


TCCS 422: OPERATING SYSTEMS

Processes & The Process API



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

April 2, 2024 TCCS422: Operating Systems [Spring 2024]
School of Engineering and Technology, University of Washington - Tacoma

1

OBJECTIVES - 4/2

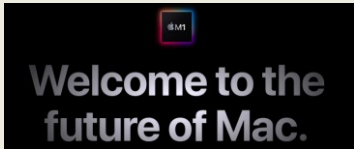
- **Questions from 3/28**
- C Review Survey – Available thru Friday Apr 5
- Student Background Survey
- Virtual Machine Survey: VM requests sent to S. Miasishchev
- Assignment 0
- Chapter 4: Processes
 - Process states, context switches
 - Kernel data structures for processes and threads
- Chapter 5: Process API
 - fork(), wait(), exec()

April 2, 2024 TCCS422: Operating Systems [Spring 2024]
School of Engineering and Technology, University of Washington - Tacoma L3.2

2

VIRTUAL MACHINE SUPPORT ON APPLE M1

- Installing a Ubuntu Virtual Machine on Apple M1 MacBooks:
- FREE
- <https://mac.getutm.app/>
- MACs have switched to using ARM-based CPUs
 - Motivation: faster, less expensive than Intel-based CPUs



April 2, 2024 TCCS422: Operating Systems [Spring 2024]
School of Engineering and Technology, University of Washington - Tacoma L3.3

3

TEXT BOOK COUPON

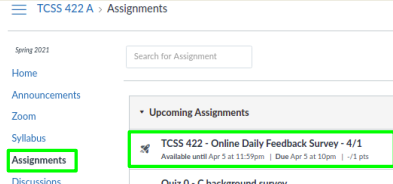
- 15% off textbook code: **POETRY15** (through Friday Apr 5)
- <https://www.lulu.com/shop/andrea-arpaci-dusseau-and-remzi-arpaci-dusseau/operating-systems-three-easy-pieces-hardcover-version-110/hardcover/product-15gieeky.html?q=three+easy+pieces+operating+systems&page=1&pageSize=4>
- With coupon textbook is only \$33.79 + tax & shipping

April 2, 2024 TCCS422: Operating Systems [Spring 2024]
School of Engineering and Technology, University of Washington - Tacoma L3.4

4

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, closes 11:59p
- Thursday surveys: due ~ Mon @ 9p, closes 11:59p



April 2, 2024 TCCS422: Computer Operating Systems [Spring 2024]
School of Engineering and Technology, University of Washington - Tacoma L3.5

5

TCCS 422 - Online Daily Feedback Survey - 4/1

Quiz Instructions

Question 1 0.5 pts

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

1	2	3	4	5	6	7	8	9	10
Not at all				Neutral					Not at all
Not at all				Not at all					Not at all

Question 2 0.5 pts

Please rate the pace of today's class:

1	2	3	4	5	6	7	8	9	10
slow				Just right					Fast

April 2, 2024 TCCS422: Computer Operating Systems [Spring 2024]
School of Engineering and Technology, University of Washington - Tacoma L3.6

6

MATERIAL / PACE

- Please classify your perspective on material covered in today's class (35 respondents):
- 1-mostly review, 5-equal new/review, 10-mostly new
- Average – 6.49 (↑ - previous 5.44)**
- Please rate the pace of today's class:
- 1-slow, 5-just right, 10-fast
- Average – 5.31 (↓ - previous 5.22)**

April 2, 2024
TCCS422: Computer Operating Systems (Spring 2024)
School of Engineering and Technology, University of Washington - Tacoma
L3.7

7

FEEDBACK FROM 3/28

- What are threads and processes from the perspective of a programmer?**
- > when should a programmer use threads ?
- > when should a programmer use processes ?
- How are concurrency and parallel programming related?**
 - Concurrency** - two or more things (processes or threads) executing at the same time.
 - Parallel programming** - writing code which splits a problem into smaller tasks that can be executed at the same time. Tasks will then be executed in parallel using multiple threads or processes

April 2, 2024
TCCS422: Operating Systems (Spring 2024)
School of Engineering and Technology, University of Washington - Tacoma
L3.8

8

FEEDBACK - 2

- What will the quizzes and exams look like in this course? Will they be In-person or through Canvas?**
- Quizzes are primarily held as in-class group activities
 - Remote participants can interact via Zoom breakout rooms
- Exams are in-person, w/ a few pages of notes allowed and basic calculator

April 2, 2024
TCCS422: Operating Systems (Spring 2024)
School of Engineering and Technology, University of Washington - Tacoma
L3.9

9

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 2, 2024
TCCS422: Operating Systems (Spring 2024)
School of Engineering and Technology, University of Washington - Tacoma
L3.10

10

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	\$0.150000 hourly	\$0.376000 hourly
CS High-CPU 18xlarge	c5d.18xlarge	144.0 GiB	72 vCPUs	\$3.456000 hourly	\$8.768000 hourly
CS High-CPU Large	c5d.large	4.0 GiB	2 vCPUs	\$0.096000 hourly	\$0.188000 hourly
CS High-CPU 24xlarge	c5d.24xlarge	192.0 GiB	96 vCPUs	\$4.608000 hourly	\$9.024000 hourly
CS High-CPU Quadrate Extra Large	c5d.4xlarge	32.0 GiB	16 vCPUs	\$0.768000 hourly	\$1.504000 hourly
CS High-CPU Metal	c5d.metal	1024.0 GiB	384 vCPUs	\$4.608000 hourly	\$9.024000 hourly
CS High-CPU Double Extra Large	c5d.2xlarge	16.0 GiB	8 vCPUs	\$0.304000 hourly	\$0.752000 hourly
CS High-CPU 6xlarge	c5d.6xlarge	48.0 GiB	24 vCPUs	\$2.304000 hourly	\$4.712000 hourly
CS High-CPU 9xlarge	c5d.9xlarge	72.0 GiB	36 vCPUs	\$3.720000 hourly	\$3.840000 hourly

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

April 2, 2024
TCCS422: Operating Systems (Spring 2024)
School of Engineering and Technology, University of Washington - Tacoma
L3.11

11

MOTIVATION FOR LINUX - 2

One year cloud hosting cost:

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

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

Windows comes at a 95.8% price premium

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

April 2, 2024
TCCS422: Operating Systems (Spring 2024)
School of Engineering and Technology, University of Washington - Tacoma
L3.12

12

OBJECTIVES – 4/2

- Questions from 3/28
- **C Review Survey – Available thru Friday Apr 5**
- Student Background Survey
- Virtual Machine Survey: VM requests 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 2, 2024
TCSS422: Operating Systems (Spring 2024)
School of Engineering and Technology, University of Washington - Tacoma
L3.13

13

C REVIEW SURVEY - AVAILABLE THRU 4/5



April 2, 2024
TCSS422: Operating Systems (Spring 2024)
School of Engineering and Technology, University of Washington - Tacoma
L3.14

14

OBJECTIVES – 4/2

- Questions from 3/28
- C Review Survey – Available thru Friday Apr 5
- **Student Background Survey**
- Virtual Machine Survey: VM requests 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 2, 2024
TCSS422: Operating Systems (Spring 2024)
School of Engineering and Technology, University of Washington - Tacoma
L3.15

15

STUDENT BACKGROUND SURVEY

- **39 of 43 Responses** as of 4/1 @ ~11pm
- Current Standings:
 - Best Office Hours times so far:
 - Rank #1: Tuesday after class (>5:40pm) ✓ **(53.1%)**
 - Rank #2: Thursday after class (>5:40p) **(50%)**
 - Best lecture format:
 - Rank #1: Hybrid synchronous w/ recordings ✓ **(89.2%)**
 - Rank #2: In-person w/ recordings **(40.5%)**

April 2, 2024
TCSS422: Operating Systems (Spring 2024)
School of Engineering and Technology, University of Washington - Tacoma
L3.16

16

TCSS 422 – OFFICE HRS – SPRING 2024

- ****Tuesdays after class until 7:00pm****
Hybrid (In-person/Zoom)
 - This session will be in person in CP 229.
 - Zoom will be monitored when no student is in CP 229.
- **Thursdays after class until 7:00pm – Hybrid (In-person/Zoom)**
 - Additional office time will be held on Thursdays after class when there is high demand indicated by a busy Tuesday office hour
 - When Thursday Office Hours are planned, Zoom links will be shared via Canvas
 - Questions after class on Thursdays are always entertained even when the formal office hour is not scheduled

April 2, 2024
TCSS422: Operating Systems (Spring 2024)
School of Engineering and Technology, University of Washington - Tacoma
L3.17

17

OBJECTIVES – 4/2

- Questions from 3/28
- C Review Survey – Available thru Friday Apr 5
- Student Background Survey
- **Virtual Machine Survey: VM requests 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 2, 2024
TCSS422: Operating Systems (Spring 2024)
School of Engineering and Technology, University of Washington - Tacoma
L3.18

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/V2sg4IW1awvhFx4W8>
- VM requests have been sent to SET sys admin Siava Miasishchev for set up
- If you missed the survey, please reach out

April 2, 2024 TCCS422: Operating Systems [Spring 2024]
 School of Engineering and Technology, University of Washington - Tacoma L3.19

19

OBJECTIVES – 4/2

- Questions from 3/28
- C Review Survey – Available thru Friday Apr 5
- Student Background Survey
- Virtual Machine Survey: VM requests 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 2, 2024 TCCS422: Operating Systems [Spring 2024]
 School of Engineering and Technology, University of Washington - Tacoma L3.20

20

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 2, 2024 TCCS422: Operating Systems [Spring 2024]
 School of Engineering and Technology, University of Washington - Tacoma L3.21

21

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 2, 2024 TCCS422: Operating Systems [Spring 2024]
 School of Engineering and Technology, University of Washington - Tacoma L3.22

22



OBJECTIVES – 4/2

- Questions from 3/28
- C Review Survey – Available thru Friday Apr 5
- Student Background Survey
- Virtual Machine Survey: VM requests 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 2, 2024 TCCS422: Operating Systems [Spring 2024]
 School of Engineering and Technology, University of Washington - Tacoma L3.23

23

CHAPTER 4: PROCESSES

April 2, 2024 TCCS422: Operating Systems [Spring 2024]
 School of Engineering and Technology, University of Washington - Tacoma L3.24

24

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 2, 2024
TCSS422: Operating Systems (Spring 2024)
School of Engineering and Technology, University of Washington - Tacoma
L3.25

25

PROCESS

A process is a running program.

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

April 2, 2024
TCSS422: Operating Systems (Spring 2024)
School of Engineering and Technology, University of Washington - Tacoma
L3.26

26

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 2, 2024
TCSS422: Operating Systems (Spring 2024)
School of Engineering and Technology, University of Washington - Tacoma
L3.27

27

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 2, 2024
TCSS422: Operating Systems (Spring 2024)
School of Engineering and Technology, University of Washington - Tacoma
L3.28

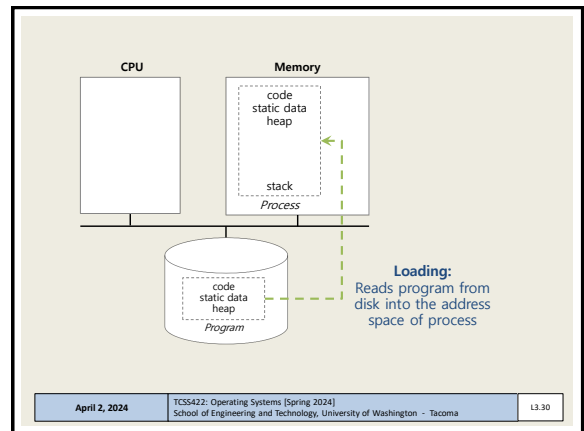
28

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 2, 2024
TCSS422: Operating Systems (Spring 2024)
School of Engineering and Technology, University of Washington - Tacoma
L3.29

29



30

WE WILL RETURN AT
4:50PM



April 2, 2024 TCSS422: Operating Systems (Spring 2024)
 School of Engineering and Technology, University of Washington - Tacoma L3.31

31

OBJECTIVES - 4/2

- Questions from 3/28
- C Review Survey – Available thru Friday Apr 5
- Student Background Survey
- Virtual Machine Survey: VM requests 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 2, 2024 TCSS422: Operating Systems (Spring 2024)
 School of Engineering and Technology, University of Washington - Tacoma L3.32

32

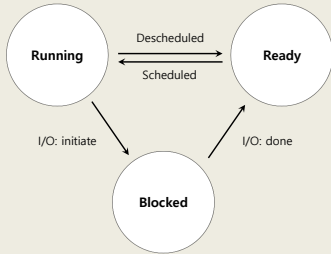
PROCESS STATES

- **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 2, 2024 TCSS422: Operating Systems (Spring 2024)
 School of Engineering and Technology, University of Washington - Tacoma L3.33

33

PROCESS STATE TRANSITIONS



April 2, 2024 TCSS422: Operating Systems (Spring 2024)
 School of Engineering and Technology, University of Washington - Tacoma L3.34

34

OBSERVING PROCESS META-DATA

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

```

    Speculation_Store_Bypass: thread vulnerable
    Cpus_allowed: ff
    Cpus_allowed_list: 0-7
    Mem_allowed: 00000000,00000001
    Mem_exclusive_allowed: 00000000,00000001
    Mem_exclusive_list: 0-7
    voluntary_ctxt_switches: 1377
    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)**

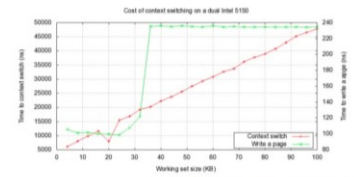
April 2, 2024 TCSS422: Operating Systems (Spring 2024)
 School of Engineering and Technology, University of Washington - Tacoma L3.35

35

CONTEXT SWITCH

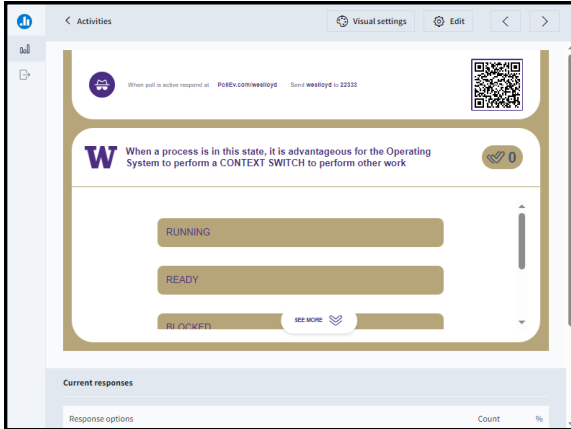
- **How long does a context switch take?**
- 10,000 to 50,000 ns (.01 to .05 ms)
- 2,000 context switches is near 100ms

Without CPU affinity



April 2, 2024 TCSS422: Operating Systems (Spring 2024)
 School of Engineering and Technology, University of Washington - Tacoma L3.36

36



37

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 2, 2024 TCSS422: Operating Systems [Spring 2024] School of Engineering and Technology, University of Washington - Tacoma L3.38

38

OBJECTIVES – 4/2

- Questions from 3/28
- C Review Survey – Available thru Friday Apr 5
- Student Background Survey
- Virtual Machine Survey: VM requests 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 2, 2024 TCSS422: Operating Systems [Spring 2024] School of Engineering and Technology, University of Washington - Tacoma L3.39

39

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 2, 2024 TCSS422: Operating Systems [Spring 2024] School of Engineering and Technology, University of Washington - Tacoma L3.40

40

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 2, 2024 TCSS422: Operating Systems [Spring 2024] School of Engineering and Technology, University of Washington - Tacoma L3.41

41

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 2, 2024 TCSS422: Operating Systems [Spring 2024] School of Engineering and Technology, University of Washington - Tacoma L3.42

42

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 2, 2024
TCSS422: Operating Systems (Spring 2024)
School of Engineering and Technology, University of Washington - Tacoma
L3.43

43

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

April 2, 2024
TCSS422: Operating Systems (Spring 2024)
School of Engineering and Technology, University of Washington - Tacoma
L3.44

44

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 #943
- **Process State**
- **/* -1 unrunnable, 0 runnable, >0 stopped: */**
- **unsigned long __state;** LOC #729
- **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 #778
 - **unsigned int time_slice;** LOC #567

April 2, 2024
TCSS422: Operating Systems (Spring 2024)
School of Engineering and Technology, University of Washington - Tacoma
L3.45

45

STRUCT TASK_STRUCT - 2

- **Address space of the process:**
 - "mm" is short for "memory map"
 - **struct mm_struct *mm;** LOC #857
- **Parent process**, that launched this one
- **struct task_struct __rcu *parent;** LOC #960
- **Child processes** (as a list)
- **struct list_head children;** LOC #965
- **Open files**
- **struct files_struct *files;** LOC #1070

April 2, 2024
TCSS422: Operating Systems (Spring 2024)
School of Engineering and Technology, University of Washington - Tacoma
L3.46

46

LINUX STRUCTURES - 2

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

April 2, 2024
TCSS422: Operating Systems (Spring 2024)
School of Engineering and Technology, University of Washington - Tacoma
L3.47

47


OBJECTIVES - 4/2

- Questions from 3/28
- C Review Survey – Available thru Friday Apr 5
- Student Background Survey
- Virtual Machine Survey: VM requests 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 2, 2024
TCSS422: Operating Systems (Spring 2024)
School of Engineering and Technology, University of Washington - Tacoma
L3.48

48

CHAPTER 5: C PROCESS API



April 2, 2024
TCSS422: Operating Systems [Spring 2024]
School of Engineering and Technology, University of Washington - Tacoma
L3.49

49

OBJECTIVES – 4/2


- Questions from 3/28
- C Review Survey – Available thru Friday Apr 5
- Student Background Survey
- Virtual Machine Survey: VM requests 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 2, 2024
TCSS422: Operating Systems [Spring 2024]
School of Engineering and Technology, University of Washington - Tacoma
L3.50

50

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 2, 2024
TCSS422: Operating Systems [Spring 2024]
School of Engineering and Technology, University of Washington - Tacoma
L3.51

51

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 2, 2024
TCSS422: Operating Systems [Spring 2024]
School of Engineering and Technology, University of Washington - Tacoma
L3.52

52

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

```

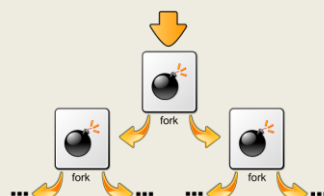
prompt> ./p1
hello world (pid:29146)
hello, I am child (pid:29147)
hello, I am parent of 29147 (pid:29146)
prompt>
    
```

- CPU scheduler determines which to run first

April 2, 2024
TCSS422: Operating Systems [Spring 2024]
School of Engineering and Technology, University of Washington - Tacoma
L3.53

53

:(){:|:&};;



April 2, 2024
TCSS422: Operating Systems [Spring 2024]
School of Engineering and Technology, University of Washington - Tacoma
L3.54

54

OBJECTIVES – 4/2


- Questions from 3/28
- C Review Survey – Available thru Friday Apr 5
- Student Background Survey
- Virtual Machine Survey: VM requests 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 2, 2024
TCCS422: Operating Systems [Spring 2024]
School of Engineering and Technology, University of Washington - Tacoma
L3.55

55

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 2, 2024
TCCS422: Operating Systems [Spring 2024]
School of Engineering and Technology, University of Washington - Tacoma
L3.56

56

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(1);
    if (rc < 0) { // fork failed; exit
        printf(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 2, 2024
TCCS422: Operating Systems [Spring 2024]
School of Engineering and Technology, University of Washington - Tacoma
L3.57

57

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 2, 2024
TCCS422: Operating Systems [Spring 2024]
School of Engineering and Technology, University of Washington - Tacoma
L3.58

58

FORK EXAMPLE

- Linux example

April 2, 2024
TCCS422: Operating Systems [Spring 2024]
School of Engineering and Technology, University of Washington - Tacoma
L3.59

59

OBJECTIVES – 4/2

- Questions from 3/28
- C Review Survey – Available thru Friday Apr 5
- Student Background Survey
- Virtual Machine Survey: VM requests 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 2, 2024
TCCS422: Operating Systems [Spring 2024]
School of Engineering and Technology, University of Washington - Tacoma
L3.60

60

exec()

- Supports running an external program by **"transferring control"**
- 6 types: `execl()`, `execlp()`, `execle()`, `execv()`, `execvp()`, `execvpe()`
- `execl()`, `execlp()`, `execle()`: `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 2, 2024
TCSS422: Operating Systems [Spring 2024]
School of Engineering and Technology, University of Washington - Tacoma
L3.61

61

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 2, 2024
TCSS422: Operating Systems [Spring 2024]
School of Engineering and Technology, University of Washington - Tacoma
L3.62

62

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 2, 2024
TCSS422: Operating Systems [Spring 2024]
School of Engineering and Technology, University of Washington - Tacoma
L3.63

63

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 2, 2024
TCSS422: Operating Systems [Spring 2024]
School of Engineering and Technology, University of Washington - Tacoma
L3.64

64

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 2, 2024
TCSS422: Operating Systems [Spring 2024]
School of Engineering and Technology, University of Washington - Tacoma
L3.65

65

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 2, 2024
TCSS422: Operating Systems [Spring 2024]
School of Engineering and Technology, University of Washington - Tacoma
L3.66

66

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 { // parent goes down this path (main)
    int wc = wait(NULL);
}
return 0;
}
```

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

April 2, 2024 | TCSS422: Operating Systems (Spring 2024)
School of Engineering and Technology, University of Washington - Tacoma | L3.67

67

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

Fork()

Exec()

Wait()

SEE MORE

Current responses

Response options

Count %

68

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 2, 2024 | TCSS422: Operating Systems (Spring 2024)
School of Engineering and Technology, University of Washington - Tacoma | L3.69

69

QUESTIONS

70