dst, int n);
devsw[CONSOLE].write = consolewrite;
devsw[CONSOLE].read = consoleread;
```

[4] sheet 40 xv6-rev9.pdf

[5] sheet 82 xv6-rev9.pdf

Pointers & buffer management

- Access raw memory

```
#define KERNBASE 0x80000000
#define P2V(a) (((void *) (a)) + KERNBASE)
uchar *code;
code = P2V(0x7000);
```

Pointers & buffer management

- Access raw memory

```
#define KERNBASE 0x80000000
#define P2V(a) (((void *) (a)) + KERNBASE)
uchar *code;
code = P2V(0x7000);
```

- kalloc, memset, kfree

```
mem = kalloc(); /* allocate a page */
memset(mem, 0, PGSIZE); /* memset */
kfree(mem); /* free it when done */
```

Pointers & buffer management

- Access raw memory

```
#define KERNBASE 0x80000000
#define P2V(a) (((void *) (a)) + KERNBASE)
uchar *code;
code = P2V(0x7000);
```

- kalloc, memset, kfree

```
mem = kalloc(); /* allocate a page */
memset(mem, 0, PGSIZE); /* memset */
kfree(mem); /* free it when done */
```

- memcpy, memmove

```
/* move start to code */
memmove(code, _binary_entryother_start,
      (uint)_binary_entryother_size);
```

Debugging - gdb

gdbinit

- gdbinit - `https://raw.githubusercontent.com/gdbinit/Gdbinit/master/gdbinit`
- cheatsheet - `http://darkdust.net/files/GDB%20Cheat%20Sheet.pdf`

Makefile

- A build automation tool for compiling libraries, executables, etc.
- Rules are written on a text based `Makefile`

Operating system

What is an OS?

A layer of abstraction between the underlying hardware and the application programs. Two main goals:

What is an OS?

A layer of abstraction between the underlying hardware and the application programs. Two main goals:

- Resource virtualization (CPU, memory, etc.)

What is an OS?

A layer of abstraction between the underlying hardware and the application programs. Two main goals:

- Resource virtualization (CPU, memory, etc.)
- Resource management

Virtualization

- CPU Virtualization
  - One CPU and ’n’ processes

Virtualization

- CPU Virtualization
  - One CPU and ’n’ processes
  - Illusion: Each process has the entire CPU

Virtualization

- CPU Virtualization
  - One CPU and ’n’ processes
  - Illusion: Each process has the entire CPU
  - Manage resource by scheduling processes

Virtualization

- CPU Virtualization
  - One CPU and ’n’ processes
  - Illusion: Each process has the entire CPU
  - Manage resource by scheduling processes
- Memory Virtualization

Virtualization

- CPU Virtualization
  - One CPU and ’n’ processes
  - Illusion: Each process has the entire CPU
  - Manage resource by scheduling processes
- Memory Virtualization
  - Illusion: Every process has the entire memory

Virtualization

- CPU Virtualization
    - One CPU and ’n’ processes
    - Illusion: Each process has the entire CPU
    - Manage resource by scheduling processes
- Memory Virtualization
    - Illusion: Every process has the entire memory
- Device virtualization

Questions?