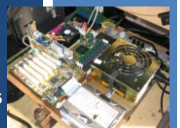
TCSS 422: OPERATING SYSTEMS



Operating Systems –
Three Easy Pieces & Processes

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

1

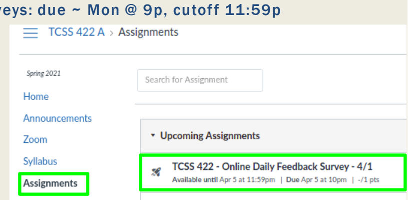
OBJECTIVES – 4/3

- Questions from 4/1
- 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

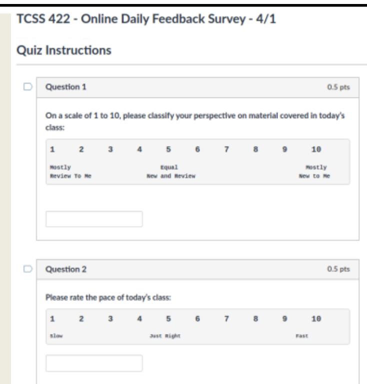
2

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



3



4

MATERIAL / PACE

- Please classify your perspective on material covered in today's class (51 of 64 respondents – 79.69%):
- 1-mostly review, 5-equal new/review, 10-mostly new
- <u>**Average – 5.92**</u> <u>*(Spring 2024, 5.44)*</u>
- Please rate the pace of today's class:
- 1-slow, 5-just right, 10-fast
- <u>**Average – 5.26**</u> <u>*(Spring 2024, 5.22)*</u>

5

FEEDBACK FROM 4/1

- <u>***How virtualization is achieved in an OS is unclear.***</u>
- <u>***How does the OS seemingly create multiple CPUs for each running program?***</u>
  - CPUs have a fixed number of PHYSICAL & LOGICAL cores
  - These cores are simultaneously shared among programs
  - '**htop**' includes CPU core graphs showing CPU utilization/core
  - Around 2004, Intel introduced 'hyperthreading', where each physical CPU has two active hyper-threads that share the physical CPU components to mimic two actual cores.
  - Since the advent of hyperthreading, OSes like Linux now report 'LOGICAL' cores when asked: *'how many CPUs do you have?'*
    - Check available cores with '**lscpu**' or '**cat /proc/cpuinfo**'

6

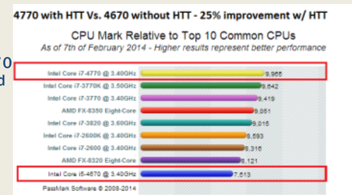
FEEDBACK - 2

- Consider some Intel desktop CPUs from 2013:
  - Intel Core i7-4770R and
  - Intel Core i5-4760R CPUs
- These are the exact same chip… *BUT*
  - The i5 CPU is imperfect, so Intel disabled hyperthreading and sold this as a lower-grade CPU
- Using 'google search', look up the specs for these CPUs
- You can type any CPU model number into Google followed by 'wikichip' to obtain a wikichip.org CPU page
- Wikipedia also had detailed CPU info
- Search for: 'Intel i7-4770R wikichip' and 'Intel i5-4760R wikichip'
  - How many CPU cores do they have ? How many hyperthreads ?

7

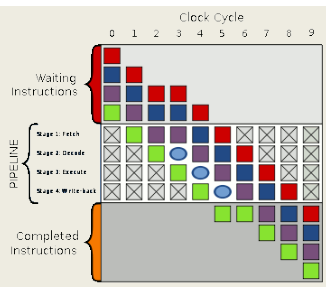
HYPER THREADING

- Modern CPUs provide multiple instruction pipelines, supporting multiple execution threads, usually 2 to feed instructions to a single CPU core…
- Two hyper-threads are not equivalent to (2) CPU cores
- i7-4770 and i5-4670 same CPU, with and without HTT
- Example: → hyperthreads add +32.9%



8

CPU PIPELINING



9

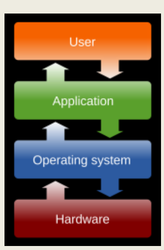
FEEDBACK - 2

- ***How does the OS allow multiple processes to use the same address space?***
- If the computer has 4GB of memory, then each program is presented with a virtual memory address space divided into 4 kb memory pages from 0 to 4GB of ram.
- Every page in this virtual memory is either vacant (unused) or present (occupied).
- Present pages are stored in physical memory (the real 4 GB), and the virtual memory space is just a map to the physical memory
- Every process has its own virtual memory map
- *We cover this extensively later in the course*

10

FEEDBACK - 3

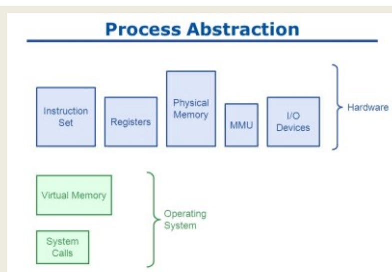
- ***The concept of 'abstraction' in OSes is unclear to me***
- The OS provides a layer between the hardware (HW) and user
- Within this layer, the OS choose how to represent HW resources to the users



11

FEEDBACK - 4

- The **process** abstraction is how OSes virtualize the CPU
- In other words, the resources of the CPU are shared with the user via 'processes'



12

RESOURCES

- **Textbook coupon 10% off "SHELFWORTHY10" until Friday at 11:59pm**
- **Hardcover edition (version 1.1) from lulu.com:**
- https://www.lulu.com/shop/andrea-arpaci-dusseau-and-remzi-arpaci-dusseau/operating-systems-three-easy-pieces-hardcover-version-110/hardcover/product-15gjeeky.html?q=three+easy+pieces+softcover&page=1&pageSize=4
- With coupon textbook is only $35.77 + tax & shipping

April 3, 2025 | TCSS422: Operating Systems [Spring 2025] School of Engineering and Technology, University of Washington - Tacoma | L2.13

13

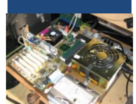
OBJECTIVES – 4/3

- Questions from 4/1
- **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

April 3, 2025 | TCSS422: Operating Systems [Spring 2025] School of Engineering and Technology, University of Washington - Tacoma | L2.14

14

C REVIEW SURVEY - AVAILABLE THRU 4/7



April 3, 2025 | TCSS422: Operating Systems [Spring 2025] School of Engineering and Technology, University of Washington - Tacoma | L2.15

15

OBJECTIVES – 4/3

- Questions from 4/1
- 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

April 3, 2025 | TCSS422: Operating Systems [Spring 2025] School of Engineering and Technology, University of Washington - Tacoma | L2.16

16

STUDENT BACKGROUND SURVEY

- Please complete the Student Background Survey
- **https://forms.gle/L1VWMoYrNueKe88dA**
- **41 of 64 Responses** as of 4/2 @ ~12pm
- Current Standings:
  - Best Office Hours times so far:
    - Rank #1: Wednesday before noon – 51.3%
    - Rank #2: Friday early afternoon (12-2pm) – 46.2%
    - Prefer online – 53.8%
- Will consider survey results through Mon Apr 8

April 3, 2025 | TCSS422: Operating Systems [Spring 2025] School of Engineering and Technology, University of Washington - Tacoma | L2.17

17

OBJECTIVES – 4/3

- Questions from 4/1
- 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

April 3, 2025 | TCSS422: Operating Systems [Spring 2025] School of Engineering and Technology, University of Washington - Tacoma | L2.18

18

VIRTUAL MACHINE SURVEY

- Please complete the Virtual Machine Survey to request a "School of Engineering and Technology" remote hosted Ubuntu VM
- https://forms.gle/vuEv5bsW57Ki4ZpDA
- **35 of 64 Responses as of 4/2 @ ~12pm**
- **Will close Wednesday 4/9...**
- **VM requests will be sent to SET for creation**
- **Survey response not required if no VM desired**

April 3, 2025 | TCSS422: Operating Systems [Spring 2025] School of Engineering and Technology, University of Washington - Tacoma | L2.19

19

OBJECTIVES – 4/3

- Questions from 4/1
- 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

April 3, 2025 | TCSS422: Operating Systems [Spring 2025] School of Engineering and Technology, University of Washington - Tacoma | L2.20

20

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, File System(s)

April 3, 2025 | TCSS422: Operating Systems [Spring 2025] School of Engineering and Technology, University of Washington - Tacoma | L2.21

21

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 3, 2025 | TCSS422: Operating Systems [Spring 2025] School of Engineering and Technology, University of Washington - Tacoma | L2.22

22

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 3, 2025 | TCSS422: Operating Systems [Spring 2025] School of Engineering and Technology, University of Washington - Tacoma | L2.23

23

OBJECTIVES – 4/3

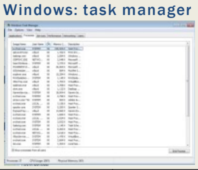
- Questions from 4/1
- C Review Survey - available thru 4/7
- Student Background Survey
- Virtual Machine Survey
- Chapter 2: Operating Systems – Three Easy Pieces
  - Concepts of virtualization/abstraction
  - **Three Easy Pieces: CPU** Memory, I/O
  - Concurrency
  - Operating system design goals
- Chapter 4: Processes
  - Process states, context switches
  - Kernel data structures for processes and threads

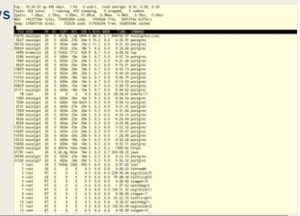
April 3, 2025 | TCSS422: Operating Systems [Spring 2025] School of Engineering and Technology, University of Washington - Tacoma | L2.24

24

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





25

VIRTUALIZING THE CPU - 2

- Simple Looping C Program

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/time.h>
#include <assert.h>
#include "common.h"

int
main(int argc, char *argv[])
{
        if (argc != 2) {
                fprintf(stderr, "usage: cpu <string>\n");
                exit(1);
        }
        char *str = argv[1];
        while (1) {
                Spin(1); // Repeatedly checks the time and
                returns once it has run for a second
                printf("%s\n", str);
        }
        return 0;
}
```


26

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…

27

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!

28

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

29

OBJECTIVES – 4/3

- Questions from 4/1
- 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

30

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

VIRTUALIZING MEMORY - 2

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

```
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include "common.h"

int
main(int argc, char *argv[])
{
        int *p = malloc(sizeof(int));  // a1: allocate some
                                         memory
        assert(p != NULL);
        printf("(%d) address of p: %08x\n",
                getpid(), (unsigned) p);  // a2: print out the
                                         address of the memmory
        *p = 0;  // a3: put zero into the first slot of the memory
        while (1) {
                Spin(1);
                *p = *p + 1;
                printf("(%d) p: %d\n", getpid(), *p); // a4
        }
        return 0;
}
```


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

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

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

OBJECTIVES – 4/3

- Questions from 4/1
- 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

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

April 3, 2025 | TCSS422: Operating Systems [Spring 2025] School of Engineering and Technology, University of Washington - Tacoma | L2.37

37

PERSISTENCE - 2

```
#include <stdio.h>
#include <unistd.h>
#include <assert.h>
#include <fcntl.h>
#include <sys/types.h>

int
main(int argc, char *argv[])
{
        int fd = open("/tmp/file", O_WRONLY | O_CREAT
                      | O_TRUNC, S_IRWXU);
        assert(fd > -1);
        int rc = write(fd, "hello world\n", 13);
        assert(rc == 13);
        close(fd);
        return 0;
}
```

- 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”`

April 3, 2025 | TCSS422: Operating Systems [Spring 2025] School of Engineering and Technology, University of Washington - Tacoma | L2.38

38

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*)

April 3, 2025 | TCSS422: Operating Systems [Spring 2025] School of Engineering and Technology, University of Washington - Tacoma | L2.39

39

WE WILL RETURN AT 4:50PM

April 3, 2025 | TCSS422: Operating Systems [Spring 2025] School of Engineering and Technology, University of Washington - Tacoma | L2.40

40

OBJECTIVES – 4/3

- Questions from 4/1
- 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

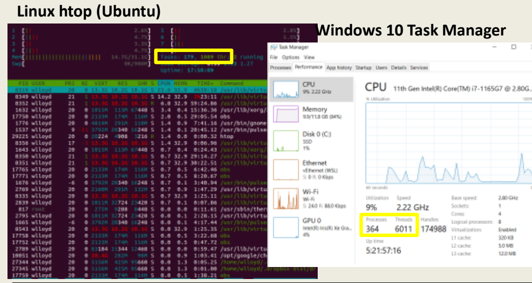
April 3, 2025 | TCSS422: Operating Systems [Spring 2025] School of Engineering and Technology, University of Washington - Tacoma | L2.41

41

CONCURRENCY

Linux htop (Ubuntu)

Windows 10 Task Manager



April 3, 2025 | TCSS422: Operating Systems [Spring 2025] School of Engineering and Technology, University of Washington - Tacoma | L2.42

42

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?***

April 3, 2025 | TCSS422: Operating Systems [Spring 2025] School of Engineering and Technology, University of Washington - Tacoma | L2.43

43

CONCURRENCY - 2

```
#include <stdio.h>
#include <stdlib.h>
#include "common.h"

volatile int counter = 0;
int loops;

void *worker(void *arg) {
        int i;
        for (i = 0; i < loops; i++) {
                counter++;
        }
        return NULL;
}
...
```

pthread.c

Listing continues …

April 3, 2025 | TCSS422: Operating Systems [Spring 2025] School of Engineering and Technology, University of Washington - Tacoma | L2.44

44

CONCURRENCY - 2

```
#include <stdio.h>
#include <stdlib.h>
#include "common.h"

volatile int counter = 0;
int loops;

void
}
...
```

Not the same as Java volatile *(java guarantees visibility of changes):*
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 …

April 3, 2025 | TCSS422: Operating Systems [Spring 2025] School of Engineering and Technology, University of Washington - Tacoma | L2.45

45

CONCURRENCY - 3

pthread.c

```
int
main(int argc, char *argv[])
{
        if (argc != 2) {
                fprintf(stderr, "usage: threads <value>\n");
                exit(1);
        }
        loops = atoi(argv[1]);
        pthread_t p1, p2;
        printf("Initial value : %d\n", counter);

        Pthread_create(&p1, NULL, worker, NULL);
        Pthread_create(&p2, NULL, worker, NULL);
        Pthread_join(p1, NULL);
        Pthread_join(p2, NULL);
        printf("Final value : %d\n", counter);
        return 0;
}
```

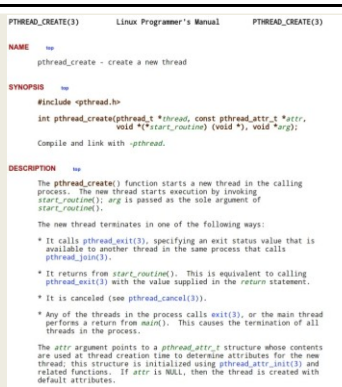
- Program creates two threads
- Check documentation: "man pthread_create"
- worker() method counts from 0 to argv[1] (loop)

April 3, 2025 | TCSS422: Operating Systems [Spring 2025] School of Engineering and Technology, University of Washington - Tacoma | L2.46

46

Linux "man" page

example


PTHREAD_CREATE(3) Linux Programmer's Manual PTHREAD_CREATE(3)

NAME top

pthread_create - create a new thread

SYNOPSIS top

```
#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 top

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

April 3, 2025 | TCSS422: Operating Systems [Spring 2025] School of Engineering and Technology, University of Washington - Tacoma | L2.47

47

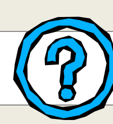
CONCURRENCY - 4

- 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 ???
```



April 3, 2025 | TCSS422: Operating Systems [Spring 2025] School of Engineering and Technology, University of Washington - Tacoma | L2.48

48

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*

April 3, 2025 | TCSS422: Operating Systems [Spring 2025] School of Engineering and Technology, University of Washington - Tacoma | L2.49

49

50

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 3, 2025 | TCSS422: Operating Systems [Spring 2025] School of Engineering and Technology, University of Washington - Tacoma | L2.51

51

OBJECTIVES – 4/3

- Questions from 4/1
- 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

April 3, 2025 | TCSS422: Operating Systems [Spring 2025] School of Engineering and Technology, University of Washington - Tacoma | L2.52

52

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 3, 2025 | TCSS422: Operating Systems [Spring 2025] School of Engineering and Technology, University of Washington - Tacoma | L2.53

53

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 3, 2025 | TCSS422: Operating Systems [Spring 2025] School of Engineering and Technology, University of Washington - Tacoma | L2.54

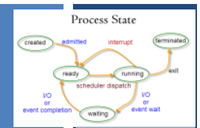
54

OBJECTIVES – 4/3

- Questions from 4/1
- 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

April 3, 2025 | TCSS422: Operating Systems [Spring 2025] School of Engineering and Technology, University of Washington - Tacoma | L2.55

55



CHAPTER 4: PROCESSES

April 3, 2025 | TCSS422: Operating Systems [Spring 2025] School of Engineering and Technology, University of Washington - Tacoma | L2.56

56

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 3, 2025 | TCSS422: Operating Systems [Spring 2025] School of Engineering and Technology, University of Washington - Tacoma | L2.57

57

PROCESS

A process is a running program.

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

April 3, 2025 | TCSS422: Operating Systems [Spring 2025] School of Engineering and Technology, University of Washington - Tacoma | L2.58

58

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 3, 2025 | TCSS422: Operating Systems [Spring 2025] School of Engineering and Technology, University of Washington - Tacoma | L2.59

59

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 3, 2025 | TCSS422: Operating Systems [Spring 2025] School of Engineering and Technology, University of Washington - Tacoma | L2.60

60

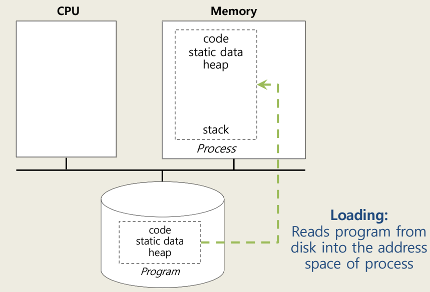
PROCESS API: CREATE

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

61



62

OBJECTIVES – 4/3

- Questions from 4/1
- 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

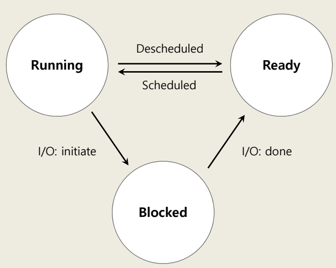
63

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

64

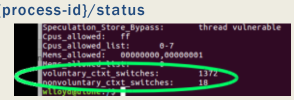
PROCESS STATE TRANSITIONS



65

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

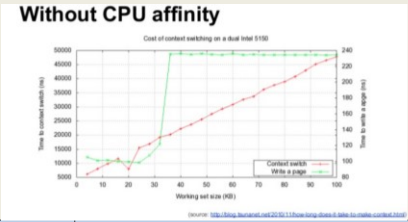


- proc "status" is a virtual file generated by Linux
- Provides a report with process related meta-data
- **What appears to happen to the number of context switches the longer a process runs? (mem.c)**

66

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



April 2, 2020 | TCSS422: Operating Systems [Spring 2020] School of Engineering and Technology, University of Washington - Tacoma | L2.67

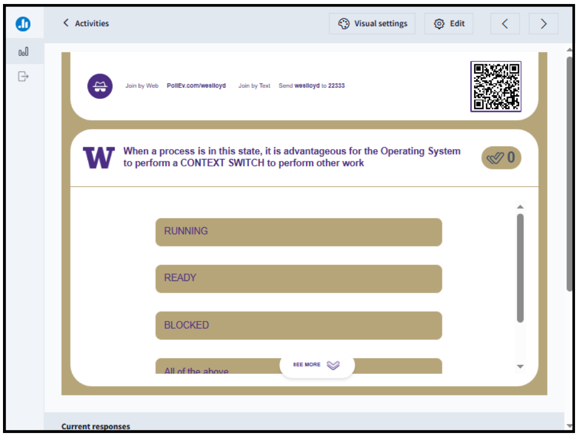
67

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 | L2.68

68



69

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 3, 2025 | TCSS422: Operating Systems [Spring 2025] School of Engineering and Technology, University of Washington - Tacoma | L2.70

70

OBJECTIVES – 4/3

- Questions from 4/1
- 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**

April 3, 2025 | TCSS422: Operating Systems [Spring 2025] School of Engineering and Technology, University of Washington - Tacoma | L2.71

71

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 3, 2025 | TCSS422: Operating Systems [Spring 2025] School of Engineering and Technology, University of Washington - Tacoma | L2.72

72

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 3, 2025 | TCSS422: Operating Systems [Spring 2025] School of Engineering and Technology, University of Washington - Tacoma | L2.73

73

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 3, 2025 | TCSS422: Operating Systems [Spring 2025] School of Engineering and Technology, University of Washington - Tacoma | L2.74

74

LINUX: STRUCTURES

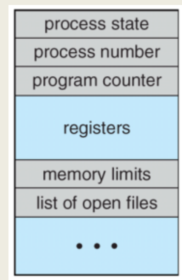
- **struct task_struct**, equivalent to **struct proc**
  - The Linux process data structure
  - Kernel data type (i.e. record) that describes individual Linux processes
  - Structure is VERY LARGE: **10,000+ bytes**
  - Defined in: /usr/src/linux-headers-{kernel version}/include/linux/sched.h
    -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

April 3, 2025 | TCSS422: Operating Systems [Spring 2025] School of Engineering and Technology, University of Washington - Tacoma | L2.75

75

STRUCT TASK_STRUCT PROCESS CONTROL BLOCK

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



April 3, 2025 | TCSS422: Operating Systems [Spring 2025] School of Engineering and Technology, University of Washington - Tacoma | L2.76

76

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

April 3, 2025 | TCSS422: Operating Systems [Spring 2025] School of Engineering and Technology, University of Washington - Tacoma | L2.77

77

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

April 3, 2025 | TCSS422: Operating Systems [Spring 2025] School of Engineering and Technology, University of Washington - Tacoma | L2.78

78

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

79

QUESTIONS

80