TCSS 422: OPERATING SYSTEMS

Process API,
Limited Direct Execution

Wes J. Lloyd
Institute of Technology
University of Washington - Tacoma

FEEDBACK FROM - 01/09

- Use of fork()
- How do parent and child processes interact with each other?
  - The parent starts the child, and can wait() until it finishes.
  - Nothing prevents the parent from exiting while a child continues to execute – they are separate processes
- Is there context switching time at the end of a process or simply in the middle?
  - When a process terminates its data structure would be deallocated, but this is not a context switch per se

FEEDBACK - 2

- Minimal CentOS Install w/ Developer Tools: though it works the problem is not being able to easily have multiple windows for dev & debug.
  - https://lecturesnippets.com/lesson/setting-up-ssh-server-in-centos-7-minimal-install/
  - http://www.tecmint.com/things-to-do-after-minimal-rhel-centos-7-installation/2/
- Is forked child that calls exec still considered a child of the parent?
  - YES

FEEDBACK - 3

- What are the differences between execl, execv, execvp?
  - From the man pages:
    execl,execlp,execle – the argument list is provided as a *list of one or more pointers* to null terminated string (`const char *`). The list must be null terminated.
    execv, execvp,execvpe – the argument list is provided as an *array*:
    of null terminated strings → `const *char argv[]`
  - **execle,execvpe –** include an extra parameter to allow the environment to be passed in
    - To see your environment try "`printenv`" or "`export`"
  - **execl, execle, execv** – allow the "path" which is searched to find the executable program to be provided
    - To see your path, type `echo $PATH`

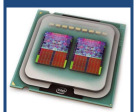
FEEDBACK - 4

- What determine a program's access level to the underlying system? Can a user process be escalated to run at a more direct (privileged) level?
  - The operating system controls the privilege level
  - The OS will escalate from USER to KERNEL mode, for example, to perform I/O
- What does the OS handle if we (user processes???) aren't allowed direct execution?
- White board positioning

OBJECTIVES

- Limited Direct Execution – Ch. 6
- Scheduling Introduction
- Scheduling Metrics
- Scheduling Methods

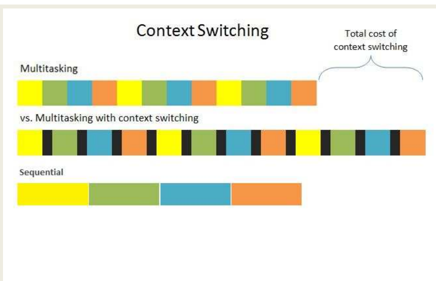
LIMITED DIRECT EXECUTION



CONTROL TRADEOFF

- Too much control:
  - No security
  - No time sharing
- Too little control:
  - Too much OS overhead
  - Poor performance for compute & I/O
  - Complex APIs (system calls), difficult to use

CONTEXT SWITCHING OVERHEAD



LIMITED DIRECT EXECUTION

- OS implements LDE to support time/resource sharing
- Enabled by ***protected (safe) control transfer***
- CPU supported context switch
- Provides data isolation

CPU MODES

- Utilize CPU Privilege Rings (Intel x86)
  - rings 0 (kernel), 1 (VM kernel), 2 (unused), 3 (user)

  access ⟵ no access
- **User mode**:
  Application is running, but w/o direct I/O access
- **Kernel mode**:
  OS kernel is running performing restricted operations

CPU MODES

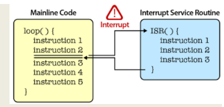
- User mode: ring 3 - untrusted
  - Some instructions and registers are disabled by the CPU
  - Exception registers
  - HALT instruction
  - MMU instructions
  - OS memory access
  - I/O device access
- Kernel mode: ring 0 – trusted
  - All instructions and registers enabled

SYSTEM CALLS

- Enable restricted "OS" operations
- Kernel exposes key functions through an API:
  - Device I/O
  - Task swapping: context switch
  - Memory management/allocation: malloc()
  - Creating/destroying processes

TRAPS: SYSTEM CALLS, EXCEPTIONS, INTERRUPTS

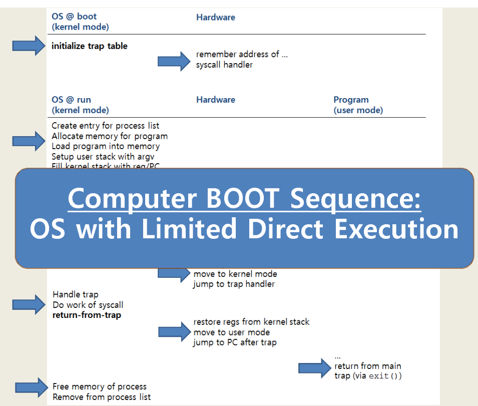
- Trap: any transfer to kernel mode



- Three kinds of traps
  - Sys call (planned) user → kernel
    - SYSCALL for I/O, etc.
  - Exception (error) user → kernel
    - Div by zero, page fault, page protection error
  - Interrupt: (event) user → kernel
    - Non-maskable vs. maskable
    - Keyboard event, network packet arrival, timer ticks
    - Memory parity error (ECC), hard drive failure

EXCEPTION TYPES

Exception type	Synchronous vs. asynchronous	User request vs. coerced	User maskable vs. nonmaskable	Within vs. between instructions	Resume vs. terminate
I/O device request	Asynchronous	Coerced	Nonmaskable	Between	Resume
Invoke operating system	Synchronous	User request	Nonmaskable	Between	Resume
Tracing instruction execution	Synchronous	User request	User maskable	Between	Resume
Breakpoint	Synchronous	User request	User maskable	Between	Resume
Integer arithmetic overflow	Synchronous	Coerced	User maskable	Within	Resume
Floating-point arithmetic overflow or underflow	Synchronous	Coerced	User maskable	Within	Resume
Page fault	Synchronous	Coerced	Nonmaskable	Within	Resume
Misaligned memory accesses	Synchronous	Coerced	User maskable	Within	Resume
Memory protection violation	Synchronous	Coerced	Nonmaskable	Within	Resume
Using undefined instruction	Synchronous	Coerced	Nonmaskable	Within	Terminate
Hardware malfunction	Asynchronous	Coerced	Nonmaskable	Within	Terminate
Power failure	Asynchronous	Coerced	Nonmaskable	Within	Terminate



MULTITASKING

- How/when should the OS regain control of the CPU to switch between processes?
- Cooperative multitasking (mostly pre 32-bit)
  - < ...
  - Op... l
    - ...
    - When performing I/O
    - Illegal operations
  - What problems could you for see with this approach?

> A process gets stuck in an infinite loop.
> → **Reboot the machine**

MULTITASKING - 2

- Preemptive multitasking (32 & 64 bit OSes)
- >= Mac OSX, Windows 95+
- Timer...
  - Rais...
  - Inte...
    1. Current program is halted
    2. Program states are saved
    3. OS Interrupt handler is run (kernel mode)
- What is a good interval for the timer interrupt?

> A **timer interrupt** gives OS the ability to run again on a CPU.

CONTEXT SWITCH

- Preemptive multitasking initiates "trap" into the OS code to determine:
  - Whether to continue running the **current process**, or switch to a **different one**.
  - If the decision is made to switch, the OS performs a context switch swapping out the current process for a new one.

CONTEXT SWITCH - 2

1. Save register values of the current process to its kernel stack
   - General purpose registers
   - PC: program counter (instruction pointer)
   - kernel stack pointer
2. Restore soon-to-be-executing process from its kernel stack
3. Switch to the kernel stack for the soon-to-be-executing process

OS @ boot (kernel mode) | **Hardware**

- **initialize trap table**
  - remember address of … syscall handler timer handler
- **start interrupt timer**
  - start timer interrupt CPU in X ms

OS @ run (kernel mode) | **Hardware** | **Program (user mode)**

Context Switch

- Call switch() routine
  save regs(A) to proc-struct(A)
  restore regs(B) from proc-struct(B)
  switch to k-stack(B)
  return-from-trap (into B)
  - restore regs(B) from k-stack(B) move to user mode jump to B's PC
    - Process B …

INTERRUPTED INTERRUPTS

- What happens if during an interrupt (trap to kernel mode), another interrupt occurs?
- Linux
  - < 2.6 kernel: non-preemptive kernel
  - >= 2.6 kernel: preemptive kernel

PREEMPTIVE KERNEL

- Use "locks" as markers of regions of non-preemptibility (non-maskable interrupt)
- Preemption counter (`preempt_count`)
  - begins at zero
  - increments for each lock acquired (not safe to preempt)
  - decrements when locks are released
- Interrupt can be interrupted when `preempt_count=0`
  - It is safe to preempt (maskable interrupt)
  - the interrupt is more important

SCHEDULING: INTRODUCTION

SCHEDULING INTRODUCTION

- For simplicity, consider job scheduling with limitations:
  - Each job requires the same CPU time
  - All jobs arrive at the same time
  - All jobs only use the CPU (no I/O)
  - The run-time of each job is known a priori

SCHEDULING METRICS

- **Metrics**: A standard measure to quantify to what degree a system possesses some property. Metrics provide *repeatable* techniques to quantify and compare systems.
- **Measurements** are the numbers derived from the application of metrics

- Scheduling Metric: **Turnaround time**
- The time at which the job completes minus the time at which the job arrived in the system

$$T_{turnaround} = T_{completion} - T_{arrival}$$

- How is turnaround time different than execution time?

SCHEDULING METRICS - 2

- Scheduling Metric: **Fairness**
  - Jain's fairness index
  - Quantifies if jobs receive a fair share of system resources

$$\mathcal{J}(x_1, x_2, \ldots, x_n) = \frac{(\sum_{i=1}^{n} x_i)^2}{n \cdot \sum_{i=1}^{n} x_i^2}$$

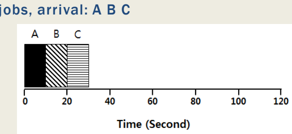
- n processes
- x_i is time share of each process
- worst case = 1/n
- best case = 1

- Consider n=3, worst case = .333, best case=1
- With n=3 and x_1=.2, x_2=.7, x_3=.1, fairness=.62
- With n=3 and x_1=.33, x_2=.33, x_3=.33, fairness=1

SCHEDULERS

- FIFO: first in, first out
  - Very simple, easy to implement

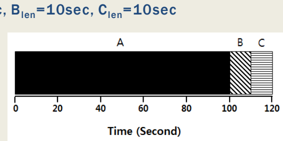
- Consider
  - 3 x 10sec jobs, arrival: A B C



$$Average\ turnaround\ time = \frac{10+20+30}{3} = 20\ sec$$

FIFO: CONVOY EFFECT

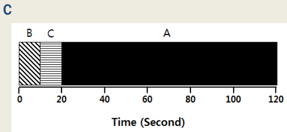
- FIFO with different jobs lengths
- Consider
  - A_{len}=100sec, B_{len}=10sec, C_{len}=10sec



$$Average\ turnaround\ time = \frac{100+110+120}{3} = 110\ sec$$

SJF: SHORTEST JOB FIRST

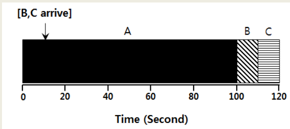
- Given that we know execution times in advance:
  - Run in order of duration, shortest to longest
  - Non preemptive scheduler
  - This is not realistic
  - Arrival: A B C



$$Average\ turnaround\ time = \frac{10+20+120}{3} = 50\ sec$$

SJF: WITH RANDOM ARRIVAL

- If jobs arrive at any time:
- A @ t=0sec, B @ t=10sec, C @ t=10sec



$$Average\ turnaround\ time = \frac{100 + (110 - 10) + (120 - 10)}{3} = 103.33\ sec$$

January 11, 2017 | TCSS422: Operating Systems [Fall 2016] Institute of Technology, University of Washington - Tacoma | L3.31

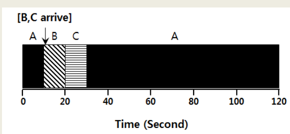
STCF – SHORTEST TIME TO COMPLETION FIRST

- Add preemption to the Shortest Job First scheduler
  - Also called preemptive shortest job first (PSJF)
- When a new job enters the system:
  - Of all jobs, *Which has the least time left?*
  - PREMPT job execution, and schedule the **new** shortest job
- More realistic, but how do we know execution time in advance?
  - Oracle: All knowing one
  - Only schedule static (fixed size) batch workloads
  - Can we predict execution time?

January 11, 2017 | TCSS422: Operating Systems [Fall 2016] Institute of Technology, University of Washington - Tacoma | L3.32

STCF - 2

- Consider:
  - A_{len}=100 $A_{arrival}$=0
  - B_{len}=10, $B_{arrival}$=10, C_{len}=10, $C_{arrival}$=10



$$Average\ turnaround\ time = \frac{(120 - 0) + (20 - 10) + (30 - 10)}{3} = 50\ sec$$

January 11, 2017 | TCSS422: Operating Systems [Fall 2016] Institute of Technology, University of Washington - Tacoma | L3.33

SCHEDULING METRICS - 3

- Scheduling Metric: **Response Time**
- Time from when job arrives until it starts execution

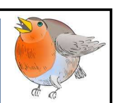
$$T_{response} = T_{firstrun} - T_{arrival}$$

- STCF, SJF, FIFO
  - can perform poorly with respect to response time

What scheduling algorithm(s) can help minimize response time?

January 11, 2017 | TCSS422: Operating Systems [Fall 2016] Institute of Technology, University of Washington - Tacoma | L3.34

RR: ROUND ROBIN



- Run each job awhile, then switch to another distributing the CPU evenly (fairly)
- Scheduling Quantum is called a time slice
- Time ... a mu... timer interrupt period.

RR is fair, but performs poorly on metrics such as turnaround time

Process	Burst Time
P1	12
P5	5

Round Robin scheduling algorithm Gantt chart

P1	P2	P3	P4	P5	P1	P2	P4	P1

0 5 10 14 19 24 29 32 37 39

Scheduling Quantum = 5 seconds

January 11, 2017 | TCSS422: Operating Systems [Fall 2016] Institute of Technology, University of Washington - Tacoma | L3.35

QUESTIONS

January 11, 2017 | TCSS422: Operating Systems [Winter 2017] Institute of Technology, University of Washington - Tacoma | L3.36