VIRTUALIZATION: CPU

Shivaram Venkataraman
CS 537, Spring 2020
- Project 1a is out! Due Jan 29 at 10.00pm
- Signup for Piazza https://piazza.com/wisc/spring2020/cs537
- Lecture notes at pages.cs.wisc.edu/~shivaram/cs537-sp20/
- Drop? Waitlist? Email enrollment@cs.wisc.edu and cc me
AGENDA / OUTCOMES

Abstraction
  What is a Process? What is its lifecycle?

Mechanism
  How does process interact with the OS?
  How does the OS switch between processes?
ABSTRACTION: PROCESS
# include <stdio.h>
# include <stdlib.h>
# include "common.h"

int main(int argc, char *argv[])
{
    char *str = argv[1];

    while (1) {
        printf("%s\n", str);
        Spin(1);
    }
    return 0;
}
WHAT IS A PROCESS?

Stream of executing instructions and their "context"

Instruction Pointer

Registers

Contents of
Memory addr

Contents of malloc

File descriptors

```
pushq %rbp
movq %rsp, %rbp
subq $32, %rsp
movl $0, -4(%rbp)
movl %edi, -8(%rbp)
movq %rsi, -16(%rbp)
cmpl $2, -8(%rbp)
je LBB0_2
```
PROCESS CREATION

CPU

Memory

Program

code
static data

Physical memory or DRAM

int main() {

char arr = {"h", "e", "l", "l", "o"};

Role of OS is to run specified program by creating processes
Can run multiple instances of same program

Each program has its own stack, heap etc.
PROCESS VS THREAD

Threads: “Lightweight process”

Execution streams that share an address space
Can directly read / write memory

Can have multiple threads within a single process

Demo?
SHARING THE CPU
SHARING CPU

code, static data
heap
stack

P1

P2

P3

time sharing
0 to 10 ms P1
10 to 20 ms P2

code, static data
heap
stack

Paused

code, static data
heap
stack

Paused

CPU
TIME SHARING

code, static data
heap
stack

CPU

unpause P2
TIME SHARING

code, static data
heap
stack

code, static data
heap
stack

code, static data
heap
stack

CPU
WHAT TO DO WITH PROCESSES THAT ARE NOT RUNNING?

OS Scheduler

Save context when process is paused
Restore context on resumption
STATE TRANSITIONS

Running → Ready

Ready → Scheduled → Running

Scheduled → Running

Blocked → I/O: initiate → Ready

I/O: done → Blocked

Using the CPU

e.g., read from disk

Scheduler

Launch

Ready

P1, P3

Blocked

P2
STATE TRANSITIONS

- Running
- Ready
- Blocked
- Descheduled
- Scheduled

I/O: initiate → Blocked

I/O: done → Ready

Descheduled → Scheduled

(States transitions diagram)
ASIDE: OSTEP HOMEWORKS!

- Optional homeworks corresponding to each chapter in book
- Little simulators to help you understand
- Can generate problems and solutions!

http://pages.cs.wisc.edu/~remzi/OSTEP/Homework/homework.html
Run ./process_run.py -l 2:100,2:0
<table>
<thead>
<tr>
<th>Time</th>
<th>PID: 0</th>
<th>PID: 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>RUN:io</td>
<td>READY</td>
</tr>
<tr>
<td>2</td>
<td>WAITING</td>
<td>RUN:cpu</td>
</tr>
<tr>
<td>3</td>
<td>WAITING</td>
<td>RUN:io</td>
</tr>
<tr>
<td>4</td>
<td>WAITING</td>
<td>WAITING</td>
</tr>
<tr>
<td>5</td>
<td>WAITING</td>
<td>WAITING</td>
</tr>
<tr>
<td>6</td>
<td>RUN:io</td>
<td>WAITING</td>
</tr>
<tr>
<td>7</td>
<td>WAITING</td>
<td>WAITING</td>
</tr>
<tr>
<td>8</td>
<td>WAITING</td>
<td>RUN:io</td>
</tr>
</tbody>
</table>

What happens at time 8?

Each IO takes 5 time units
Policy goals

Virtualize CPU resource using processes
Reschedule process for fairness? efficiency?

Mechanism goals

Efficiency: Sharing should not add overhead
Control: OS should be able to intervene when required
EFFICIENT EXECUTION

Simple answer !?: **Direct Execution**
- Allow user process to run directly
- Create process and transfer control to main()

Challenges
- What if the process wants to do something restricted? Access disk?
- What if the process runs forever? Buggy? Malicious?

Solution: **Limited Direct Execution (LDE)**
PROBLEM 1: RESTRICTED OPS

How can we ensure user process can’t harm others?

Solution: privilege levels supported by hardware (bit of status)

User processes run in user mode (restricted mode)
OS runs in kernel mode (not restricted)

How can process access devices?

System calls (function call implemented by OS)
SYSTEM CALL

Similar to library call except executed with higher permission level.
Process P wants to call read()
Process P

P can only see its own memory because of user mode
(other areas, including kernel, are hidden)

P wants to call read() but no way to call it directly
SYSTEM CALL

Process P

movl $6, %eax;   int $64

sys_read

RAM
movl $6, %eax;  int $64
Process P

movl $6, %eax; int $64
Follow entries to correct system call code
Kernel can access user memory to fill in user buffer
return-from-trap at end to return to Process P
SYSCALL SUMMARY

Separate user-mode from kernel mode for security

Syscall: call kernel mode functions
- Transfer from user-mode to kernel-mode (trap)
- Return from kernel-mode to user-mode (return-from-trap)
QUIZ 2

To call SYS_read the instructions we used were:

```
movl $6, %eax
int $64
```

To call SYS_exec what will be the instructions?

```
movl $9, %eax
int $64
```

---

// System call numbers

```
#define SYS_fork 1
#define SYS_exit 2
#define SYS_wait 3
#define SYS_pipe 4
#define SYS_write 5
#define SYS_read 6
#define SYS_close 7
#define SYS_kill 8
#define SYS_exec 9
#define SYS_open 10
```
PROBLEM2: HOW TO TAKE CPU AWAY

**Policy**
- To decide which process to schedule when
  - Decision-maker to optimize some workload performance metric

**Mechanism**
- To switch between processes
  - Low-level code that implements the decision

Separation of policy and mechanism: Recurring theme in OS
OS runs **dispatch loop**

```c
while (1) {
    run process A for some time-slice
    stop process A and save its context
    load context of another process B
}
```

**Question 1:** How does dispatcher gain control?
**Question 2:** What must be saved and restored?
How does Dispatcher get control?

Option 1: *Cooperative Multi-tasking:* Trust process to relinquish CPU through traps

- Examples: System call, page fault (access page not in main memory), or error (illegal instruction or divide by zero)
- Provide special `yield()` system call

![Diagram showing process P1 yielding to OS, which then yields to process P2.](image-url)
PROBLEMS WITH COOPERATIVE?

Disadvantages: Processes can misbehave

By avoiding all traps and performing no I/O, can take over entire machine
Only solution: Reboot!

Not performed in modern operating systems
Option 2: Timer-based Multi-tasking

Guarantee OS can obtain control periodically

Enter OS by enabling periodic alarm clock
  Hardware generates timer interrupt (CPU or separate chip) Example: Every 10ms
  User must not be able to mask timer interrupt
<table>
<thead>
<tr>
<th>Operating System</th>
<th>Hardware</th>
<th>Program</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Process A</td>
</tr>
</tbody>
</table>
timer interrupt
save regs(A) to k-stack(A)
moves to kernel mode
jump to trap handler
Handle the trap
Call switch() routine
save kernel regs(A) to proc-struct(A)
restore kernel regs(B) from proc-struct(B)
switch to k-stack(B)
return-from-trap (into B)

Hardware

timer interrupt
save regs(A) to k-stack(A)
move to kernel mode
jump to trap handler
Operating System

Handle the trap
Call switch() routine
  save kernel regs(A) to proc-struct(A)
  restore kernel regs(B) from proc-struct(B)
  switch to k-stack(B)
  return-from-trap (into B)

Hardware

timer interrupt
  save regs(A) to k-stack(A)
  move to kernel mode
  jump to trap handler

Program

Process A

  restore regs(B) from k-stack(B)
  move to user mode
  jump to B’s IP
Operating System:

Handle the trap
Call switch() routine
save kernel regs(A) to proc-struct(A)
restore kernel regs(B) from proc-struct(B)
switch to k-stack(B)
return-from-trap (into B)

Hardware:

timer interrupt
save regs(A) to k-stack(A)
move to kernel mode
jump to trap handler

Program:

Process A

Process B

restore regs(B) from k-stack(B)
move to user mode
jump to B’s IP
SUMMARY

Process: Abstraction to virtualize CPU
Use time-sharing in OS to switch between processes

Key aspects
- Use system calls to run access devices etc. from user mode
- Context-switch using interrupts for multi-tasking
Running \(\rightarrow\) Ready

Descheduled \(\leftrightarrow\) Scheduled

I/O: initiate \(\rightarrow\) Blocked

I/O: done \(\rightarrow\) Scheduled

POLICY?

NEXT CLASS!
NEXT STEPS

Project 1a: Due Jan 29th (Wednesday) at 10pm
Project 1b: Out on Jan 29th

Discussion section: Thursday 5.30pm-6.30pm at 105 Psychology

Waitlist? Email enrollment@cs.wisc and cc me (will finalize by Monday)