


TCCS 422: OPERATING SYSTEMS

Operating Systems – Three Easy Pieces & Processes

Wes J. Lloyd
 School of Engineering and Technology
 University of Washington - Tacoma



April 1, 2021 TCCS422: Operating Systems [Spring 2021]
 School of Engineering and Technology, University of Washington - Tacoma

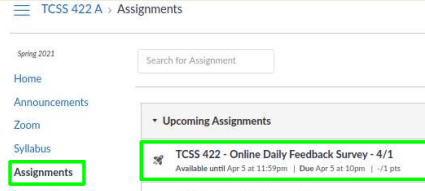
OBJECTIVES – 4/1

- **Questions from 3/30**
- C Review Survey
- Student Background Survey
- Virtual Machine Survey
- Chapter 2: Operating Systems – Three Easy Pieces
 - Concepts of virtualization/abstraction
 - Three Easy Pieces: CPU, Memory, I/O
 - Concurrency
 - Operating system design goals
- Chapter 4: Processes
 - Process states, context switches
 - Kernel data structures for processes and threads

April 1, 2021 TCCS422: Operating Systems [Spring 2021]
 School of Engineering and Technology, University of Washington - Tacoma L2.2

ONLINE DAILY FEEDBACK SURVEY

- Daily Feedback Quiz in Canvas – Available After Each Class
- Extra credit available for completing surveys **ON TIME**
- Tuesday surveys: due by ~ Wed @ 11:59p
- Thursday surveys: due ~ Mon @ 11:59p



April 1, 2021 TCCS422: Computer Operating Systems [Spring 2021]
 School of Engineering and Technology, University of Washington - Tacoma L2.3

TCCS 422 - Online Daily Feedback Survey - 4/1

Quiz Instructions

Question 1 0.5 pts

On a scale of 1 to 10, please classify your perspective on material covered in today's class:

1	2	3	4	5	6	7	8	9	10
Mostly Review To Me			Equal New and Review				Mostly New To Me		

Question 2 0.5 pts

Please rate the pace of today's class:

1	2	3	4	5	6	7	8	9	10
slow	just right				fast				

April 1, 2021 TCCS422: Computer Operating Systems [Spring 2021]
 School of Engineering and Technology, University of Washington - Tacoma L2.4

MATERIAL / PACE

- Please classify your perspective on material covered in today's class (63 respondents):
- 1-mostly review, 5-equal new/review, 10-mostly new
- **Average – 5.59 (no previous trend yet)**
- Please rate the pace of today's class:
- 1-slow, 5-just right, 10-fast
- **Average – 5.33 (no previous trend yet)**

April 1, 2021 TCCS422: Computer Operating Systems [Spring 2021]
 School of Engineering and Technology, University of Washington - Tacoma L2.5

FEEDBACK


April 1, 2021 TCCS422: Operating Systems [Spring 2021]
 School of Engineering and Technology, University of Washington - Tacoma L2.6

OBJECTIVES – 4/1

- Questions from 3/30
- **C Review Survey**
- Student Background Survey
- Virtual Machine Survey
- Chapter 2: Operating Systems – Three Easy Pieces
 - Concepts of virtualization/abstraction
 - Three Easy Pieces: CPU, Memory, I/O
 - Concurrency
 - Operating system design goals
- Chapter 4: Processes
 - Process states, context switches
 - Kernel data structures for processes and threads

April 1, 2021	TCCS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma	L2.7
---------------	---	------

C REVIEW SURVEY



April 1, 2021	TCCS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma	L2.8
---------------	---	------

OBJECTIVES – 4/1

- Questions from 3/30
- C Review Survey
- **Student Background Survey**
- Virtual Machine Survey
- Chapter 2: Operating Systems – Three Easy Pieces
 - Concepts of virtualization/abstraction
 - Three Easy Pieces: CPU, Memory, I/O
 - Concurrency
 - Operating system design goals
- Chapter 4: Processes
 - Process states, context switches
 - Kernel data structures for processes and threads

April 1, 2021	TCCS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma	L2.9
---------------	---	------

STUDENT BACKGROUND SURVEY

- Please complete the Student Background Survey
- <https://forms.gle/yr6Dc9x9rX516U6t6>

April 1, 2021	TCCS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma	L2.10
---------------	---	-------

OBJECTIVES – 4/1

- Questions from 3/30
- C Review Survey
- Student Background Survey
- **Virtual Machine Survey**
- Chapter 2: Operating Systems – Three Easy Pieces
 - Concepts of virtualization/abstraction
 - Three Easy Pieces: CPU, Memory, I/O
 - Concurrency
 - Operating system design goals
- Chapter 4: Processes
 - Process states, context switches
 - Kernel data structures for processes and threads

April 1, 2021	TCCS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma	L2.11
---------------	---	-------

VIRTUAL MACHINE SURVEY

- Please complete the Virtual Machine Survey to request a "School of Engineering and Technology" remote hosted Ubuntu VM
- <https://forms.gle/BR2G1wr9RDBVB9AK8>

April 1, 2021	TCCS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma	L2.12
---------------	---	-------

OBJECTIVES – 4/1

- Questions from 3/30
- C Review Survey
- Student Background Survey
- Virtual Machine Survey
- Chapter 2: Operating Systems – Three Easy Pieces
 - **Concepts of virtualization/abstraction**
 - Three Easy Pieces: CPU, Memory, I/O
 - Concurrency
 - Operating system design goals
- Chapter 4: Processes
 - Process states, context switches
 - Kernel data structures for processes and threads

April 1, 2021 TCCS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma L2.13

ABSTRACTIONS

- What form of abstraction does the OS provide?
 - CPU
 - Process and/or thread
 - Memory
 - Address space
 - → large array of bytes
 - All programs see the same “size” of RAM
 - Disk
 - Files

April 1, 2021 TCCS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma L2.14

WHY ABSTRACTION?

- Allow applications to reuse common facilities
- Make different devices look the same
 - Easier to write common code to use devices
 - Linux/Unix Block Devices
- Provide higher level abstractions
- More useful functionality

April 1, 2021 TCCS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma L2.15

ABSTRACTION CHALLENGES

- What level of abstraction?
 - How much of the underlying hardware should be exposed?
 - What if **too much**?
 - What if **too little**?
- What are the correct abstractions?
 - Security concerns

April 1, 2021 TCCS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma L2.16

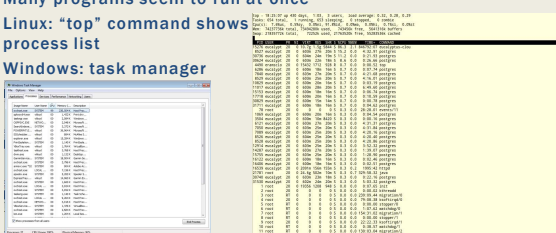
OBJECTIVES – 4/1

- Questions from 3/30
- C Review Survey
- Student Background Survey
- Virtual Machine Survey
- Chapter 2: Operating Systems – Three Easy Pieces
 - Concepts of virtualization/abstraction
 - **Three Easy Pieces: CPU** Memory, I/O
 - Concurrency
 - Operating system design goals
- Chapter 4: Processes
 - Process states, context switches
 - Kernel data structures for processes and threads

April 1, 2021 TCCS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma L2.17

VIRTUALIZING THE CPU

- Each running program gets its own “virtual” representation of the CPU
- Many programs seem to run at once
- Linux: “top” command shows process list
- Windows: task manager



April 1, 2021 TCCS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma L2.18

VIRTUALIZING THE CPU - 2

■ Simple Looping C Program

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <sys/time.h>
4  #include <assert.h>
5  #include "common.h"
6
7  int
8  main(int argc, char *argv[])
9  {
10     if (argc != 2) {
11         fprintf(stderr, "usage: cpu <string>\n");
12         exit(1);
13     }
14     char *str = argv[1];
15     while (1) {
16         Spin(1); // Repeatedly checks the time and
17                 // returns once it has run for a second
18         printf("%s\n", str);
19     }
20     return 0;
    
```

April 1, 2021 TCCS422: Operating Systems [Spring 2021] L2.19
 School of Engineering and Technology, University of Washington - Tacoma

VIRTUALIZING THE CPU - 3

```

prompt> gcc -o cpu cpu.c -Wall
prompt> ./cpu "A"
A
A
A
^C
prompt>
    
```

■ Runs forever, must Ctrl-C to halt...

April 1, 2021 TCCS422: Operating Systems [Spring 2021] L2.20
 School of Engineering and Technology, University of Washington - Tacoma

VIRTUALIZATION THE CPU - 4

```

prompt> ./cpu A & ; ./cpu B & ; ./cpu C & ; ./cpu D &
[1] 7353
[2] 7354
[3] 7355
[4] 7356
A
B
D
C
A
B
D
C
A
C
B
D
...
    
```

Even though we have only one processor all four instances of our program seem to be running at the same time!

April 1, 2021 TCCS422: Operating Systems [Spring 2021] L2.21
 School of Engineering and Technology, University of Washington - Tacoma

MANAGING PROCESSES FROM THE CLI

- & - run a job in the background
- fg - brings a job to the foreground
- bg - sends a job to the background
- CTRL-Z to suspend a job
- CTRL-C to kill a job
- "jobs" command - lists running jobs
- "jobs -p" command - lists running jobs by process ID

- top -d .2 top utility shows active running jobs like the Windows task manager
- top -H -d .2 display all processes & threads
- top -H -p <pid> display all threads of a process
- htop alternative to top, shows CPU core graphs

April 1, 2021 TCCS422: Operating Systems [Spring 2021] L2.22
 School of Engineering and Technology, University of Washington - Tacoma

OBJECTIVES – 4/1

- Questions from 3/30
- C Review Survey
- Student Background Survey
- Virtual Machine Survey
- Chapter 2: Operating Systems – Three Easy Pieces
 - Concepts of virtualization/abstraction
 - Three Easy Pieces: CPU, **Memory**, I/O
 - Concurrency
 - Operating system design goals
- Chapter 4: Processes
 - Process states, context switches
 - Kernel data structures for processes and threads

April 1, 2021 TCCS422: Operating Systems [Spring 2021] L2.23
 School of Engineering and Technology, University of Washington - Tacoma

VIRTUALIZING MEMORY

- Computer memory is treated as a large array of bytes
- Programs store all data in this large array
 - Read memory (load)
 - Specify an address to read data from
 - Write memory (store)
 - Specify data to write to an address

April 1, 2021 TCCS422: Operating Systems [Spring 2021] L2.24
 School of Engineering and Technology, University of Washington - Tacoma

VIRTUALIZING MEMORY - 2

- Program to read/write memory:

```

1  #include <unistd.h>
2  #include <stdio.h>
3  #include <stdlib.h>
4  #include "common.h"
5
6  int
7  main(int argc, char *argv[])
8  {
9      int *p = malloc(sizeof(int)); // a1: allocate some
                                // memory
10     assert(p != NULL);
11     printf("(%d) address of p: %08x\n",
12            getpid(), (unsigned) p); // a2: print out the
                                // address of the memory
13     *p = 0; // a3: put zero into the first slot of the memory
14     while (1) {
15         Spin(1);
16         *p = *p + 1;
17         printf("(%d) p: %d\n", getpid(), *p); // a4
18     }
19     return 0;
20 }
    
```

April 1, 2021 TCCS422: Operating Systems [Spring 2021]
 School of Engineering and Technology, University of Washington - Tacoma L2.25

VIRTUALIZING MEMORY - 3

- Output of mem.c

```

prompt> ./mem
(2134) memory address of p: 00200000
(2134) p: 1
(2134) p: 2
(2134) p: 3
(2134) p: 4
(2134) p: 5
^C
    
```

- int value stored at 00200000
- program increments int value

April 1, 2021 TCCS422: Operating Systems [Spring 2021]
 School of Engineering and Technology, University of Washington - Tacoma L2.26

VIRTUALIZING MEMORY - 4

- Multiple instances of mem.c

```

prompt> ./mem & ./mem &
[1] 24113
[2] 24114
(24113) memory address of p: 00200000
(24114) memory address of p: 00200000
(24113) p: 1
(24114) p: 1
(24114) p: 2
(24113) p: 2
(24113) p: 3
(24114) p: 3
...
    
```

- (int*)p receives the same memory location 00200000
- Why does modifying (int*)p in program #1 (PID=24113), not interfere with (int*)p in program #2 (PID=24114) ?
 - The OS has "virtualized" memory, and provides a "virtual" address

April 1, 2021 TCCS422: Operating Systems [Spring 2021]
 School of Engineering and Technology, University of Washington - Tacoma L2.27

VIRTUAL MEMORY

- Key take-aways:
- Each process (program) has its own **virtual address space**
- The OS maps virtual **address spaces** onto **physical memory**
- A memory reference from one process can not affect the address space of others.
 - Isolation**
- Physical memory, a **shared resource**, is managed by the OS

April 1, 2021 TCCS422: Operating Systems [Spring 2021]
 School of Engineering and Technology, University of Washington - Tacoma L2.28

OBJECTIVES – 4/1

- Questions from 3/30
- C Review Survey
- Student Background Survey
- Virtual Machine Survey
- Chapter 2: Operating Systems – Three Easy Pieces
 - Concepts of virtualization/abstraction
 - Three Easy Pieces: CPU, Memory, I/O**
 - Concurrency
 - Operating system design goals
- Chapter 4: Processes
 - Process states, context switches
 - Kernel data structures for processes and threads

April 1, 2021 TCCS422: Operating Systems [Spring 2021]
 School of Engineering and Technology, University of Washington - Tacoma L2.29

PERSISTENCE

- DRAM: Dynamic Random Access Memory: DIMMs/SIMMs
 - Stores data while power is present
 - When power is lost, data is lost (*volatile*)
- Operating System helps "persist" data more **permanently**
 - I/O device(s): hard disk drive (HDD), solid state drive (SSD)
 - File system(s): "catalog" data for storage and retrieval

April 1, 2021 TCCS422: Operating Systems [Spring 2021]
 School of Engineering and Technology, University of Washington - Tacoma L2.30

PERSISTENCE - 2

```

1  #include <stdio.h>
2  #include <unistd.h>
3  #include <assert.h>
4  #include <fcntl.h>
5  #include <sys/types.h>
6
7  int
8  main(int argc, char *argv[])
9  {
10     int fd = open("/tmp/file", O_WRONLY | O_CREAT
11                | O_TRUNC, S_IRWXU);
12     assert(fd > -1);
13     int rc = write(fd, "hello world\n", 13);
14     assert(rc == 13);
15     close(fd);
16     return 0;
    }
    
```

- open(), write(), close(): OS system calls for device I/O
- Note: man page for open(), write() require page number: "man 2 open", "man 2 write", "man close"

April 1, 2021
TCCS422: Operating Systems [Spring 2021]
 School of Engineering and Technology, University of Washington - Tacoma
L2.31

PERSISTENCE - 3

- To write to disk, OS must:
 - Determine where on disk data should reside
 - Perform sys calls to perform I/O:
 - Read/write to file system (inode record)
 - Read/write data to file
- Provide fault tolerance for system crashes
 - Journaling: Record disk operations in a journal for replay
 - Copy-on-write - replicating shared data - see ZFS
 - Carefully order writes on disk

April 1, 2021
TCCS422: Operating Systems [Spring 2021]
 School of Engineering and Technology, University of Washington - Tacoma
L2.32

OBJECTIVES - 4/1

- Questions from 3/30
- C Review Survey
- Student Background Survey
- Virtual Machine Survey
- Chapter 2: Operating Systems – Three Easy Pieces
 - Concepts of virtualization/abstraction
 - Three Easy Pieces: CPU, Memory, I/O
 - Concurrency**
 - Operating system design goals
- Chapter 4: Processes
 - Process states, context switches
 - Kernel data structures for processes and threads

April 1, 2021
TCCS422: Operating Systems [Spring 2021]
 School of Engineering and Technology, University of Washington - Tacoma
L2.33

CONCURRENCY

The screenshot shows the Windows Task Manager window with the 'Processes' tab selected. It displays a list of running processes including System Idle Process, System, smss.exe, csrss.exe, explorer.exe, and various system services. The CPU usage is 100% and Physical Memory usage is 36%.

April 1, 2021
TCCS422: Operating Systems [Spring 2021]
 School of Engineering and Technology, University of Washington - Tacoma
L2.34

CONCURRENCY

- Linux: 654 tasks
- Windows: 37 processes
- The OS appears to run many programs at once, juggling them
- Modern multi-threaded programs feature concurrent threads and processes
- What is a key difference between a process and a thread?**

April 1, 2021
TCCS422: Operating Systems [Spring 2021]
 School of Engineering and Technology, University of Washington - Tacoma
L2.35

CONCURRENCY - 2

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include "common.h"
4
5  volatile int counter = 0;
6  int loops;
7
8  void *worker(void *arg) {
9      int i;
10     for (i = 0; i < loops; i++) {
11         counter++;
12     }
13     return NULL;
14 }
15 ...
    
```

thread.c

Listing continues ...

April 1, 2021
TCCS422: Operating Systems [Spring 2021]
 School of Engineering and Technology, University of Washington - Tacoma
L2.36

CONCURRENCY - 2

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include "common.h"
4
5  volatile int counter = 0;
6  int loops;
7
8  void
9
10
11
12
13 }
14 }
15 ...
    
```

Not the same as Java volatile:
 Provides a compiler hint that an object may change value unexpectedly (in this case by a separate thread) so aggressive optimization must be avoided.

thread.c

Listing continues ...

April 1, 2021	TCCS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma	L2.37
---------------	---	-------

CONCURRENCY - 3

```

16  int
17  main(int argc, char *argv[])
18  {
19      if (argc != 2) {
20          fprintf(stderr, "usage: threads <value>\n");
21          exit(1);
22      }
23      loops = atoi(argv[1]);
24      pthread_t p1, p2;
25      printf("Initial value : %d\n", counter);
26
27      pthread_create(&p1, NULL, worker, NULL);
28      pthread_create(&p2, NULL, worker, NULL);
29      pthread_join(p1, NULL);
30      pthread_join(p2, NULL);
31      printf("Final value : %d\n", counter);
32      return 0;
33  }
    
```

- Program creates two threads
- Check documentation: "man pthread_create"
- worker() method counts from 0 to argv[1] (loop)

April 1, 2021	TCCS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma	L2.38
---------------	---	-------

Linux
"man"
page
example

```

PTHREAD_CREATE(3)  Linux Programmer's Manual  PTHREAD_CREATE(3)
NAME
  pthread_create - create a new thread
SYNOPSIS
  #include <pthread.h>
  int pthread_create(pthread_t *thread, const pthread_attr_t *attr,
  void *(*start_routine)(void *), void *arg);
  Compile and link with -pthread.
DESCRIPTION
  The pthread_create() function starts a new thread in the calling process. The new thread starts execution by invoking start_routine(). arg is passed as the sole argument of start_routine().
  The new thread terminates in one of the following ways:
  * It calls pthread_exit(), specifying an exit status value that is available to another thread in the same process that calls pthread_join().
  * It returns from start_routine(). This is equivalent to calling pthread_exit() with the value supplied in the return statement.
  * It is canceled (see pthread_cancel(3)).
  * Any of the threads in the process calls exit(3), or the main thread performs a return from main(). This causes the termination of all threads in the process.
  The attr argument points to a pthread_attr_t structure whose contents are used at thread creation time to determine attributes for the new thread: this structure is initialized using pthread_attr_init() and related functions. If attr is NULL, then the thread is created with default attributes.
            
```

April 1, 2021	TCCS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma	L2.39
---------------	---	-------

CONCURRENCY - 4

- Command line parameter argv[1] provides loop length
- Defines number of times the shared counter is incremented
- Loops: 1000


```

prompt> gcc -o thread thread.c -Wall -pthread
prompt> ./thread 1000
Initial value : 0
Final value : 2000
    
```

- Loops 100000

```

prompt> ./thread 100000
Initial value : 0
Final value : 143012 // huh??
prompt> ./thread 100000
Initial value : 0
Final value : 137298 // what the??
    
```



April 1, 2021	TCCS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma	L2.40
---------------	---	-------

CONCURRENCY - 5

- When loop value is large why do we not achieve 200000 ?
- C code is translated to (3) assembly code operations
 1. Load counter variable into register
 2. Increment it
 3. Store the register value back in memory
- These instructions happen concurrently and VERY FAST
- (P1 || P2) write incremented register values back to memory, While (P1 || P2) read same memory
- Memory access here is **unsynchronized (non-atomic)**
- Some of the increments are lost

April 1, 2021	TCCS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma	L2.41
---------------	---	-------

W To perform parallel work, a single process may:

A

Launch multiple threads to execute code in parallel while sharing global data in memory

B

Launch multiple processes to execute code in parallel while sharing global data in memory

C

Both A and B

D

None of the above

April 1, 2021	TCCS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma	L2.42
---------------	---	-------

PARALLEL PROGRAMMING

- To perform parallel work, a single process may:
 - A. Launch multiple threads to execute code in parallel while sharing global data in memory
 - B. Launch multiple processes to execute code in parallel without sharing global data in memory
 - C. Both A and B
 - D. None of the above

April 1, 2021	TCCS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma	L2.43
---------------	---	-------

OBJECTIVES – 4/1

- Questions from 3/30
- C Review Survey
- Student Background Survey
- Virtual Machine Survey
- Chapter 2: Operating Systems – Three Easy Pieces
 - Concepts of virtualization/abstraction
 - Three Easy Pieces: CPU, Memory, I/O
 - Concurrency
 - **Operating system design goals**
- Chapter 4: Processes
 - Process states, context switches
 - Kernel data structures for processes and threads

April 1, 2021	TCCS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma	L2.44
---------------	---	-------

SUMMARY: OPERATING SYSTEM DESIGN GOALS


- **ABSTRACTING THE HARDWARE**
 - Makes programming code easier to write
 - Automate sharing resources – save programmer burden
- **PROVIDE HIGH PERFORMANCE**
 - Minimize overhead from OS abstraction (Virtualization of CPU, RAM, I/O)
 - Share resources fairly
 - Attempt to tradeoff performance vs. fairness → consider priority
- **PROVIDE ISOLATION**
 - User programs can't interfere with each other's virtual machines, the underlying OS, or the sharing of resources

April 1, 2021	TCCS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma	L2.45
---------------	---	-------

SUMMARY: OPERATING SYSTEM DESIGN GOALS - 2

- **RELIABILITY**
 - OS must not crash, 24/7 Up-time
 - Poor user programs must not bring down the system:

Blue Screen



- Other Issues:
 - Energy-efficiency
 - Security (of data)
 - Cloud: Virtual Machines

April 1, 2021	TCCS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma	L2.46
---------------	---	-------

WE WILL RETURN AT 5:00PM



April 1, 2021	TCCS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma	L2.47
---------------	---	-------

OBJECTIVES – 4/1

- Questions from 3/30
- C Review Survey
- Student Background Survey
- Virtual Machine Survey
- Chapter 2: Operating Systems – Three Easy Pieces
 - Concepts of virtualization/abstraction
 - Three Easy Pieces: CPU, Memory, I/O
 - Concurrency
 - Operating system design goals
- **Chapter 4: Processes**
 - Process states, context switches
 - Kernel data structures for processes and threads

April 1, 2021	TCCS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma	L2.48
---------------	---	-------

CHAPTER 4: PROCESSES

Process State

/proc

April 1, 2021 TCCS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma L2.49

VIRTUALIZING THE CPU

- How should the CPU be shared?
- Time Sharing:
Run one process, pause it, run another
- The act of swapping process A out of the CPU to run process B is called a:
 - **CONTEXT SWITCH**
- How do we SWAP processes in and out of the CPU efficiently?
 - Goal is to minimize **overhead** of the swap
- **OVERHEAD** is time spent performing OS management activities that don't help accomplish real work

April 1, 2021 TCCS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma L2.50

PROCESS

A process is a running program.

- Process comprises of:
 - Memory
 - Instructions ("the code")
 - Data (heap)
 - Registers
 - PC: Program counter
 - Stack pointer

April 1, 2021 TCCS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma L2.51

PROCESS API

- Modern Oses provide a Process API for process support
- Create
 - Create a new process
- Destroy
 - Terminate a process (ctrl-c)
- Wait
 - Wait for a process to complete/stop
- Miscellaneous Control
 - Suspend process (ctrl-z)
 - Resume process (fg, bg)
- Status
 - Obtain process statistics: (top)

April 1, 2021 TCCS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma L2.52

PROCESS API: CREATE

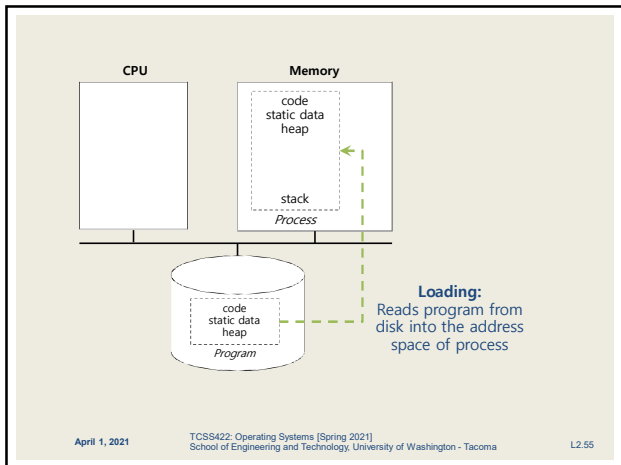
1. Load program code (and static data) into memory
 - Program executable code (binary): loaded from disk
 - Static data: also loaded/created in address space
 - **Eager loading:** Load entire program before running
 - **Lazy loading:** Only load what is immediately needed
 - Modern Oses: Supports paging & swapping
2. Run-time stack creation
 - Stack: local variables, function params, return address(es)

April 1, 2021 TCCS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma L2.53

PROCESS API: CREATE

3. Create program's heap memory
 - For dynamically allocated data
4. Other initialization
 - I/O Setup
 - Each process has three open file descriptors: Standard Input, Standard Output, Standard Error
5. Start program running at the entry point: `main()`
 - OS transfers CPU control to the new process

April 1, 2021 TCCS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma L2.54



OBJECTIVES – 4/1

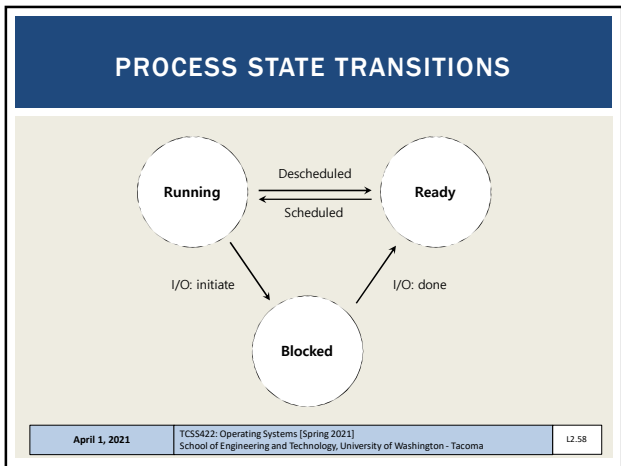
- Questions from 3/30
- C Review Survey
- Student Background Survey
- Virtual Machine Survey
- Chapter 2: Operating Systems – Three Easy Pieces
 - Concepts of virtualization/abstraction
 - Three Easy Pieces: CPU, Memory, I/O
 - Concurrency
 - Operating system design goals
- Chapter 4: Processes
 - **Process states, context switches**
 - Kernel data structures for processes and threads

April 1, 2021 TCCS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma L2.56

PROCESS STATES

- **RUNNING**
 - Currently executing instructions
- **READY**
 - Process is ready to run, but has been preempted
 - CPU is presently allocated for other tasks
- **BLOCKED**
 - Process is **not** ready to run. It is waiting for another event to complete:
 - Process has already been initialized and run for awhile
 - Is now waiting on I/O from disk(s) or other devices

April 1, 2021 TCCS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma L2.57



OBSERVING PROCESS META-DATA

- Can inspect the number of **CONTEXT SWITCHES** made by a process
- Let's run mem.c (from chapter 2)
- `cat /proc/{process-id}/status`

```

Speculation_Store_Bypass: thread vulnerable
Cpus_allowed: ff
Cpus_allowed_list: 0-7
Mems_allowed: 00000000,00000001
Mems_allowed_nodevec: 00000000
Voluntary_ctxt_switches: 1372
Nonvoluntary_ctxt_switches: 18
    
```

- `proc "status"` is a virtual file generated by Linux
- Provides a report with process related meta-data
- **What appears to happen to the number of context switches the longer a process runs? (mem.c)**

April 2, 2020 TCCS422: Operating Systems [Spring 2020] School of Engineering and Technology, University of Washington - Tacoma L2.59

CONTEXT SWITCH

- **How long does a context switch take?**
- 10,000 to 50,000 ns (.01 to .05 ms)
- 2,000 context switches is near 100ms

Without CPU affinity

April 2, 2020 TCCS422: Operating Systems [Spring 2020] School of Engineering and Technology, University of Washington - Tacoma L2.60

W When a process is in this state, it is advantageous for the Operating System to perform a CONTEXT SWITCH to perform other work

RUNNING READY BLOCKED All of the above None of the above

April 1, 2021 TCCS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma L2.62

QUESTION: WHEN TO CONTEXT SWITCH

- When a process is in this state, it is advantageous for the Operating System to perform a CONTEXT SWITCH to perform other work:
 - (a) RUNNING
 - (b) READY
 - (c) BLOCKED
 - (d) All of the above
 - (e) None of the above

April 1, 2021 TCCS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma L2.62

OBJECTIVES – 4/1

- Questions from 3/30
- C Review Survey
- Student Background Survey
- Virtual Machine Survey
- Chapter 2: Operating Systems – Three Easy Pieces
 - Concepts of virtualization/abstraction
 - Three Easy Pieces: CPU, Memory, I/O
 - Concurrency
 - Operating system design goals
- Chapter 4: Processes
 - Process states, context switches
 - Kernel data structures for processes and threads**

April 1, 2021 TCCS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma L2.63

PROCESS DATA STRUCTURES

- OS provides data structures to track process information
 - Process list
 - Process Data
 - State of process: Ready, Blocked, Running
 - Register context
- PCB (Process Control Block)
 - A C-structure that contains information about each process

April 1, 2021 TCCS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma L2.64

XV6 KERNEL DATA STRUCTURES

- xv6: pedagogical implementation of Linux
- Simplified structures shown in book

```

// the registers xv6 will save and restore
// to stop and subsequently restart a process
struct context {
    int eip; // Index pointer register
    int esp; // Stack pointer register
    int ebx; // Called the base register
    int ecx; // Called the counter register
    int edx; // Called the data register
    int esi; // Source index register
    int edi; // Destination index register
    int ebp; // Stack base pointer register
};

// the different states a process can be in
enum proc_state { UNUSED, EMBRYO, SLEEPING,
    RUNNABLE, RUNNING, ZOMBIE };
    
```

April 1, 2021 TCCS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma L2.65

XV6 KERNEL DATA STRUCTURES - 2

```

// the information xv6 tracks about each process
// including its register context and state
struct proc {
    char *mem; // Start of process memory
    uint sz; // Size of process memory
    char *kstack; // Bottom of kernel stack
    // for this process
    enum proc_state state; // Process state
    int pid; // Process ID
    struct proc *parent; // Parent process
    void *chan; // If non-zero, sleeping on chan
    int killed; // If non-zero, have been killed
    struct file *ofile[NOFILE]; // Open files
    struct inode *cwd; // Current directory
    struct context context; // Switch here to run process
    struct trapframe *tf; // Trap frame for the
    // current interrupt
};
    
```

April 1, 2021 TCCS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma L2.66

LINUX: STRUCTURES

- `struct task_struct`, equivalent to struct proc
 - The Linux process data structure
 - **VERY LARGE: 10,000+ bytes**
 - `/usr/src/linux-headers-{kernel version}/include/linux/sched.h`
 - - LOC 1391 - 1852 (4.4.0-170)
 - `task_struct` originally stored in the kernel's stack space
 - Limited to 2 x 4KB pages = 8 KB
 - `task_struct` is LARGE, has been moved outside the kernel stack
 - The smaller `thread_info` struct is now stored on the kernel's stack & provides a ptr to `task_struct` allocated using the slab allocator
 - Slab allocator allocates memory for common data structures in Linux
- `struct thread_info`, provides ptr to `task_struct`
 - `thread_info.h` is at:
 - `/usr/src/linux-headers-{kernel version}/arch/x86/include/asm/`

April 1, 2021
TCCS422: Operating Systems [Spring 2021]
School of Engineering and Technology, University of Washington - Tacoma
L2.67

LINUX: THREAD_INFO

```

struct thread_info {
    struct task_struct *task;          /* main task structure */
    struct exec_domain *exec_domain; /* execution domain */
    __u32 flags;                       /* low level flags */
    __u32 status;                       /* thread synchronous flags */
    __u32 cpu;                          /* current CPU */
    int preempt_count;                 /* 0 => preemptable,
                                        <0 => BUG */

    mm_segment_t addr_limit;
    struct restart_block restart_block;
    void __user *sysenter_return;

#ifdef CONFIG_X86_32
    unsigned long previous_esp; /* ESP of the previous stack in
                                case of nested (IRQ) stacks
                                */
    __u8 supervisor_stack[0];
#endif
    int uaccess_err;
};
    
```


April 1, 2021
TCCS422: Operating Systems [Spring 2021]
School of Engineering and Technology, University of Washington - Tacoma
L2.68

LINUX STRUCTURES - 2


- List of Linux data structures:
 - <http://www.tldp.org/LDP/tlk/ds/ds.html>
- Description of process data structures:
 - <https://learning.oreilly.com/library/view/linux-kernel-development/9780768696974/cover.html>
 - 3rd edition is online (dated from 2010):
 - See chapter 3 on Process Management
 - Safari online – accessible using UW ID SSO login
 - Linux Kernel Development, 3rd edition
 - Robert Love
 - Addison-Wesley

April 1, 2021
TCCS422: Operating Systems [Spring 2021]
School of Engineering and Technology, University of Washington - Tacoma
L2.69

QUESTIONS



QUESTIONS



TCCS 422

OFFICE HOURS

PLEASE SAY HELLO





OFFICE HOURS
HAVE STEPPED OUT
WILL RETURN
SHORTLY

The slide features a blue background with white text. On the right side, there is a vertical strip containing a small photograph of an office desk with a computer monitor and keyboard.