


TCSS 422: OPERATING SYSTEMS

Operating Systems – Three Easy Pieces & Processes



Wes J. Lloyd

School of Engineering and Technology

University of Washington - Tacoma

March 30, 2023

TCSS422: Operating Systems [Spring 2023]
School of Engineering and Technology, University of Washington - Tacoma

L2.1

1

OBJECTIVES – 3/30

■ Questions from 3/28

■ C Review Survey - available thru 4/7

■ 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

March 30, 2023

TCSS422: Operating Systems [Spring 2023]
School of Engineering and Technology, University of Washington - Tacoma

L2.2

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 @ 9p, cutoff 11:59p

■ Thursday surveys: due ~ Mon @ 9p, cutoff 11:59p

TCSS 422 A > Assignments

Spring 2023

Home

Announcements

Zoom

Syllabus

Assignments

Discussions

Upcoming Assignments

TCSS 422 - Online Daily Feedback Survey - 4/1

Available until Apr 3 at 11:59pm | Due Apr 3 at 10pm | /15 pts

March 30, 2023

TCSS422: Computer Operating Systems [Spring 2023]
School of Engineering and Technology, University of Washington - Tacoma

L2.3

3

TCSS 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

March 30, 2023

TCSS422: Computer Operating Systems [Spring 2023]
School of Engineering and Technology, University of Washington - Tacoma

L2.4

4

MATERIAL / PACE

■ Please classify your perspective on material covered in today's class (48 respondents):

■ 1-mostly review, 5-equal new/review, 10-mostly new

■ **Average – 6.18 (fall 2021, 5.64)**

■ Please rate the pace of today's class:

■ 1-slow, 5-just right, 10-fast

■ **Average – 5.91 (fall 2021, 5.38)**

March 30, 2023

TCSS422: Computer Operating Systems [Spring 2023]
School of Engineering and Technology, University of Washington - Tacoma

L2.5

5

FFEEDBACK FROM 3/28

■ **The point I'm the least clear on is the concept of virtualization, like what it means for memory to be virtually represented.**

■ The type of **virtualization** we talk about in operating systems is different than **virtualization** like with a virtual machine.

■ For OSes, virtualization refers to the abstractions or interfaces provided to programmers to interface with the CPU, memory, and I/O devices – **no access is direct – everything goes through OS**

- **CPU:** processes and thread constructs
 - Programmer creates processes and threads using language specific APIs to distribute work of a user program as needed
- **MEM:** virtual memory (accessed using `C malloc()`, Java `new` etc.)
 - All addresses presented to user programs feature virtual addresses that index the large memory array (e.g. 32 GB)
- **I/O:** I/O language specific APIs: `open()`, `close()`, `write()`, etc.
 - APIs interface with the OS kernel to perform I/O in privileged mode
 - **User code does not directly perform I/O operations, it must do so via the OS**

March 30, 2023

TCSS422: Operating Systems [Spring 2023]
School of Engineering and Technology, University of Washington - Tacoma

L2.6

6

FEEDBACK - 2

- **Why do we need the layers of abstraction that an OS provides? What are the specific cases for how the abstractions can benefit us?**
- A key aspect is 'fair' resource sharing
- All computer components must be shared with all programs: CPU, memory, I/O devices (network card, disks, etc.)
- The OS acts like a conductor or director
- The OS ensures that different user programs obtain a 'fair share' of each resource
- The OS ensures that different user programs can see each others data while sharing the CPU, memory, network, disk
- Without the OS only 1 program can 'run' at the same time
- **But how do you share resources fairly without introducing too much overhead (time cost of the abstractions) ?**

March 30, 2023

TCSS422: Operating Systems [Spring 2023]
School of Engineering and Technology, University of Washington - Tacoma

L2.7

7

FEEDBACK - 3

- **What type of algorithms are implemented in the operating system for virtualizing each component?**
- OS algorithms primarily concern resource sharing
- They are 'scheduling algorithms'
- The most classic is probably 'round robin (RR)'
 - Round robin evenly divides the time share of a resources among all users that belong to a "queue" or user group
- Another classic is 'first in first out (FIFO)'
 - FIFO allows the first arriving resource to take the full timeshare of a resource until it finishes. This is similar to 'greedy'.
 - Other variants: last in first out (LIFO), first in last out (FILO)
- Other algorithms may assume some knowledge about a program's required timeshare of a resource

March 30, 2023

TCSS422: Operating Systems [Spring 2023]
School of Engineering and Technology, University of Washington - Tacoma

L2.8

8

FEEDBACK - 4

- **I am not sure about how to identify when more or less abstraction should be used to properly balance performance and security ?**
 - This is an open question. Every OS makes an attempt to balance these trade-offs. There is NO PERFECT BALANCE.
 - Some OSes focus on specific goals, for example REAL-time OS
- **Is there some way this can be achieved generally (i.e., independently of the programs that will be run?)**
 - OSes provide reusable abstractions (processes, threads, virtual memory) to every user program
 - The OS is a program called a kernel, that user program interact with
 - You can see the 'executable' file that is the 'OS kernel'
 - Use the command '**ls -l /boot**'
 - The active OS kernel is pointed to by the 'vmlinuz' file link
 - What is the size of our active OS kernel ?

March 30, 2023

TCSS422: Operating Systems [Spring 2023]
School of Engineering and Technology, University of Washington - Tacoma

L2.9

9

FEEDBACK - 5

- **Virtual memory was still unclear to me**
 - 32 GB computer: Linux indexes memory as large array of **4 KB pages**
 - 32 GB is 2^{35}
 - A 4096 byte (4KB) page is 2^{12}
 - We can divide $2^{35} / 2^{12}$ to calculate the total # of pages
 - Divide by subtracting exponents ($35 - 12 = 23$)
 - Linux indexes 32 GB with 2^{23} 4KB pages = 8,388,608
 - Linux tracks the physical index of every page (0 to 8,388,607)
 - When run hello.c 4 pages: 1 each for the stack, heap, code, and data segments - hello.c will require 16 KB of storage in pages
 - When run, the OS provides hello.c virtual addresses for 4 pages that are located somewhere in the 32GB physical address space
 - The OS translates every virtual address used by hello.c into a physical address on demand as the program is run
 - There are millions of translations!
 - To make it fast – special circuitry is added to CPUs called the TLB

March 30, 2023

TCSS422: Operating Systems [Spring 2023]
School of Engineering and Technology, University of Washington - Tacoma

L2.10

10

RESOURCES

- Textbook coupon 10% off "BOOKFAIR10" until Friday at 11:59pm
- <https://www.lulu.com/shop/andrea-arpaci-dusseau-and-remzi-arpaci-dusseau/operating-systems-three-easy-pieces-softcover-version-100/paperback/product-14mjrrgk.html>
- With coupon textbook is only \$19.80 + tax & shipping

March 30, 2023

TCSS422: Operating Systems [Spring 2023]
School of Engineering and Technology, University of Washington - Tacoma

L2.11

11

OBJECTIVES – 3/30

- Questions from 3/28
- **C Review Survey - available thru 4/7**
- 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

March 30, 2023

TCSS422: Operating Systems [Spring 2023]
School of Engineering and Technology, University of Washington - Tacoma

L2.12

12

C REVIEW SURVEY -
AVAILABLE THRU 4/7



March 30, 2023

TCSS422: Operating Systems [Spring 2023]
School of Engineering and Technology, University of Washington - Tacoma

L2.13

13

OBJECTIVES – 3/30

- Questions from 3/28
- C Review Survey - available thru 4/7
- **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

March 30, 2023

TCSS422: Operating Systems [Spring 2023]
School of Engineering and Technology, University of Washington - Tacoma

L2.14

14

STUDENT BACKGROUND SURVEY

- Please complete the Student Background Survey
- <https://forms.gle/BuJwXPwZpqf6cnTQ9>
- **44 of 59 Responses** as of 3/29 @ ~11pm
- Current Standings:
 - Best Office Hours times so far:
 - Rank #1: Friday 12 – 2pm
 - Rank #2: Tues/Thur before class (12 – 3:30p)
 - Best lecture format:
 - Rank #1: Hybrid synchronous w/ recordings
 - Rank #2: In-person w/ recordings
- Will consider survey results through Mon Apr 3

March 30, 2023

TCSS422: Operating Systems [Spring 2023]
School of Engineering and Technology, University of Washington - Tacoma

L2.15

15

OBJECTIVES – 3/30

- Questions from 3/28
- C Review Survey - available thru 4/7
- 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

March 30, 2023

TCSS422: Operating Systems [Spring 2023]
School of Engineering and Technology, University of Washington - Tacoma

L2.16

16

VIRTUAL MACHINE SURVEY

- Please complete the Virtual Machine Survey to request a “School of Engineering and Technology” remote hosted Ubuntu VM
- <https://forms.gle/V2sg4IW1awvhFx4W8>
- **40 of 59 Responses** as of 3/29 @ ~11pm
- **Will close Wednesday 4/5...**
- **VM requests will be sent to Stephen Rondeau for set up**

March 30, 2023

TCSS422: Operating Systems [Spring 2023]
School of Engineering and Technology, University of Washington - Tacoma

L2.17

17

OBJECTIVES – 3/30

- Questions from 3/28
- C Review Survey - available thru 4/7
- 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

March 30, 2023

TCSS422: Operating Systems [Spring 2023]
School of Engineering and Technology, University of Washington - Tacoma

L2.18

18

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

March 30, 2023

TCSS422: Operating Systems [Spring 2023]
School of Engineering and Technology, University of Washington - Tacoma

L2.19

19

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

March 30, 2023

TCSS422: Operating Systems [Spring 2023]
School of Engineering and Technology, University of Washington - Tacoma

L2.20

20

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

March 30, 2023

TCSS422: Operating Systems [Spring 2023]
School of Engineering and Technology, University of Washington - Tacoma

L2.21

21

OBJECTIVES – 3/30

- Questions from 3/28
- C Review Survey - available thru 4/7
- 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

March 30, 2023

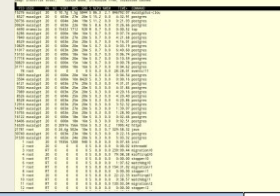
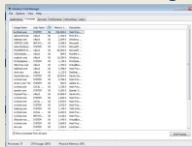
TCSS422: Operating Systems [Spring 2023]
School of Engineering and Technology, University of Washington - Tacoma

L2.22

22

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



March 30, 2023

TCSS422: Operating Systems [Spring 2023]
School of Engineering and Technology, University of Washington - Tacoma

L2.23

23

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 (!) {
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;
21 }
```

March 30, 2023

TCSS422: Operating Systems [Spring 2023]
School of Engineering and Technology, University of Washington - Tacoma

L2.24

24

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...

March 30, 2023

TCSS422: Operating Systems [Spring 2023]
School of Engineering and Technology, University of Washington - Tacoma

L2.25

25

VIRTUALIZATION THE CPU - 4

```
prompt> ./cpu A & 1 ./cpu B & 1 ./cpu C & 1 ./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!

March 30, 2023

TCSS422: Operating Systems [Spring 2023]
School of Engineering and Technology, University of Washington - Tacoma

L2.26

26

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

March 30, 2023

TCSS422: Operating Systems [Spring 2023]
School of Engineering and Technology, University of Washington - Tacoma

L2.27

27

OBJECTIVES - 3/30

- Questions from 3/28
- C Review Survey - available thru 4/7
- 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

March 30, 2023

TCSS422: Operating Systems [Spring 2023]
School of Engineering and Technology, University of Washington - Tacoma

L2.28

28

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

March 30, 2023

TCSS422: Operating Systems [Spring 2023]
School of Engineering and Technology, University of Washington - Tacoma

L2.29

29

VIRTUALIZING MEMORY - 2

Program to read/write memory: (**mem.c**) (from ch. 2 pgs. 5-6)

```
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
10     // memory
11     assert(p != NULL);
12     printf("(1) address of p: %08x\n",
13            getpid(), (unsigned) p); // a2: print out the
14                                     // address of the memory
15     *p = 0; // a3: put zero into the first slot of the memory
16     while (1) {
17         Spin(1);
18         *p = *p + 1;
19         printf("(2) p: %d\n", getpid(), *p); // a4
20     }
21     return 0;
22 }
```

March 30, 2023

TCSS422: Operating Systems [Spring 2023]
School of Engineering and Technology, University of Washington - Tacoma

L2.30

30

VIRTUALIZING MEMORY - 3

- Output of **mem.c** (example from ch. 2 pgs. 5-6)

```
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 virtual address 00200000
- program increments int value pointed to by p

March 30, 2023

TCSS422: Operating Systems [Spring 2023]
School of Engineering and Technology, University of Washington - Tacoma

L2.31

31

VIRTUALIZING MEMORY - 4

- Multiple instances of **mem.c**

By default this example no longer works as advertised!

Ubuntu now applies address space randomization (ASR) by default.

ASR makes the ptr location of program instances not identical. Having identical addresses is considered a security issue.

```
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
...
```

- BOOK SHOWS: (int*)p with the same memory location **00200000**
- To disable ASR: 'echo 0 | tee /proc/sys/kernel/randomize_va_space'
- Why does modifying the value of *p in program #1 (PID 24113), not interfere with the value of *p in program #2 (PID 24114) ?
 - The OS has "virtualized" memory, and provides a "virtual" address

March 30, 2023

TCSS422: Operating Systems [Spring 2023]
School of Engineering and Technology, University of Washington - Tacoma

L2.32

32

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

March 30, 2023

TCSS422: Operating Systems [Spring 2023]
School of Engineering and Technology, University of Washington - Tacoma

L2.33

33

OBJECTIVES – 3/30

- Questions from 3/28
- C Review Survey - available thru 4/7
- 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

March 30, 2023

TCSS422: Operating Systems [Spring 2023]
School of Engineering and Technology, University of Washington - Tacoma

L2.34

34

WHY PERSISTENCE ?

- DRAM: Dynamic Random Access Memory: DIMMs/SIMMs
 - Store data while power is present
 - When power is lost, data is lost (i.e. **volatile memory**)
- 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

March 30, 2023

TCSS422: Operating Systems [Spring 2023]
School of Engineering and Technology, University of Washington - Tacoma

L2.35

35

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;
17 }
```

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

March 30, 2023

TCSS422: Operating Systems [Spring 2023]
School of Engineering and Technology, University of Washington - Tacoma

L2.36

36

PERSISTENCE - 3

- To write to disk, OS must:
 - Determine where on disk data should reside
 - Instrument system calls to perform I/O:
 - Read/write to file system (*inode record*)
 - Read/write data to file
- OS provides fault tolerance for system crashes via special filesystem features:
 - Journaling**: Record disk operations in a journal for replay
 - Copy-on-write**: replicate shared data across multiple disks - see *ZFS filesystem*
 - Carefully order writes on disk (*especially spindle drives*)

March 30, 2023

TCSS422: Operating Systems [Spring 2023]
School of Engineering and Technology, University of Washington - Tacoma

L2.37

37

WE WILL RETURN AT
2:45PM



March 30, 2023

TCSS422: Operating Systems [Spring 2023]
School of Engineering and Technology, University of Washington - Tacoma

L2.38

38

OBJECTIVES – 3/30

- Questions from 3/28
- C Review Survey - available thru 4/7
- 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

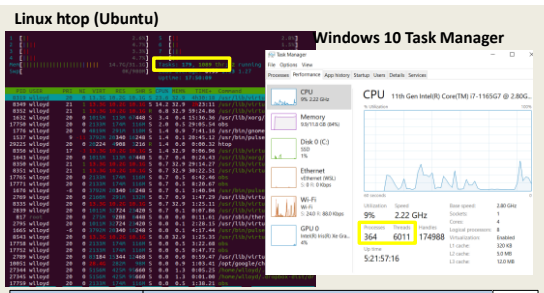
March 30, 2023

TCSS422: Operating Systems [Spring 2023]
School of Engineering and Technology, University of Washington - Tacoma

L2.39

39

CONCURRENCY



March 30, 2023

TCSS422: Operating Systems [Spring 2023]
School of Engineering and Technology, University of Washington - Tacoma

L2.40

40

CONCURRENCY

- Linux: 179 processes, 1089 threads (**htop**)
- Windows 10: 364 processes, 6011 threads (task mgr)
- OSes appear 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?

March 30, 2023

TCSS422: Operating Systems [Spring 2023]
School of Engineering and Technology, University of Washington - Tacoma

L2.41

41

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 ...
```

pthread.c

Listing continues ...

March 30, 2023

TCSS422: Operating Systems [Spring 2023]
School of Engineering and Technology, University of Washington - Tacoma

L2.42

42

CONCURRENCY - 2

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

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

pthread.c

Listing continues ...

March 30, 2023

TCSS422: Operating Systems [Spring 2023]
School of Engineering and Technology, University of Washington - Tacoma

L2.43

43

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)

March 30, 2023

TCSS422: Operating Systems [Spring 2023]
School of Engineering and Technology, University of Washington - Tacoma

L2.44

44

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(3).
* 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(), 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_t_init(3) and
related functions. If attr is NULL, then the thread is created with
default attributes.
```

March 30, 2023

TCSS422: Operating Systems [Spring 2023]
School of Engineering and Technology, University of Washington - Tacoma

L2.45

45

CONCURRENCY - 4

Command line parameter argv[1] provides loop length

Defines number of times the shared counter is incremented

Loops: 1000

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

Loops 100000

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

March 30, 2023

TCSS422: Operating Systems [Spring 2023]
School of Engineering and Technology, University of Washington - Tacoma

L2.46

46

CONCURRENCY - 5

When loop value is large why do we not achieve 200,000 ?

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

March 30, 2023

TCSS422: Operating Systems [Spring 2023]
School of Engineering and Technology, University of Washington - Tacoma

L2.47

47

Activities

Visual settings Edit

When poll is active, respond at PollEv.com/wesleylloyd641

Text WESLEYLOYD641 to 22333 once to join

To perform parallel work, a single process may:

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

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

Both A and B

None of the above

Total Results: 0

Powered by Poll Everywhere

48

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

March 30, 2023

TCSS422: Operating Systems [Spring 2023]
School of Engineering and Technology, University of Washington - Tacoma

L2.49

49

OBJECTIVES – 3/30

- Questions from 3/28
- C Review Survey - available thru 4/7
- 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

March 30, 2023

TCSS422: Operating Systems [Spring 2023]
School of Engineering and Technology, University of Washington - Tacoma

L2.50

50

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

March 30, 2023

TCSS422: Operating Systems [Spring 2023]
School of Engineering and Technology, University of Washington - Tacoma


L2.51

51

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

March 30, 2023

TCSS422: Operating Systems [Spring 2023]
School of Engineering and Technology, University of Washington - Tacoma

L2.52

52

OBJECTIVES – 3/30

- Questions from 3/28
- C Review Survey - available thru 4/7
- 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

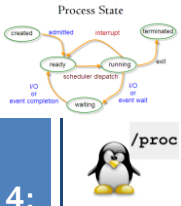
March 30, 2023

TCSS422: Operating Systems [Spring 2023]
School of Engineering and Technology, University of Washington - Tacoma

L2.53

53

CHAPTER 4:
PROCESSES



March 30, 2023

TCSS422: Operating Systems [Spring 2023]
School of Engineering and Technology, University of Washington - Tacoma

L2.54

54

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

March 30, 2023

TCSS422: Operating Systems [Spring 2023]
School of Engineering and Technology, University of Washington - Tacoma

L2.55

55

PROCESS

A process is a running program.

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

March 30, 2023

TCSS422: Operating Systems [Spring 2023]
School of Engineering and Technology, University of Washington - Tacoma

L2.56

56

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)

March 30, 2023

TCSS422: Operating Systems [Spring 2023]
School of Engineering and Technology, University of Washington - Tacoma

L2.57

57

PROCESS API: CREATE

- 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
- Run-time stack creation
 - Stack: local variables, function params, return address(es)

March 30, 2023

TCSS422: Operating Systems [Spring 2023]
School of Engineering and Technology, University of Washington - Tacoma

L2.58

58

PROCESS API: CREATE

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

March 30, 2023

TCSS422: Operating Systems [Spring 2023]
School of Engineering and Technology, University of Washington - Tacoma

L2.59

59

Diagram illustrating the loading of a program into memory:

- CPU**: Represented by a box on the left.
- Memory**: Contains a box for the **Process**, which includes **code static data heap** and **stack**.
- Program**: Represented by a cylinder on the disk.
- Loading**: A dashed arrow points from the **Program** to the **code static data heap** in the **Process** box, indicating the transfer of program code into the address space of the process.

March 30, 2023

TCSS422: Operating Systems [Spring 2023]
School of Engineering and Technology, University of Washington - Tacoma

L2.60

60

OBJECTIVES – 3/30

- Questions from 3/28
- C Review Survey - available thru 4/7
- 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

March 30, 2023

TCSS422: Operating Systems [Spring 2023]
School of Engineering and Technology, University of Washington - Tacoma

L2.61

61

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

March 30, 2023

TCSS422: Operating Systems [Spring 2023]
School of Engineering and Technology, University of Washington - Tacoma

L2.62

62

PROCESS STATE TRANSITIONS

```
graph LR
    Running((Running)) -- Descheduled --> Ready((Ready))
    Ready -- Scheduled --> Running
    Running -- "I/O: initiate" --> Blocked((Blocked))
    Blocked -- "I/O: done" --> Ready
```

March 30, 2023

TCSS422: Operating Systems [Spring 2023]
School of Engineering and Technology, University of Washington - Tacoma

L2.63

63

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

- 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

TCSS422: Operating Systems [Spring 2020]
School of Engineering and Technology, University of Washington - Tacoma

L2.64

64

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

TCSS422: Operating Systems [Spring 2020]
School of Engineering and Technology, University of Washington - Tacoma

L2.65

65

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
- Mileage can vary depending on system conditions, etc.
- See blog:
<https://blog.tsunanet.net/2010/11/how-long-does-it-take-to-make-context.html>

April 2, 2020

TCSS422: Operating Systems [Spring 2020]
School of Engineering and Technology, University of Washington - Tacoma

L2.66

66

Activities

Visual settingsEdit

Respond at PollEv.com/wesleytloyd641

Text WESLEYLOYD641 to 22333 once to join, then 1, 2, 3, 4, or 5

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

RUNNING	1
READY	2
BLOCKED	3
All of the above	4
None of the above	5

Total Results: 0

Powered by

Poll Everywhere

67

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

March 30, 2023

TCCS422: Operating Systems [Spring 2023]
School of Engineering and Technology, University of Washington - Tacoma

L2.68

68

OBJECTIVES – 3/30

■ Questions from 3/28

■ C Review Survey - available thru 4/7

■ 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

March 30, 2023

TCCS422: Operating Systems [Spring 2023]
School of Engineering and Technology, University of Washington - Tacoma

L2.69

69

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

March 30, 2023

TCCS422: Operating Systems [Spring 2023]
School of Engineering and Technology, University of Washington - Tacoma

L2.70

70

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 };

March 30, 2023

TCCS422: Operating Systems [Spring 2023]
School of Engineering and Technology, University of Washington - Tacoma

L2.71

71

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
};

March 30, 2023

TCCS422: Operating Systems [Spring 2023]
School of Engineering and Technology, University of Washington - Tacoma

L2.72

72

LINUX: STRUCTURES

- `struct task_struct`, equivalent to `struct proc`
 - The Linux process data structure
 - Kernel data type (i.e. record) that describes individual Linux processes
 - Structure is VERY LARGE: **10,000+ bytes**
 - Defined in:
 `/usr/src/linux-headers-{kernel version}/include/linux/sched.h`
 - Ubuntu 20.04 w/ kernel version 5.11, **LOC: 657 – 1394**
 - Ubuntu 20.04 w/ kernel version 4.4, **LOC: 1391 – 1852**

March 30, 2023

TCCS422: Operating Systems [Spring 2023]
School of Engineering and Technology, University of Washington - Tacoma

L2.73

73

STRUCT TASK_STRUCT
PROCESS CONTROL BLOCK

- Process Control Block (PCB)
- Key data regarding a process

process state
process number
program counter
registers
memory limits
list of open files
...

March 30, 2023

TCCS422: Operating Systems [Spring 2023]
School of Engineering and Technology, University of Washington - Tacoma

L2.74

74

STRUCT TASK_STRUCT

- Key elements (e.g. PCB) in Linux are captured in `struct task_struct`: (LOC from Linux kernel v 5.11)
- Process ID
- `pid_t pid;` LOC #857
- Process State
- `/* -1 unrunnable, 0 runnable, >0 stopped: */`
- `volatile long state;` LOC #666
- Process time slice
how long the process will run before context switching
- `struct sched_rt_entity` used in `task_struct` contains `timeslice`:
 - `struct sched_rt_entity rt;` LOC #710
 - `unsigned int time_slice;` LOC #503

March 30, 2023

TCCS422: Operating Systems [Spring 2023]
School of Engineering and Technology, University of Washington - Tacoma

L2.75

75

STRUCT TASK_STRUCT - 2

- Address space of the process:
 - "mm" is short for "memory map"
 - `struct mm_struct *mm;` LOC #779
- Parent process, that launched this one
 - `struct task_struct __rcu *parent;` LOC #874
- Child processes (as a list)
 - `struct list_head children;` LOC #879
- Open files
 - `struct files_struct *files;` LOC #981

March 30, 2023

TCCS422: Operating Systems [Spring 2023]
School of Engineering and Technology, University of Washington - Tacoma

L2.76

76

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 SSO login
Linux Kernel Development, 3rd edition
Robert Love
Addison-Wesley

March 30, 2023

TCCS422: Operating Systems [Spring 2023]
School of Engineering and Technology, University of Washington - Tacoma

L2.77

77

QUESTIONS

78