


TCSS 422: OPERATING SYSTEMS

INTRODUCTION



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

October 1, 2018

TCSS422: Operating Systems [Fall 2018]
School of Engineering and Technology, University of Washington - Tacoma

FEEDBACK FROM 9/26

- **Mostly Review to Me:**
 - 1 – 2 respondents
 - 2 – 2 respondents
 - 4 – 3 respondents
- **Mostly New to Me** **>=5 – rest of class**
 - 10 – 7 respondents

October 1, 2018

TCSS422: Operating Systems [Fall 2018]
School of Engineering and Technology, University of Washington - Tacoma

L2.2

FEEDBACK - 2

- Abstraction
- Virtualization
- Physical memory vs Virtual memory
- The OS is a resource manager, and acts almost like a brain
- Processes vs Threads: What are threads inside a process?
- “Task” not defined as process or thread
 - “Task” is seen on Linux top
- Command line
- Linux commands

October 1, 2018

TCSS422: Operating Systems [Fall 2018]
School of Engineering and Technology, University of Washington - Tacoma

L2.3

OBJECTIVES


- **Chapter 2 - Introduction to operating systems**
 - **THREE EASY PIECES:**
 - *Virtualizing the CPU (review)*
 - Virtualizing Memory
 - Virtualizing I/O
 - Operating system design goals
- Chapter 4 – Processes
- Chapter 5 – Process API
- Chapter 6 – Limited Direct Execution

October 1, 2018

TCSS422: Operating Systems [Fall 2018]
School of Engineering and Technology, University of Washington - Tacoma

L2.4

INTRODUCTION TO OPERATING SYSTEMS



October 1, 2018

TCSS422: Operating Systems [Fall 2018]
School of Engineering and Technology, University of Washington - Tacoma

L2.5

OPERATING SYSTEMS

- Responsible for:
 - Making it easy to **run** programs
 - Allowing programs to **share** memory
 - Enabling programs to **interact** with devices

OS is in charge of making sure the system operates correctly and efficiently.

October 1, 2018

TCSS422: Operating Systems [Fall 2018]
School of Engineering and Technology, University of Washington - Tacoma

L2.6

RESOURCE MANAGEMENT

- The OS is a resource manager
- Manages CPU, disk, network I/O
- Enables many programs to
 - **Share the CPU**
 - **Share the underlying physical memory (RAM)**
 - **Share physical devices**
 - Disks
 - Network Devices
 - ...

October 1, 2018

TCSS422: Operating Systems [Fall 2018]
School of Engineering and Technology, University of Washington - Tacoma

L2.7

VIRTUALIZATION

- Operating systems present **physical resources** as **virtual representations** to the programs sharing them
 - Physical resources: CPU, disk, memory, ...
 - The virtual form is “**abstract**”
 - The OS presents an illusion that each user program runs in isolation on its own hardware
 - This virtual form is general, powerful, and easy-to-use

October 1, 2018

TCSS422: Operating Systems [Fall 2018]
School of Engineering and Technology, University of Washington - Tacoma

L2.8

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

October 1, 2018

TCSS422: Operating Systems [Fall 2018]
School of Engineering and Technology, University of Washington - Tacoma

L2.9

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**

October 1, 2018

TCSS422: Operating Systems [Fall 2018]
School of Engineering and Technology, University of Washington - Tacoma

L2.10

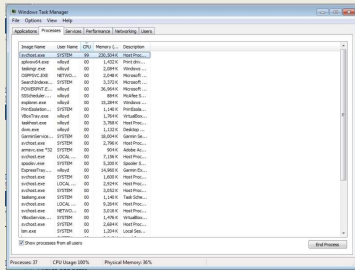
ABSTRACTION CHALLENGES

- 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

October 1, 2018 TCSS422: Operating Systems [Fall 2018]
 School of Engineering and Technology, University of Washington - Tacoma L2.11

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



```
top - 18:25:07 up 430 days, 1:03, 3 users, load average: 0.32, 0.28, 0.29
tasks: 654 total, 1 running, 653 sleeping, 0 stopped, 0 zombie
Cpu(s): 7.8%us, 0.5%sy, 0.0%ni, 91.6%id, 0.0%wa, 0.0%st, 0.7%hi, 0.0%mi
Mem: 7423728k total, 7594208k used, 746456k free, 561316k buffers
Swap: 2183572k total, 72252k used, 2176350k free, 5528350k cached

PID DOPRI  NI  VSZ  RSS  TMS  MEM%  CPU%  COMMAND
1524 root    0  6036 276 20m 5 15:2 0.0  4.32:31 postgres
8127 root    0  6036 276 20m 5 15:2 0.0  4.32:31 postgres
30796 root    0  6006 246 18m 5 15:2 0.0  0:21:50 postgres
34624 root    0  6036 226 18m 5 8.6 0.0  0:26:46 postgres
4460 root    0  15432 1712 202K 5 0.2 0.0  0:00:52 top
6280 root    0  6006 18m 5 0.7 0.0  0:07:74 postgres
7460 root    0  6036 226 20m 5 0.2 0.0  4:21:48 postgres
8220 root    0  6036 256 20m 5 0.7 0.0  4:16:41 postgres
10252 root   0  6006 20m 5 0.2 0.0  0:02:39 postgres
11913 root   0  6036 28m 20m 5 0.7 0.0  6:49:40 postgres
13512 root   0  6006 18m 5 0.2 0.0  0:06:78 postgres
17118 root   0  6006 20m 5 0.7 0.0  0:10:59 postgres
18483 root   0  6006 18m 5 0.2 0.0  0:00:78 postgres
31713 root   0  6006 18m 5 0.7 0.0  0:04:42 postgres
718 root    0  0 0 0 0 0 0 0:28:01 postgres
1500 root   0  6006 20m 18m 5 0.3 0.0  0:04:36 postgres
3504 root   0  6006 104 8420 5 0.3 0.0  0:00:16 postgres
6113 root   0  6036 22m 18m 5 0.3 0.0  4:31:34 postgres
7920 root   0  6036 22m 20m 5 0.3 0.0  4:31:34 postgres
7988 root   0  6036 22m 18m 5 0.3 0.0  4:31:34 postgres
8526 root   0  6036 22m 20m 5 0.3 0.0  4:26:40 postgres
8624 root   0  6036 22m 18m 5 0.3 0.0  4:26:40 postgres
12914 root  0  6036 22m 20m 5 0.3 0.0  3:52:32 postgres
14287 root  0  6036 22m 18m 5 0.2 0.0  1:39:50 postgres
15755 root  0  6036 22m 20m 5 0.3 0.0  1:28:40 postgres
16522 root  0  6006 18m 5 0.2 0.0  0:02:46 postgres
16460 root  0  6006 18m 5 0.3 0.0  0:02:31 postgres
16520 root  0  20916 146k 155k 0 0.0  1:00:40 top
21781 root  0  24.8g 5628k 10m 5 0.3 0.7 3:39:58.32 java
30743 root  0  6036 24m 20m 5 0.2 0.0  0:02:16 postgres
15150 root  0  6036 24m 20m 5 0.3 0.0  5:03:32 postgres
1 root    0  0 0 0 0 0 0 0:00:02 kthreadd
2 root    0  0 0 0 0 0 0 0:00:02 kthreadd
3 root    0  0 0 0 0 0 0 0:22:09.46 migration/0
4 root    0  0 0 0 0 0 0 0:00:00 ksoftirqd/0
5 root    0  0 0 0 0 0 0 0:00:00 stopper/0
6 root    0  0 0 0 0 0 0 1:07:42 watchdog/0
7 root    0  0 0 0 0 0 0 1:54:31 migration/1
8 root    0  0 0 0 0 0 0 0:00:00 stopper/1
9 root    0  0 0 0 0 0 0 2:32:32 migration/2
10 root   0  0 0 0 0 0 0 0:28:40 watchdog/2
11 root   0  0 0 0 0 0 0 0:13:03.04 migration/2
12 root   0  0 0 0 0 0 0 0:00:00 stopper/2
```

October 1, 2018 TCSS422: Operating Systems [Fall 2018]
 School of Engineering and Technology, University of Washington - Tacoma L2.12

VIRTUALIZING THE CPU - 2

■ Simple Looping C Program

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <sys/time.h>
4  #include <assert.h>
5  #include "common.h"
6
7  int
8  main(int argc, char *argv[])
9  {
10     if (argc != 2) {
11         fprintf(stderr, "usage: cpu <string>\n");
12         exit(1);
13     }
14     char *str = argv[1];
15     while (1) {
16         Spin(1); // Repeatedly checks the time and
17                 // returns once it has run for a second
18         printf("%s\n", str);
19     }
20     return 0;
21 }
```

October 1, 2018

TCSS422: Operating Systems [Fall 2018]
School of Engineering and Technology, University of Washington - Tacoma

L2.13

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...

October 1, 2018

TCSS422: Operating Systems [Fall 2018]
School of Engineering and Technology, University of Washington - Tacoma

L2.14

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!

October 1, 2018

TCSS422: Operating Systems [Fall 2018]
School of Engineering and Technology, University of Washington - Tacoma

L2.15

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

October 1, 2018

TCSS422: Operating Systems [Fall 2018]
School of Engineering and Technology, University of Washington - Tacoma

L2.16

VIRTUALIZING MEMORY - 2

■ Program to read/write memory:

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

October 1, 2018

TCSS422: Operating Systems [Fall 2018]
School of Engineering and Technology, University of Washington - Tacoma

L2.17

VIRTUALIZING MEMORY - 3

■ Output of mem.c

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

- int value stored at 00200000
- program increments int value

October 1, 2018

TCSS422: Operating Systems [Fall 2018]
School of Engineering and Technology, University of Washington - Tacoma

L2.18

VIRTUALIZING MEMORY - 4

- Multiple instances of mem.c

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

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

October 1, 2018

TCCS422: Operating Systems [Fall 2018]
School of Engineering and Technology, University of Washington - Tacoma

L2.19

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

October 1, 2018

TCCS422: Operating Systems [Fall 2018]
School of Engineering and Technology, University of Washington - Tacoma

L2.20

CONCURRENCY

Image Name	User Name	CPU	Memory (K)	Description
svchost.exe	SYSTEM	99	230,504 K	Host Proc...
explorer.exe	wlfloyd	00	1,422 K	Print dr...
taskmgr.exe	wlfloyd	00	2,084 K	Windows ...
OSPPSVIC.EVE	NETWO...	00	2,048 K	Microsoft ...
SearchIndexe...	SYSTEM	00	3,372 K	Microsoft ...
POWERPNT.E...	wlfloyd	00	36,964 K	Microsoft ...
SSScheduler...	wlfloyd	00	864 K	McAfee S...
explorer.exe	wlfloyd	00	15,284 K	Windows ...
PrintIsolatio...	SYSTEM	00	1,140 K	PrintIsola...
VBoxTray.exe	wlfloyd	00	1,764 K	VirtualBox...
taskhost.exe	wlfloyd	00	3,768 K	Host Proc...
dmv.exe	wlfloyd	00	1,132 K	Desktop ...
GameService...	SYSTEM	00	18,004 K	Game Ser...
svchost.exe	SYSTEM	00	2,796 K	Host Proc...
amsvcx.exe *32	SYSTEM	00	904 K	Adobe Ac...
svchost.exe	LOCAL ...	00	7,156 K	Host Proc...
spoolsv.exe	SYSTEM	00	5,200 K	Spooler S...
ExpressTray...	wlfloyd	00	14,960 K	Game De...
svchost.exe	SYSTEM	00	1,600 K	Host Proc...
svchost.exe	LOCAL ...	00	2,924 K	Host Proc...
svchost.exe	SYSTEM	00	3,052 K	Host Proc...
taskeng.exe	SYSTEM	00	1,140 K	Task Sche...
svchost.exe	LOCAL ...	00	9,264 K	Host Proc...
svchost.exe	NETWO...	00	3,016 K	Host Proc...
VBoxService...	SYSTEM	00	1,476 K	VirtualBox...
svchost.exe	SYSTEM	00	2,684 K	Host Proc...
lsm.exe	SYSTEM	00	1,204 K	Local Ses...

October 1, 2018

TCSS422: Operating Systems [Fall 2018]
School of Engineering and Technology, University of Washington - Tacoma

L2.21

CONCURRENCY

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

October 1, 2018

TCSS422: Operating Systems [Fall 2018]
School of Engineering and Technology, University of Washington - Tacoma

L2.21

CONCURRENCY - 2

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

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

thread.c

Listing continues ...

October 1, 2018

TCSS422: Operating Systems [Fall 2018]
School of Engineering and Technology, University of Washington - Tacoma

L2.23

CONCURRENCY - 3

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

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

October 1, 2018

TCSS422: Operating Systems [Fall 2018]
School of Engineering and Technology, University of Washington - Tacoma

L2.24

**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.

```

October 1, 2018

TCSS422: Operating Systems [Fall 2018]
School of Engineering and Technology, University of Washington - Tacoma

L2.25

CONCURRENCY - 4

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

```

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

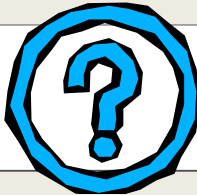
```

- Loops 100000

```

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

```



October 1, 2018

TCSS422: Operating Systems [Fall 2018]
School of Engineering and Technology, University of Washington - Tacoma

L2.26

CONCURRENCY - 5

- When loop value is large why do we not achieve 200000 ?

- C code is translated to (3) assembly code operations
 1. Load counter variable into register
 2. Increment it
 3. Store the register value back in memory

- These instructions happen concurrently and VERY FAST
- (P1 || P2) write incremented register values back to memory, While (P1 || P2) read same memory
- Memory access here is **unsynchronized (non-atomic)**
- *Some of the increments are lost*

October 1, 2018	TCSS422: Operating Systems [Fall 2018] School of Engineering and Technology, University of Washington - Tacoma	L2.27
-----------------	---	-------

W To perform parallel work, a single process may:

Launch multiple threads to execute code in parallel while sharing global data in memory	Launch multiple processes to execute code in parallel without sharing global data in memory	Both A and B	None of the above
---	---	--------------	-------------------

Start the presentation to see live content. Still no live content? Install the app or get help at PollEv.com/app Total Results

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

October 1, 2018

TCSS422: Operating Systems [Fall 2018]
School of Engineering and Technology, University of Washington - Tacoma

L2.29

PERSISTENCE

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

October 1, 2018

TCSS422: Operating Systems [Fall 2018]
School of Engineering and Technology, University of Washington - Tacoma

L2.30

PERSISTENCE - 2

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

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

PERSISTENCE - 3

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

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

October 1, 2018

TCSS422: Operating Systems [Fall 2018]
School of Engineering and Technology, University of Washington - Tacoma

L2.33

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

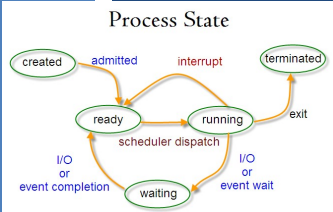


October 1, 2018


TCSS422: Operating Systems [Fall 2018]
School of Engineering and Technology, University of Washington - Tacoma

L2.34

CHAPTER 4: PROCESSES



The diagram illustrates the process state transitions. It starts with 'created' leading to 'ready' via 'admitted'. From 'ready', 'scheduler dispatch' leads to 'running'. From 'running', 'interrupt' leads to 'ready', and 'exit' leads to 'terminated'. From 'running', 'I/O or event wait' leads to 'waiting'. From 'waiting', 'I/O or event completion' leads back to 'ready'.



`/proc`

October 1, 2018 TCSS422: Operating Systems [Fall 2018]
School of Engineering and Technology, University of Washington - Tacoma L2.35

CPU VIRTUALIZING

- How should the CPU be shared?
- Time Sharing:
Run one process, pause it, run another
- How do we SWAP processes in and out of the CPU efficiently?
 - Goal is to minimize **overhead** of the swap

October 1, 2018 TCSS422: Operating Systems [Fall 2018]
School of Engineering and Technology, University of Washington - Tacoma L2.36

PROCESS

A process is a running program.

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

October 1, 2018

TCSS422: Operating Systems [Fall 2018]
School of Engineering and Technology, University of Washington - Tacoma

L2.37

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)

October 1, 2018

TCSS422: Operating Systems [Fall 2018]
School of Engineering and Technology, University of Washington - Tacoma

L2.38

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)

October 1, 2018

TCSS422: Operating Systems [Fall 2018]
School of Engineering and Technology, University of Washington - Tacoma

L2.39

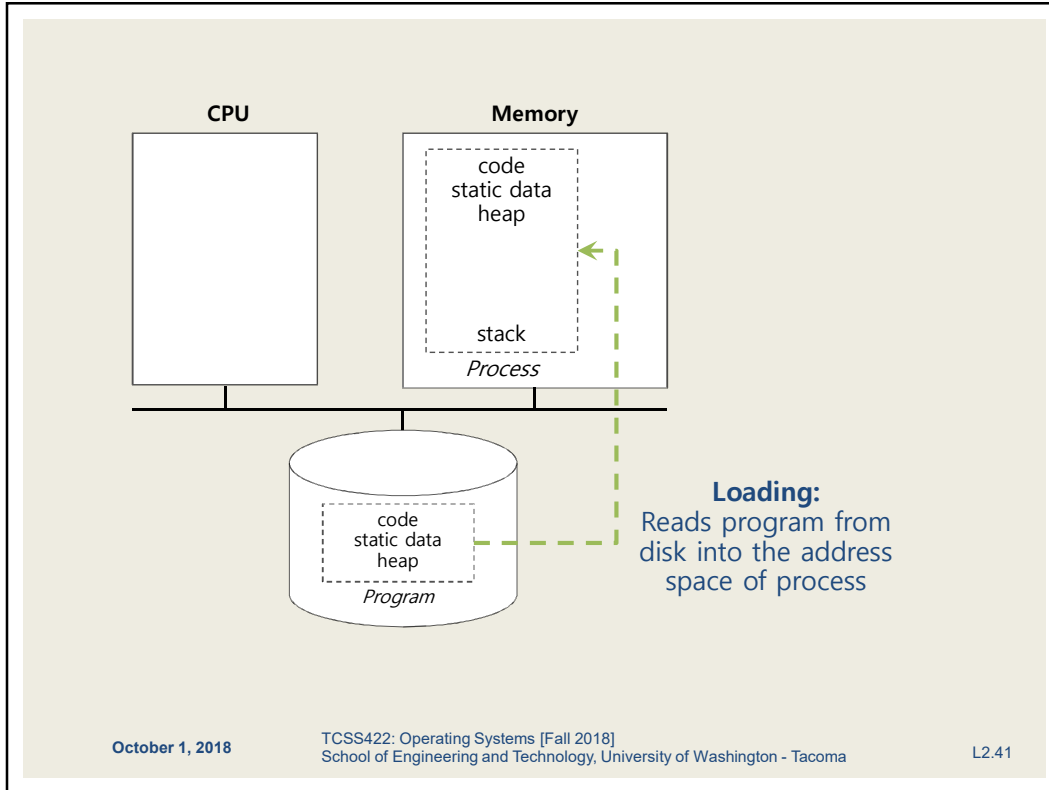
PROCESS API: CREATE

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

October 1, 2018

TCSS422: Operating Systems [Fall 2018]
School of Engineering and Technology, University of Washington - Tacoma

L2.40



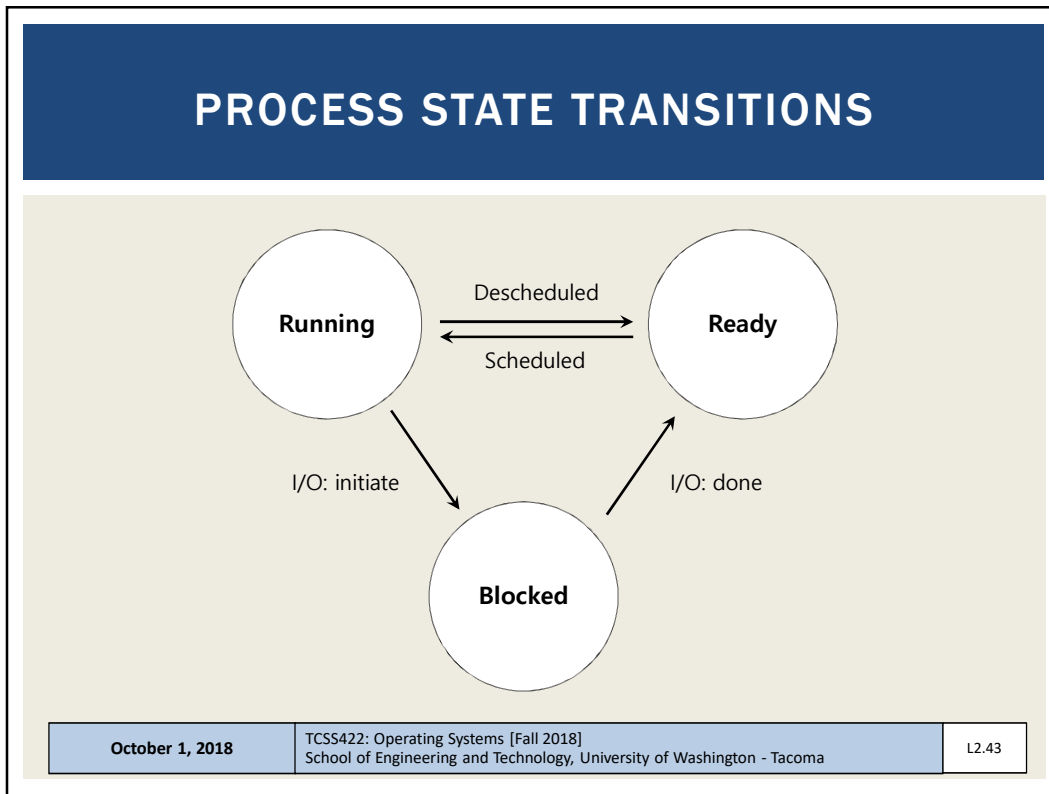
PROCESS STATES

- **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

October 1, 2018	TCSS422: Operating Systems [Fall 2018] School of Engineering and Technology, University of Washington - Tacoma	L2.42
-----------------	---	-------



- ## 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
- | | | |
|-----------------|---|-------|
| October 1, 2018 | TCSS422: Operating Systems [Fall 2018]
School of Engineering and Technology, University of Washington - Tacoma | L2.44 |
|-----------------|---|-------|

XV6 KERNEL DATA STRUCTURES

- xv6: pedagogical implementation of Linux
- Simplified structures

```
// 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 };
```

October 1, 2018

TCCS422: Operating Systems [Fall 2018]
School of Engineering and Technology, University of Washington - Tacoma

L2.45

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
};
```

October 1, 2018

TCCS422: Operating Systems [Fall 2018]
School of Engineering and Technology, University of Washington - Tacoma

L2.46

LINUX: STRUCTURES

- **struct task_struct**, equivalent to struct proc
 - Provides process description
 - Large: 10,000+ bytes
 - /usr/src/linux-headers-{kernel version}/include/linux/sched.h
 - 1227 - 1587
- **struct thread_info**, provides “context”
 - thread_info.h is at:
/usr/src/linux-headers-{kernel version}/arch/x86/include/asm/

October 1, 2018

TCSS422: Operating Systems [Fall 2018]
School of Engineering and Technology, University of Washington - Tacoma

L2.47

LINUX: THREAD_INFO

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

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

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

October 1, 2018

TCSS422: Operating Systems [Fall 2018]
School of Engineering and Technology, University of Washington - Tacoma

L2.48

LINUX STRUCTURES - 2

- List of Linux data structures:
<http://www.tldp.org/LDP/tlk/ds/ds.html>
- Description of process data structures:
<http://www.makelinux.net/books/lkd2/ch03lev1sec1>
2nd edition is online (dated from 2005):
Linux Kernel Development, 2nd edition
Robert Love
Sams Publishing

October 1, 2018

TCSS422: Operating Systems [Fall 2018]
School of Engineering and Technology, University of Washington - Tacoma

L2.49

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 None of
above the above

October 1, 2018

TCSS422: Operating Systems [Fall 2018]
Start the presentation at the beginning of the slide. School of Engineering and Technology, University of Washington - Tacoma

Total Re L2.50

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**

October 1, 2018

TCSS422: Operating Systems [Fall 2018]
School of Engineering and Technology, University of Washington - Tacoma

L2.51

QUESTIONS

