

1

2

3

4

5

6

MATERIAL / PACE

- Please classify your perspective on material covered in today's class (57 of 63 respondents – 90.48% !!):
- 1-mostly review, 5-equal new/review, 10-mostly new
- **Average – 5.86 (↓ - previous 5.92)**
- Please rate the pace of today's class:
- 1-slow, 5-just right, 10-fast
- **Average – 5.11 (↓ - previous 5.26)**

April 8, 2025

TCSS422: Computer Operating Systems [Spring 2025]  
School of Engineering and Technology, University of Washington - Tacoma

L3.7

7

FEEDBACK FROM 4/3

- I do not understand how a virtual address space can be the same size as the physical address space. If the entire 4GB virtual address space is full for a single program, won't the entire 4GB physical address space be full with no space left for the other programs?
- Uses use lazy loading. Code pages are only read into memory when they are needed. In the rare event that a single program consumes all of the physical memory, Linux augments RAM with Swap space
- Swap space is a disk volume used as extended memory
- Commands to check your swap space:

```
free
cat /proc/swaps
lsblk | grep -C 5 -i swap
swapon -s
vmstat 1
```

April 8, 2025

TCSS422: Operating Systems [Spring 2025]  
School of Engineering and Technology, University of Washington - Tacoma

L3.8

8

FEEDBACK - 2

- ***Are we using ubuntu as our main OS to work from? or are we allowed to use windows/MacOS?***
- The OS directly installed on your computer which boots the computer is called the **host operating system**.
- This can be Windows, MacOS, Linux, etc.
- Ubuntu will be installed on a virtual machine (VM)
- The OS installed on a VM is called a **guest operating system**.

April 8, 2025

TCSS422: Operating Systems [Spring 2025]  
School of Engineering and Technology, University of Washington - Tacoma

L3.9

9

FEEDBACK - 3

- ***What exactly is the difference between processes and threads in regards to OS' virtualization of memory?***
- ***If my understanding is correct, then I know that many threads can share the same physical address space via virtualization, but does the same apply to processes?***
- Threads share the same virtual memory space as the parent process.
- This sharing of memory is what causes synchronization errors when two threads try to modify shared memory at the same time without proper coordination
- Every process has its own distinct virtual memory space

April 8, 2025

TCSS422: Operating Systems [Spring 2025]  
School of Engineering and Technology, University of Washington - Tacoma

L3.10

10

FEEDBACK - 4

- ***What counts as a process? Does a running program constitute a single process? Or are there many processes in a single program?***
- As we see when inspecting running processes using tools like 'top', 'htop', and 'ps aux', programs like Zoom and Chrome have multiple processes
- Not every program uses multiple processes
- The issue with multiple processes is they \* don't \* share memory, so it is hard to coordinate multiple processes
- Linux features IPC – Inter Process Communication
- This is the notion of use "I/O streams" similar to files called pipes to allow programs to open various streams between each other

April 8, 2025

TCSS422: Operating Systems [Spring 2025]  
School of Engineering and Technology, University of Washington - Tacoma

L3.11

11

FEEDBACK - 5

- ***Why does the shared counter work at smaller loop lengths but not at larger loops?***
- ***I would expect it to be unstable at all scales.***

April 8, 2025

TCSS422: Operating Systems [Spring 2025]  
School of Engineering and Technology, University of Washington - Tacoma

L3.12

12

MOTIVATION FOR LINUX

- It is worth noting the importance of Linux for today's developers and computer scientists.
- The CLOUD runs many virtual machines, recently in 2019 a key milestone was reached.
- Even on Microsoft Azure (the Microsoft Cloud), there were more Linux Virtual Machines (> 50%) than Windows.
- <https://www.zdnet.com/article/microsoft-developer-reveals-linux-is-now-more-used-on-azure-than-windows-server/>
- <https://www.zdnet.com/article/it-runs-on-the-cloud-and-the-cloud-runs-on-linux-any-questions/>
- The majority of application back-ends (server-side), cloud or not, run on Linux.
- This is due to licensing costs, example:

April 8, 2025

TCSS422: Operating Systems [Spring 2025]  
School of Engineering and Technology, University of Washington - Tacoma

L3.13

13

MOTIVATION FOR LINUX - 2

- Consider an example where you're asked to develop a web services backend that requires 10 x 8-CPU-core virtual servers
- Your organization investigates hosting costs on Amazon cloud
- 8-core VM is "c5d.2xlarge"

Name	Instance type	Memory	vCPUs	Linux On Demand cost	Windows On Demand cost
CS High-CPU Extra Large	c5d.xlarge	8.0 GiB	4 vCPUs	\$0.192000 hourly	\$0.376000 hourly
CS High-CPU 1xlarge	c5d.1xlarge	16.0 GiB	8 vCPUs	\$0.456000 hourly	\$0.768000 hourly
CS High-CPU Large	c5d.large	4.0 GiB	2 vCPUs	\$0.096000 hourly	\$0.188000 hourly
CS High-CPU 2xlarge	c5d.2xlarge	16.0 GiB	8 vCPUs	\$0.456000 hourly	\$0.768000 hourly
CS High-CPU Quadruple Extra Large	c5d.4xlarge	32.0 GiB	16 vCPUs	\$0.768000 hourly	\$1.504000 hourly
CS High-CPU 8xlarge	c5d.8xlarge	128.0 GiB	64 vCPUs	\$3.072000 hourly	\$6.016000 hourly
CS High-CPU Double Extra Large	c5d.2xlarge	16.0 GiB	8 vCPUs	\$0.384000 hourly	\$0.752000 hourly
CS High-CPU z2xlarge	c5d.z2xlarge	96.0 GiB	48 vCPUs	\$2.400000 hourly	\$4.512000 hourly
CS High-CPU 12xlarge	c5d.12xlarge	172.0 GiB	36 vCPUs	\$3.720000 hourly	\$6.384000 hourly

- Windows hourly price 75.2¢
- Linux hourly price 38.4¢
- See: <https://instances.vantage.sh/>

April 8, 2025

TCSS422: Operating Systems [Spring 2025]  
School of Engineering and Technology, University of Washington - Tacoma

L3.14

14

MOTIVATION FOR LINUX - 2

- One year cloud hosting cost:

CS High-CPU 8xlarge

CS High-CPU 8xlarge

CS High-CPU 8xlarge

CS High-CPU 8xlarge

CS High-CPU 8xlarge

CS High-CPU 8xlarge

CS High-CPU 8xlarge

CS High-CPU 8xlarge

CS High-CPU 8xlarge

CS High-CPU 8xlarge

Windows

Linux

Windows comes at a 95.8% price premium

10 VMs x 8,760 hours x \$.752 = \$65,875.20

10 VMs x 8,760 hours x \$.384 = \$33,638.40

- See: <https://www.ec2instances.info/>

April 8, 2025

TCSS422: Operating Systems [Spring 2025]  
School of Engineering and Technology, University of Washington - Tacoma

L3.15

15

OBJECTIVES – 4/8

- Questions from 4/3
- C Review Survey – Late After Apr 9. Closes Apr 11
- Student Background Survey
- Virtual Machine Survey: VM to be sent to S. Miasishchev
- Assignment 0
- Chapter 4: Processes
  - Process states, context switches
  - Kernel data structures for processes and threads
- Chapter 5: Process API
  - fork(), wait(), exec()

April 8, 2025

TCSS422: Operating Systems [Spring 2025]  
School of Engineering and Technology, University of Washington - Tacoma

L3.16

16

C REVIEW SURVEY - AVAILABLE THRU 4/5

April 8, 2025

TCSS422: Operating Systems [Spring 2025]  
School of Engineering and Technology, University of Washington - Tacoma

L3.17

17

OBJECTIVES – 4/8

- Questions from 4/3
- C Review Survey – Late After Apr 9, Closes Apr 11
- Student Background Survey
- Virtual Machine Survey: VM to be sent to S. Miasishchev
- Assignment 0
- Chapter 4: Processes
  - Process states, context switches
  - Kernel data structures for processes and threads
- Chapter 5: Process API
  - fork(), wait(), exec()

April 8, 2025

TCSS422: Operating Systems [Spring 2025]  
School of Engineering and Technology, University of Washington - Tacoma

L3.18

18

STUDENT BACKGROUND SURVEY

- **46 of 63 Responses** as of 4/8 @ ~12am
- **Current Standings:**
  - **Best Office Hours times so far:**
    - Rank #1: Wednesday morning (before noon) ✓ (47.7%)
    - Rank #2: Friday early afternoon ✓ (12-2p) (45.5%)
  - **Format:**
    - Rank #1: Prefer online (Zoom) ✓ (54.5%)

April 8, 2025

TCSS422: Operating Systems [Spring 2025]  
School of Engineering and Technology, University of Washington - Tacoma

L3.19

19

TCSS 422 – OFFICE HRS – SPRING 2025

- **Session 1. Wednesday 11am to noon**
  - This session will be held by zoom.
- **Session 2. Fridays noon to 1pm (Zoom)**
  - This session will mostly be held on zoom.
  - Some Fridays will be canceled due to instructor scheduling conflicts
    - Known conflicts on 4/11, 4/18, 5/16 (?)
- Zoom links for Office Hours will be shared via Canvas
- Also available after class on Tuesdays and Thursdays in CP 229 at 6pm

April 8, 2025

TCSS422: Operating Systems [Spring 2025]  
School of Engineering and Technology, University of Washington - Tacoma

L3.20

20

OBJECTIVES – 4/8

- Questions from 4/3
- C Review Survey – Late After Apr 9, Closes Apr 11
- Student Background Survey
- **Virtual Machine Survey: VM to be sent to S. Miasishchev**
- Assignment 0
- Chapter 4: Processes
  - Process states, context switches
  - Kernel data structures for processes and threads
- Chapter 5: Process API
  - fork(), wait(), exec()

April 8, 2025

TCSS422: Operating Systems [Spring 2025]  
School of Engineering and Technology, University of Washington - Tacoma

L3.21

21

VIRTUAL MACHINE SURVEY

- Please complete the Virtual Machine Survey to request a “School of Engineering and Technology” remote hosted Ubuntu VM
- <https://forms.gle/jSwcl1qeKDy2W9498>
- **VM requests have been sent to SET sys admin Slava Miasishchev for set up**
- **If you missed the survey, and need a VM, please complete it**

April 8, 2025

TCSS422: Operating Systems [Spring 2025]  
School of Engineering and Technology, University of Washington - Tacoma

L3.22

22

OBJECTIVES – 4/8

- Questions from 4/3
- C Review Survey – Late After Apr 9, Closes Apr 11
- Student Background Survey
- Virtual Machine Survey: VM to be sent to S. Miasishchev
- **Assignment 0**
- Chapter 4: Processes
  - Process states, context switches
  - Kernel data structures for processes and threads
- Chapter 5: Process API
  - fork(), wait(), exec()

April 8, 2025

TCSS422: Operating Systems [Spring 2025]  
School of Engineering and Technology, University of Washington - Tacoma

L3.23

23

CHAPTER 2 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 8, 2025

TCSS422: Operating Systems [Spring 2025]  
School of Engineering and Technology, University of Washington - Tacoma


L3.24

24

CHAPTER 2 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 8, 2025

TCCS422: Operating Systems [Spring 2025]  
School of Engineering and Technology, University of Washington - Tacoma

L3.25

25

OBJECTIVES – 4/8

- Questions from 4/3
- C Review Survey – Late After Apr 9, Closes Apr 11
- Student Background Survey
- Virtual Machine Survey: VM to be sent to S. Miasishchev
- Assignment 0

■ **Chapter 4: Processes**

- Process states, context switches
- Kernel data structures for processes and threads

- Chapter 5: Process API
  - fork(), wait(), exec()

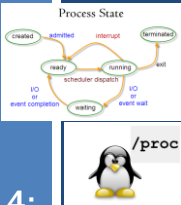
April 8, 2025

TCCS422: Operating Systems [Spring 2025]  
School of Engineering and Technology, University of Washington - Tacoma

L3.26

26

CHAPTER 4:  
PROCESSES



April 8, 2025

TCCS422: Operating Systems [Spring 2025]  
School of Engineering and Technology, University of Washington - Tacoma

L3.27

27

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 8, 2025

TCCS422: Operating Systems [Spring 2025]  
School of Engineering and Technology, University of Washington - Tacoma

L3.28

28

PROCESS

A process is a running program.

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

April 8, 2025

TCCS422: Operating Systems [Spring 2025]  
School of Engineering and Technology, University of Washington - Tacoma

L3.29

29

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 8, 2025

TCCS422: Operating Systems [Spring 2025]  
School of Engineering and Technology, University of Washington - Tacoma

L3.30

30

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 8, 2025

TCSS422: Operating Systems (Spring 2025)  
School of Engineering and Technology, University of Washington - Tacoma

L3.31

31

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()`

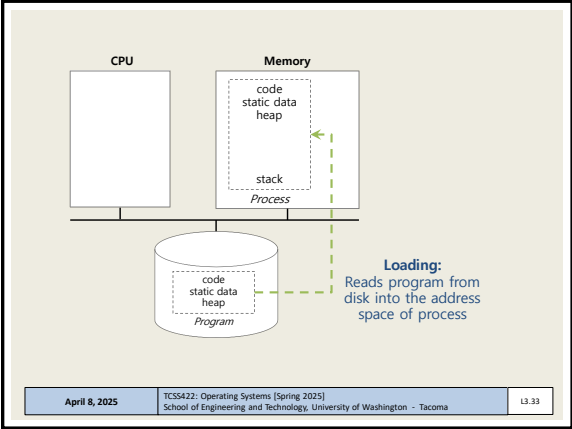
- OS transfers CPU control to the new process

April 8, 2025

TCSS422: Operating Systems (Spring 2025)  
School of Engineering and Technology, University of Washington - Tacoma

L3.32

32



33

WE WILL RETURN AT 5:00PM

April 8, 2025

TCSS422: Operating Systems (Spring 2025)  
School of Engineering and Technology, University of Washington - Tacoma

L3.34

34

OBJECTIVES – 4/8

Questions from 4/3

C Review Survey – Late After Apr 9, Closes Apr 11

Student Background Survey

Virtual Machine Survey: VM to be sent to S. Miasishchev

Assignment 0

Chapter 4: Processes

- Process states, context switches**
  - Kernel data structures for processes and threads

Chapter 5: Process API

- `fork()`, `wait()`, `exec()`

April 8, 2025

TCSS422: Operating Systems (Spring 2025)  
School of Engineering and Technology, University of Washington - Tacoma

L3.35

35

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 8, 2025

TCSS422: Operating Systems (Spring 2025)  
School of Engineering and Technology, University of Washington - Tacoma

L3.36

36

PROCESS STATE TRANSITIONS

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

April 8, 2025

TCSS422: Operating Systems [Spring 2025]  
School of Engineering and Technology, University of Washington - Tacoma

L3.37

37

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
cpu_allowed: ff
cpu_allowed_list: 0-7
mem_allowed: 00000000,00000001
mem_allowed_list:
voluntary_ctxt_switches: 117
involuntary_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 8, 2025

TCSS422: Operating Systems [Spring 2025]  
School of Engineering and Technology, University of Washington - Tacoma

L3.38

38

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**

Cost of context switching on a dual Intel Xeon

Time to switch to target (ns)

Working set size (KB)

Context switch with a page

Context switch without a page

April 8, 2025

TCSS422: Operating Systems [Spring 2025]  
School of Engineering and Technology, University of Washington - Tacoma

L3.39

39

Activities

Visual settings

Edit

<

>

When poll is active respond at Pollix.com/wealloyd

Send wealloyd to 22333

QR code

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

Get more

Current responses

40

QUESTION: WHEN TO CONTEXT SWITCH

- When a process is about to go into 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 8, 2025

TCSS422: Operating Systems [Spring 2025]  
School of Engineering and Technology, University of Washington - Tacoma

L3.41

41

OBJECTIVES – 4/8

- Questions from 4/3
- C Review Survey – Late After Apr 9, Closes Apr 11
- Student Background Survey
- Virtual Machine Survey: VM to be sent to S. Miasishchev
- Assignment 0
- Chapter 4: Processes
  - Process states, context switches
  - Kernel data structures for processes and threads
- Chapter 5: Process API
  - fork(), wait(), exec()

April 8, 2025

TCSS422: Operating Systems [Spring 2025]  
School of Engineering and Technology, University of Washington - Tacoma

L3.42

42

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 8, 2025

TCSS422: Operating Systems [Spring 2025]  
School of Engineering and Technology, University of Washington - Tacoma

L3.43

43

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
...

April 8, 2025

TCSS422: Operating Systems [Spring 2025]  
School of Engineering and Technology, University of Washington - Tacoma

L3.44

44

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 8, 2025

TCSS422: Operating Systems [Spring 2025]  
School of Engineering and Technology, University of Washington - Tacoma

L3.45

45

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 8, 2025

TCSS422: Operating Systems [Spring 2025]  
School of Engineering and Technology, University of Washington - Tacoma

L3.46

46

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 kernel version 6.11, **LOC 758 - 1588**
    - Ubuntu kernel version 5.15, **LOC: 721 - 1507**
    - Ubuntu kernel version 5.11, **LOC: 657 - 1394**
    - Ubuntu kernel version 4.4, **LOC: 1391 - 1852**

April 8, 2025

TCSS422: Operating Systems [Spring 2025]  
School of Engineering and Technology, University of Washington - Tacoma

L3.47

47

STRUCT TASK\_STRUCT

- Key elements (e.g. PCB) in Linux are captured in struct task\_struct: (LOC from Linux kernel v 6.11)
- Process ID**
  - pid\_t pid; LOC #995
- Process State**
  - /\* -1 unrunnable, 0 runnable, >0 stopped: \*/
  - unsigned int \_\_state; LOC #766
- 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 #812
  - unsigned int time\_slice; LOC #583

April 8, 2025

TCSS422: Operating Systems [Spring 2025]  
School of Engineering and Technology, University of Washington - Tacoma

L3.48

48



STRUCT TASK\_STRUCT - 2

- **Address space of the process:**
- "mm" is short for "memory map"
- `struct mm_struct *mm;` LOC #898
- **Parent process**, that launched this one
- `struct task_struct __rcu *parent;` LOC #1009
- **Child processes** (as a list)
- `struct list_head children;` LOC #1017
- **Open files**
- `struct files_struct *files;` LOC #1121

April 8, 2025

TCSS422: Operating Systems [Spring 2025]  
School of Engineering and Technology, University of Washington - Tacoma

L3.49

49

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, 3<sup>rd</sup> edition  
Robert Love  
Addison-Wesley

April 8, 2025

TCSS422: Operating Systems [Spring 2025]  
School of Engineering and Technology, University of Washington - Tacoma

L3.50

50

OBJECTIVES – 4/8

- Questions from 4/3
- C Review Survey – Late After Apr 9, Closes Apr 11
- Student Background Survey
- Virtual Machine Survey: VM to be sent to S. Miasishchev
- Assignment 0
- Chapter 4: Processes
  - Process states, context switches
  - Kernel data structures for processes and threads
- **Chapter 5: Process API**
  - `fork()`, `wait()`, `exec()`


April 8, 2025

TCSS422: Operating Systems [Spring 2025]  
School of Engineering and Technology, University of Washington - Tacoma

L3.51

51

CHAPTER 5:  
C PROCESS API



April 8, 2025

TCSS422: Operating Systems [Spring 2025]  
School of Engineering and Technology, University of Washington - Tacoma

L3.52

52

OBJECTIVES – 4/8

- Questions from 4/3
- C Review Survey – Late After Apr 9, Closes Apr 11
- Student Background Survey
- Virtual Machine Survey: VM to be sent to S. Miasishchev
- Assignment 0
- Chapter 4: Processes
  - Process states, context switches
  - Kernel data structures for processes and threads
- **Chapter 5: Process API**
  - `fork()`, `wait()`, `exec()`

April 8, 2025


TCSS422: Operating Systems [Spring 2025]  
School of Engineering and Technology, University of Washington - Tacoma

L3.53

53

fork()

- Creates a new process - think of "a fork in the road"
- "Parent" process is the original
- Creates "child" process of the program from the **current execution point**
- Book says "pretty odd"
- Creates a **duplicate** program instance (these are **processes**)
- **Copy of**
  - Address space (memory)
  - Register
  - Program Counter (PC)
- Fork returns
  - child PID to parent
  - 0 to child



April 8, 2025

TCSS422: Operating Systems [Spring 2025]  
School of Engineering and Technology, University of Washington - Tacoma

L3.54

54

FORK EXAMPLE

■ p1.c

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

int main(int argc, char *argv[]){
    printf("hello world (pid:%d)\n", (int) getpid());
    int rc = fork();
    if (rc < 0) { // fork failed; exit
        fprintf(stderr, "fork failed\n");
        exit(1);
    } else if (rc == 0) { // child (new process)
        printf("hello, I am child (pid:%d)\n", (int) getpid());
    } else { // parent goes down this path (main)
        printf("hello, I am parent of %d (pid:%d)\n",
            rc, (int) getpid());
    }
    return 0;
}
```

April 8, 2025

TCSS422: Operating Systems [Spring 2025]  
School of Engineering and Technology, University of Washington - Tacoma

L3.55

55

FORK EXAMPLE - 2

■ Non deterministic ordering of execution

```
prompt> ./p1
hello world (pid:29146)
hello, I am parent of 29147 (pid:29146)
hello, I am child (pid:29147)
prompt>
```

or

```
prompt> ./p1
hello world (pid:29146)
hello, I am child (pid:29147)
hello, I am parent of 29147 (pid:29146)
prompt>
```

■ CPU scheduler determines which to run first

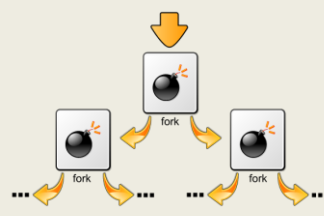
April 8, 2025

TCSS422: Operating Systems [Spring 2025]  
School of Engineering and Technology, University of Washington - Tacoma

L3.56

56

:(){:|:&};;



April 8, 2025

TCSS422: Operating Systems [Spring 2025]  
School of Engineering and Technology, University of Washington - Tacoma

L3.57

57

OBJECTIVES – 4/8

■ Questions from 4/3

■ C Review Survey – Late After Apr 9, Closes Apr 11

■ Student Background Survey

■ Virtual Machine Survey: VM to be sent to S. Miasishchev

■ Assignment 0

■ Chapter 4: Processes

- Process states, context switches
- Kernel data structures for processes and threads

■ Chapter 5: Process API

- fork(), wait(), exec()

April 8, 2025

TCSS422: Operating Systems [Spring 2025]  
School of Engineering and Technology, University of Washington - Tacoma

L3.58

58

wait()

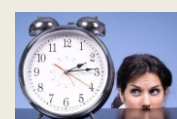
■ wait(), waitpid()

■ Called by parent process

■ Waits for a child process to finish executing

■ Not a sleep() function

■ Provides some ordering to multi-process execution



April 8, 2025

TCSS422: Operating Systems [Spring 2025]  
School of Engineering and Technology, University of Washington - Tacoma

L3.59

59

FORK WITH WAIT

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>

int main(int argc, char *argv[]){
    printf("hello world (pid:%d)\n", (int) getpid());
    int rc = fork();
    if (rc < 0) { // fork failed; exit
        fprintf(stderr, "fork failed\n");
        exit(1);
    } else if (rc == 0) { // child (new process)
        printf("hello, I am child (pid:%d)\n", (int) getpid());
    } else { // parent goes down this path (main)
        int wc = wait(NULL);
        printf("hello, I am parent of %d (wc:%d) (pid:%d)\n",
            rc, wc, (int) getpid());
    }
    return 0;
}
```

April 8, 2025

TCSS422: Operating Systems [Spring 2025]  
School of Engineering and Technology, University of Washington - Tacoma

L3.60

60

FORK WITH WAIT - 2

- Deterministic ordering of execution

```
prompt> ./p2
hello world (pid:29266)
hello, I am child (pid:29267)
hello, I am parent of 29267 (wc:29267) (pid:29266)
prompt>
```

April 8, 2025

TCSS422: Operating Systems [Spring 2025]  
School of Engineering and Technology, University of Washington - Tacoma

L3.61

61

FORK EXAMPLE

- Linux example

April 8, 2025

TCSS422: Operating Systems [Spring 2025]  
School of Engineering and Technology, University of Washington - Tacoma

L3.62

62

OBJECTIVES – 4/8

- Questions from 4/3
- C Review Survey – Late After Apr 9, Closes Apr 11
- Student Background Survey
- Virtual Machine Survey: VM to be sent to S. Miasishchev
- Assignment 0
- Chapter 4: Processes
  - Process states, context switches
  - Kernel data structures for processes and threads
- Chapter 5: Process API
  - fork(), wait(), **exec()**

April 8, 2025

TCSS422: Operating Systems [Spring 2025]  
School of Engineering and Technology, University of Washington - Tacoma

L3.63

63

exec()

- Supports running an external program by **“transferring control”**
- 6 types: execl(), execlp(), execlx(), execv(), execvp(), execvpe()
- execl(), execlp(), execlx(): const char \*arg (**example: execl.c**)  
Provide cmd and args as individual params to the function  
Each arg is a pointer to a null-terminated string  
**ODD**: pass a variable number of args: (arg0, arg1, .. argn)
- execv(), execvp(), execvpe() (**example: exec.c**)  
Provide cmd and args as an Array of pointers to strings  
Strings are null-terminated  
First argument is name of command being executed  
Fixed number of args passed in

April 8, 2025

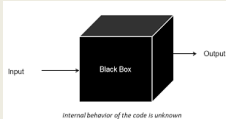
TCSS422: Operating Systems [Spring 2025]  
School of Engineering and Technology, University of Washington - Tacoma

L3.64

64

EXEC() - 2

- Common use case:
  - Write a new program which wraps a legacy one
  - Provide a new interface to an old system: Web services
  - Legacy program thought of as a “black box”
- We don’t want to know what is inside... ☹️



April 8, 2025

TCSS422: Operating Systems [Spring 2025]  
School of Engineering and Technology, University of Washington - Tacoma

L3.65

65

EXEC EXAMPLE

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/wait.h>

int main(int argc, char *argv[]){
    printf("hello world (pid:%d)\n", (int) getpid());
    int rc = fork();
    if (rc < 0) { // fork failed; exit
        fprintf(stderr, "fork failed\n");
        exit(1);
    } else if (rc == 0) { // child (new process)
        printf("hello, I am child (pid:%d)\n", (int) getpid());
        char *myargs[3];
        myargs[0] = strdup("wc"); // program: "wc" (word count)
        myargs[1] = strdup("p3.c"); // argument: file to count
        myargs[2] = NULL; // marks end of array
        ...
    }
```

April 8, 2025

TCSS422: Operating Systems [Spring 2025]  
School of Engineering and Technology, University of Washington - Tacoma

L3.66

66

EXEC EXAMPLE - 2

```
execvp(myargs[0], myargs); // runs word count
printf("this shouldn't print out");
} else { // parent goes down this path (main)
    int wc = wait(NULL);
    printf("Hello, I am parent of %d (wc:%d) (pid:%d)\n",
        rc, wc, (int) getpid());
}
return 0;
```

```
prompt> ./p3
hello world (pid:29383)
hello, I am child (pid:29384)
29 107 1030 p3.c
hello, I am parent of 29384 (wc:29384) (pid:29383)
prompt>
```

April 8, 2025

TCSS422: Operating Systems [Spring 2025]  
School of Engineering and Technology, University of Washington - Tacoma

L3.67

67

EXEC WITH FILE REDIRECTION (OUTPUT)

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <fcntl.h>
#include <sys/wait.h>

int
main(int argc, char *argv[]){
    int rc = fork();
    if (rc < 0) { // fork failed; exit
        fprintf(stderr, "fork failed\n");
        exit(1);
    } else if (rc == 0) { // child: redirect standard output to a file
        close(STDOUT_FILENO);
        open("./p4.output", O_CREAT|O_WRONLY|O_TRUNC, S_IRWXU);
    }
}
```

April 8, 2025

TCSS422: Operating Systems [Spring 2025]  
School of Engineering and Technology, University of Washington - Tacoma

L3.68

68

FILE MODE BITS

```
S_IRWXU
read, write, execute/search by owner
S_IRUSR
read permission, owner
S_IWUSR
write permission, owner
S_IXUSR
execute/search permission, owner
S_IRWXG
read, write, execute/search by group
S_IRGRP
read permission, group
S_IWGRP
write permission, group
S_IXGRP
execute/search permission, group
S_IRWXO
read, write, execute/search by others
S_IROTH
read permission, others
S_IWOTH
write permission, others
```

April 8, 2025

TCSS422: Operating Systems [Spring 2025]  
School of Engineering and Technology, University of Washington - Tacoma

L3.69

69

EXEC W/ FILE REDIRECTION (OUTPUT) - 2

```
// now exec "wc"...
char *myargs[3];
myargs[0] = strdup("wc"); // program: "wc" (word count)
myargs[1] = strdup("p4.c"); // argument: file to count
myargs[2] = NULL; // marks end of array
execvp(myargs[0], myargs); // runs word count
} else {
    int wc = wait(NULL);
}
return 0;
```

```
prompt> ./p4
prompt> cat p4.output
32 109 846 p4.c
prompt>
```

April 8, 2025

TCSS422: Operating Systems [Spring 2025]  
School of Engineering and Technology, University of Washington - Tacoma

L3.70

70

W Which Process API call is used to launch a different program from the current program?

Fork()

Exec()

Wait()

None of the above

SEE MORE

71

QUESTION: PROCESS API

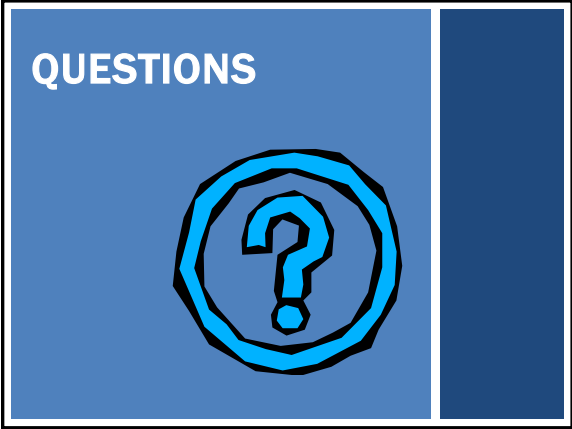
- Which Process API call is used to launch a different program from the current program?
- (a) Fork()
- (b) Exec()
- (c) Wait()
- (d) None of the above
- (e) All of the above

April 8, 2025

TCSS422: Operating Systems [Spring 2025]  
School of Engineering and Technology, University of Washington - Tacoma

L3.72

72



73