


# TCSS 422: OPERATING SYSTEMS

## The Process API & Limited Direct Execution

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



April 8, 2021      TCSS422: Operating Systems [Spring 2021]  
 School of Engineering and Technology, University of Washington - Tacoma

## OBJECTIVES – 4/8

- **Questions from 4/6**
- C Review Survey – Closes Friday Apr 9
- Assignment 0
- Chapter 5: Process API
  - fork(), wait(), exec()
- Chapter 6: Limited Direct Execution
  - Direct execution
  - Limited direct execution
  - CPU modes
  - System calls and traps
  - Cooperative multi-tasking
  - Context switching and preemptive multi-tasking

April 8, 2021      TCSS422: Operating Systems [Spring 2021]  
 School of Engineering and Technology, University of Washington - Tacoma

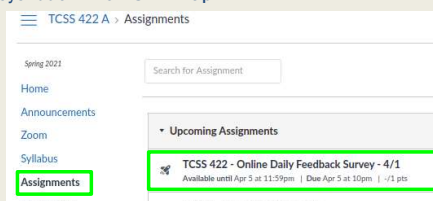
## TEXT BOOK COUPON

- 15% off textbook code: **INSPIRE15** (through Friday April 9)
- <https://www.lulu.com/shop/remzi-arpaci-dusseau-and-andrea-arpaci-dusseau/operating-systems-three-easy-pieces-softcover-version-100/paperback/product-23779877.html?page=1&pageSize=4>

April 8, 2021      TCSS422: Operating Systems [Spring 2021]  
 School of Engineering and Technology, University of Washington - Tacoma

## 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 @ 11:59p
- Thursday surveys: due ~ Mon @ 11:59p



April 8, 2021      TCSS422: Computer Operating Systems [Spring 2021]  
 School of Engineering and Technology, University of Washington - Tacoma

### TCSS 422 - Online Daily Feedback Survey - 4/1

**Quiz Instructions**

Question 1      0.5 pts

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

1   2   3   4   5   6   7   8   9   10

Mostly Review to Me      Equal New and Review      Mostly New to Me

Question 2      0.5 pts

Please rate the pace of today's class:

1   2   3   4   5   6   7   8   9   10

slow      just right      fast

April 8, 2021      TCSS422: Computer Operating Systems [Spring 2021]  
 School of Engineering and Technology, University of Washington - Tacoma

## MATERIAL / PACE

- Please classify your perspective on material covered in today's class (50 respondents):
- 1-mostly review, 5-equal new/review, 10-mostly new
- **Average – 6.91 (↓ - previous 6.92)**
- Please rate the pace of today's class:
- 1-slow, 5-just right, 10-fast
- **Average – 5.67 (↑ - previous 5.57)**

April 8, 2021      TCSS422: Computer Operating Systems [Spring 2021]  
 School of Engineering and Technology, University of Washington - Tacoma

### FEEDBACK

- **Can we say the number of tasks is the number of processes?**
  - YES
  - Processes are identified as "tasks" in top
- **How are memory leaks taken care of when we close a program?**
  - When a program is closed, all memory is freed
- **Does the OS keep track of what parts of memory were being used by a program even if the program itself dereferences it?**
  - Unlike Java, C does not have automatic garbage collection
  - A programmer releases malloc'd memory using the free() function
  - The OS tracks the location of the heap. The data may still reside on the heap but it is no longer referenced. Allocating new variables on the heap may result in finding the old data. The values can be seen if the new variables are not initialized.

April 8, 2021	TCCS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma	L4.7
---------------	---	------

### FEEDBACK - 2

- **I'm confused about the differences between the READY and BLOCKED process states**
- **BLOCKED:** can not run, waiting on I/O to finish
- **READY:** is able to run, but not yet scheduled on the CPU

April 8, 2021	TCCS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma	L4.8
---------------	---	------

### FEEDBACK - 3

- **Can you fork a process multiple times?**
  - Yes
- **Can a parent have more than one child?**
  - Yes
- **Can a child have a child and become a parent?**
  - Yes
- **I saw in a diagram on pg 8 of the lecture 3 slides that you can keep forking but if the child process's PID is 0 then how can you make a new child with a process PID 0 and then differentiate which is the parent?**
  - The child can call getpid() to discover its true PID
  - When the child calls fork, it will also receive back its PID

April 8, 2021	TCCS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma	L4.9
---------------	---	------

### FEEDBACK - 4

- **How is prioritization between the parent and child processes done after the call to fork() ?**
  - The operating system schedules which process goes next
  - If the computer has multiple cores, they may be scheduled to run at the same time
  - The programmer can enforce execution ordering by using the wait() API
- **Is there a similar command to fork() that can create a child process without also copying memory, registers, and the program counter?**
  - Yes, these are threads, and the API is pthread\_create()

April 8, 2021	TCCS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma	L4.10
---------------	---	-------

### FEEDBACK - 5

- **What is overhead?**
  - Question for the class...
- **(Assignment questions with example)**  
 As I researched and understood, we have several command options to find the number of processes (question #1 ex: "ps" or "top").
- **Is it okay to use any commands from these options, or do you expect specific commands which were mentioned from the lecture?**
  - Any command can be used

April 8, 2021	TCCS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma	L4.11
---------------	---	-------

### OBJECTIVES – 4/8

- Questions from 4/6
- **C Review Survey – Closes Friday Apr 9**
- Assignment 0
- Chapter 5: Process API
  - fork(), wait(), exec()
- Chapter 6: Limited Direct Execution
  - Direct execution
  - Limited direct execution
  - CPU modes
  - System calls and traps
  - Cooperative multi-tasking
  - Context switching and preemptive multi-tasking

April 8, 2021	TCCS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma	L4.12
---------------	---	-------

## OBJECTIVES – 4/8

- Questions from 4/6
- C Review Survey – Closes Friday Apr 9
- **Assignment 0**
- Chapter 5: Process API
  - fork(), wait(), exec()
- Chapter 6: Limited Direct Execution
  - Direct execution
  - Limited direct execution
  - CPU modes
  - System calls and traps
  - Cooperative multi-tasking
  - Context switching and preemptive multi-tasking


April 8, 2021
TCCS422: Operating Systems [Spring 2021]  
School of Engineering and Technology, University of Washington - Tacoma
L4.13

## OBJECTIVES – 4/8

- Questions from 4/6
- C Review Survey – Closes Friday Apr 9
- Assignment 0
- Chapter 5: Process API
  - **fork()**, wait(), exec()
- Chapter 6: Limited Direct Execution
  - Direct execution
  - Limited direct execution
  - CPU modes
  - System calls and traps
  - Cooperative multi-tasking
  - Context switching and preemptive multi-tasking

April 8, 2021
TCCS422: Operating Systems [Spring 2021]  
School of Engineering and Technology, University of Washington - Tacoma
L4.14


# CHAPTER 5: C PROCESS API



April 8, 2021
TCCS422: Operating Systems [Spring 2021]  
School of Engineering and Technology, University of Washington - Tacoma
L4.15

## 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, 2021
TCCS422: Operating Systems [Spring 2021]  
School of Engineering and Technology, University of Washington - Tacoma
L4.16

## 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, 2021
TCCS422: Operating Systems [Spring 2021]  
School of Engineering and Technology, University of Washington - Tacoma
L4.17

## 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, 2021
TCCS422: Operating Systems [Spring 2021]  
School of Engineering and Technology, University of Washington - Tacoma
L4.18

## :(){ :|: & }::

fork

fork

fork

fork

April 8, 2021    TCSS422: Operating Systems [Spring 2021]  
 School of Engineering and Technology, University of Washington - Tacoma    L4.19

## OBJECTIVES – 4/8

- Questions from 4/6
- C Review Survey – Closes Friday Apr 9
- Assignment 0
- Chapter 5: Process API
  - fork(), wait(), exec()
- Chapter 6: Limited Direct Execution
  - Direct execution
  - Limited direct execution
  - CPU modes
  - System calls and traps
  - Cooperative multi-tasking
  - Context switching and preemptive multi-tasking

April 8, 2021    TCSS422: Operating Systems [Spring 2021]  
 School of Engineering and Technology, University of Washington - Tacoma    L4.20

## 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, 2021    TCSS422: Operating Systems [Spring 2021]  
 School of Engineering and Technology, University of Washington - Tacoma    L4.21

## 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, 2021    TCSS422: Operating Systems [Spring 2021]  
 School of Engineering and Technology, University of Washington - Tacoma    L4.22

## 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, 2021    TCSS422: Operating Systems [Spring 2021]  
 School of Engineering and Technology, University of Washington - Tacoma    L4.23

## FORK EXAMPLE

- Linux example

April 8, 2021    TCSS422: Operating Systems [Spring 2021]  
 School of Engineering and Technology, University of Washington - Tacoma    L4.24

## OBJECTIVES – 4/8

- Questions from 4/6
- C Review Survey – Closes Friday Apr 9
- Assignment 0
- Chapter 5: Process API
  - fork(), wait(), **exec()**
- Chapter 6: Limited Direct Execution
  - Direct execution
  - Limited direct execution
  - CPU modes
  - System calls and traps
  - Cooperative multi-tasking
  - Context switching and preemptive multi-tasking

April 8, 2021 TCCS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma L4.25

## 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, 2021 TCCS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma L4.26

## 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, 2021 TCCS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma L4.27

## 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, 2021 TCCS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma L4.28

## 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, 2021 TCCS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma L4.29

## 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, 2021 TCCS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma L4.30

### 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_IRWGO
read, write, execute/search by others
S_IROTH
read permission, others
S_IWOTH
write permission, others
    
```

April 8, 2021    TCSS422: Operating Systems [Spring 2021]  
 School of Engineering and Technology, University of Washington - Tacoma    L4.31

### 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
execlp(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, 2021    TCSS422: Operating Systems [Spring 2021]  
 School of Engineering and Technology, University of Washington - Tacoma    L4.32

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

Fork()    Exec()    Wait()    None of the above    All of the above

Start the presentation to see live content. Still no live content? Install the app or get help at [PollEv.com/app](http://PollEv.com/app)    Total: 1/20/21

### 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

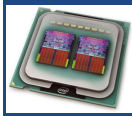
April 8, 2021    TCSS422: Operating Systems [Spring 2021]  
 School of Engineering and Technology, University of Washington - Tacoma    L4.34

### OBJECTIVES – 4/8

- Questions from 4/6
- C Review Survey – Closes Friday Apr 9
- Assignment 0
- Chapter 5: Process API
  - fork(), wait(), exec()
  - **Chapter 6: Limited Direct Execution**
    - Direct execution
    - Limited direct execution
    - CPU modes
    - System calls and traps
    - Cooperative multi-tasking
    - Context switching and preemptive multi-tasking

April 8, 2021    TCSS422: Operating Systems [Spring 2021]  
 School of Engineering and Technology, University of Washington - Tacoma    L4.35

## CH. 6: LIMITED DIRECT EXECUTION



April 8, 2021    TCSS422: Operating Systems [Spring 2021]  
 School of Engineering and Technology, University of Washington - Tacoma    L3.36



### CONTEXT SWITCHING OVERHEAD

Context Switching

Total cost of context switching

Multitasking

vs. Multitasking with context switching

Sequential

Overhead

Time

April 8, 2021 TCCS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma L4.43

## WE WILL RETURN AT 5:10PM

April 8, 2021 TCCS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma L4.44

### OBJECTIVES – 4/8

- Questions from 4/6
- C Review Survey – Closes Friday Apr 9
- Assignment 0
- Chapter 5: Process API
  - fork(), wait(), exec()
- Chapter 6: Limited Direct Execution
  - Direct execution
  - **Limited direct execution**
  - CPU modes
  - System calls and traps
  - Cooperative multi-tasking
  - Context switching and preemptive multi-tasking

April 8, 2021 TCCS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma L4.45

### LIMITED DIRECT EXECUTION

- OS implements LDE to support time/resource sharing
- Limited direct execution means “only limited” processes can execute **DIRECTLY** on the CPU in **trusted** mode
- TRUSTED means the process is trusted, and it can do anything... (e.g. it is a system / kernel level process)
- Enabled by **protected (safe) control transfer**
- CPU supported context switch
- Provides data isolation

April 8, 2021 TCCS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma L4.46

### OBJECTIVES – 4/8

- Questions from 4/6
- C Review Survey – Closes Friday Apr 9
- Assignment 0
- Chapter 5: Process API
  - fork(), wait(), exec()
- Chapter 6: Limited Direct Execution
  - Direct execution
  - Limited direct execution
  - **CPU modes**
  - System calls and traps
  - Cooperative multi-tasking
  - Context switching and preemptive multi-tasking

April 8, 2021 TCCS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma L4.47

### CPU MODES

- Utilize CPU Privilege Rings (Intel x86)
  - rings 0 (kernel), 1 (VM kernel), 2 (unused), 3 (user)

access ← no access

- **User mode:**  
Application is running, but w/o direct I/O access
- **Kernel mode:**  
OS kernel is running performing restricted operations

April 8, 2021 TCCS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma L4.48



## CPU MODES

- **User mode: ring 3 - untrusted**
  - Some instructions and registers are disabled by the CPU
  - Exception registers
  - HALT instruction
  - MMU instructions
  - OS memory access
  - I/O device access
- **Kernel mode: ring 0 – trusted**
  - All instructions and registers enabled

April 8, 2021
TCCS422: Operating Systems [Spring 2021]  
School of Engineering and Technology, University of Washington - Tacoma
L4.49

## OBJECTIVES – 4/8

- Questions from 4/6
- C Review Survey – Closes Friday Apr 9
- Assignment 0
- Chapter 5: Process API
  - fork(), wait(), exec()
- Chapter 6: Limited Direct Execution
  - Direct execution
  - Limited direct execution
  - CPU modes
  - **System calls and traps**
  - Cooperative multi-tasking
  - Context switching and preemptive multi-tasking

April 8, 2021
TCCS422: Operating Systems [Spring 2021]  
School of Engineering and Technology, University of Washington - Tacoma
L4.50

## SYSTEM CALLS

- Implement restricted “OS” operations
- Kernel exposes key functions through an API:
  - Device I/O (e.g. file I/O)
  - Task swapping: context switching between processes
  - Memory management/allocation: malloc()
  - Creating/destroying processes

April 8, 2021
TCCS422: Operating Systems [Spring 2021]  
School of Engineering and Technology, University of Washington - Tacoma
L4.51

## TRAPS: SYSTEM CALLS, EXCEPTIONS, INTERRUPTS

- Trap: any transfer to kernel mode
- Three kinds of traps
  - **System call:** (planned) user → kernel
    - SYSCALL for I/O, etc.
  - **Exception:** (error) user → kernel
    - Div by zero, page fault, page protection error
  - **Interrupt:** (event) user → kernel
    - Non-maskable vs. maskable
    - Keyboard event, network packet arrival, timer ticks
    - Memory parity error (ECC), hard drive failure

April 8, 2021
TCCS422: Operating Systems [Spring 2021]  
School of Engineering and Technology, University of Washington - Tacoma
L4.52

## EXCEPTION TYPES

Exception type	Synchronous vs. asynchronous	User request vs. coerced	User maskable vs. nonmaskable	Within vs. between instructions	Resume vs. terminate
I/O device request	Asynchronous	Coerced	Nonmaskable	Between	Resume
Invoke operating system	Synchronous	User request	Nonmaskable	Between	Resume
Trapping instruction execution	Synchronous	User request	User maskable	Between	Resume
Illegal op	Synchronous	User request	User maskable	Between	Resume
Integer arithmetic overflow	Synchronous	Coerced	User maskable	Within	Resume
Floating point arithmetic overflow or underflow	Synchronous	Coerced	User maskable	Within	Resume
Page fault	Synchronous	Coerced	Nonmaskable	Within	Resume
Illegal memory accesses	Synchronous	Coerced	User maskable	Within	Resume
Memory protection violation	Synchronous	Coerced	Nonmaskable	Within	Resume
Using undefined instruction	Synchronous	Coerced	Nonmaskable	Within	Terminate
Hardware malfunction	Asynchronous	Coerced	Nonmaskable	Within	Terminate
Power failure	Asynchronous	Coerced	Nonmaskable	Within	Terminate

April 8, 2021
TCCS422: Operating Systems [Spring 2021]  
School of Engineering and Technology, University of Washington - Tacoma
L4.53

April 8, 2021
TCCS422: Operating Systems [Spring 2021]  
School of Engineering and Technology, University of Washington - Tacoma
L4.54

OS @ boot (kernel mode)	Hardware	
initialize trap table	remember address of ... syscall handler	
OS @ run (kernel mode)	Hardware	Program (user mode)
Create entry for process list Allocate memory for program Load program into memory Setup user stack with argv Fill kernel stack with user PC		
	move to kernel mode jump to trap handler	
Handle trap Do work of syscall return-from-trap	restore regs from kernel stack move to user mode jump to PC after trap	
		return from main trap (via exit())
Free memory of process Remove from process list		

**Computer BOOT Sequence:  
OS with Limited Direct Execution**

April 8, 2021 TCS5422: Operating Systems [Spring 2021]  
 School of Engineering and Technology, University of Washington - Tacoma L4.55

## OBJECTIVES – 4/8

- Questions from 4/6
- C Review Survey – Closes Friday Apr 9
- Assignment 0
- Chapter 5: Process API
  - fork(), wait(), exec()
- Chapter 6: Limited Direct Execution
  - Direct execution
  - Limited direct execution
  - CPU modes
  - System calls and traps
  - Cooperative multi-tasking**
  - Context switching and preemptive multi-tasking

April 8, 2021 TCS5422: Operating Systems [Spring 2021]  
 School of Engineering and Technology, University of Washington - Tacoma L4.56

## MULTITASKING

- How/when should the OS regain control of the CPU to switch between processes?
- Cooperative multitasking (mostly pre 32-bit)
  - < Windows 95, Mac OSX
  - Opportunistic: running programs must give up control
    - User programs must call a special **yield** system call
    - When performing I/O
    - Illegal operations
- (POLLEV)

What problems could you see with this approach?

April 8, 2021 TCS5422: Operating Systems [Spring 2021]  
 School of Engineering and Technology, University of Washington - Tacoma L4.57

## MULTITASKING

- How/when should the OS regain control of the CPU to switch between processes?
- Cooperative multitasking (mostly pre 32-bit)
  - < A process gets stuck in an infinite loop. → **Reboot the machine**
  - Opportunistic: running programs must give up control
    - When performing I/O
    - Illegal operations
- (POLLEV)

What problems could you see with this approach?

April 8, 2021 TCS5422: Operating Systems [Spring 2021]  
 School of Engineering and Technology, University of Washington - Tacoma L4.58

## What problems exist for regaining the control of the CPU with cooperative multitasking OSes?

Start the presentation to see live content. Still no live content? Install the app or get help at [PollEv.com/app](http://PollEv.com/app) Total Results

## QUESTION: MULTITASKING

- What problems exist for regaining the control of the CPU with cooperative multitasking OSes?

April 8, 2021 TCS5422: Operating Systems [Spring 2021]  
 School of Engineering and Technology, University of Washington - Tacoma L4.60

## MULTITASKING - 2

- Preemptive multitasking (32 & 64 bit OSes)
- >= Mac OSX, Windows 95+
- Timer interrupt
  - Raised at some regular interval (in ms)
  - Interrupt handling
    1. Current program is halted
    2. Program states are saved
    3. OS Interrupt handler is run (kernel mode)
- (PollEV) What is a good interval for the timer interrupt?

April 8, 2021	TCCS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma	L4.61
---------------	---	-------

## MULTITASKING - 2

- Preemptive multitasking (32 & 64 bit OSes)
- >= Mac OSX, Windows 95+
- Timer interrupt
  - Raised at some regular interval (in ms)
  - Interrupt handling
    1. Current program is halted
    2. Program states are saved
    3. OS Interrupt handler is run (kernel mode)
- (PollEV) What is a good interval for the timer interrupt?

A timer interrupt gives OS the ability to run again on a CPU.

April 8, 2021	TCCS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma	L4.62
---------------	---	-------

W

For an OS that uses a system timer to force arbitrary context switches to share the CPU, what is a good value (in seconds) for the timer interrupt?

April 8, 2021	TCCS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma	L4.63
---------------	---	-------

## QUESTION: TIME SLICE

- For an OS that uses a system timer to force arbitrary context switches to share the CPU, what is a good value (in seconds) for the timer interrupt?

April 8, 2021	TCCS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma	L4.64
---------------	---	-------

## OBJECTIVES – 4/8

- Questions from 4/6
- C Review Survey – Closes Friday Apr 9
- Assignment 0
- Chapter 5: Process API
  - fork(), wait(), exec()
- Chapter 6: Limited Direct Execution
  - Direct execution
  - Limited direct execution
  - CPU modes
  - System calls and traps
  - Cooperative multi-tasking
  - Context switching and preemptive multi-tasking

April 8, 2021	TCCS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma	L4.65
---------------	---	-------

## CONTEXT SWITCH

- Preemptive multitasking initiates “trap” into the OS code to determine:
  - Whether to continue running the **current process**, or switch to a **different one**.
  - If the decision is made to switch, the OS performs a context switch swapping out the current process for a new one.

April 8, 2021	TCCS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma	L4.66
---------------	---	-------

## CONTEXT SWITCH - 2

1. Save register values of the current process to its kernel stack
  - General purpose registers
  - PC: program counter (instruction pointer)
  - kernel stack pointer
2. Restore soon-to-be-executing process from its kernel stack
3. Switch to the kernel stack for the soon-to-be-executing process

April 8, 2021
TCSS422: Operating Systems [Spring 2021]  
School of Engineering and Technology, University of Washington - Tacoma
L4.67

The diagram illustrates the OS boot and run phases. In the 'OS @ boot (kernel mode)' phase, the OS performs two main actions: 'initialize trap table' (remembering addresses of syscall and timer handlers) and 'start interrupt timer' (starting a timer to interrupt the CPU in X ms). In the 'OS @ run (kernel mode)' phase, 'Process A' is running in 'Program (user mode)'. A 'timer interrupt' occurs, saving registers(A) to k-stack(A) and moving to kernel mode to jump to a trap handler. The trap handler then calls a switch() routine: save regs(A) to proc-struct(A), restore regs(B) from proc-struct(B), switch to k-stack(B), and return-from-trap (into B). Finally, registers(B) are restored from k-stack(B), the system moves to user mode, and jumps to B's PC to start 'Process B'.

April 8, 2021
TCSS422: Operating Systems [Spring 2021]  
School of Engineering and Technology, University of Washington - Tacoma
L4.68

This slide shows a detailed view of the context switch process. It includes the same diagram as slide L4.68 but with a large blue box overlaid in the center containing the text 'Context Switch'. The text in the box is white and bold.

April 8, 2021
TCSS422: Operating Systems [Spring 2021]  
School of Engineering and Technology, University of Washington - Tacoma
L4.69

## INTERRUPTED INTERRUPTS

- What happens if during an interrupt (trap to kernel mode), another interrupt occurs?
- Linux
  - < 2.6 kernel: non-preemptive kernel
  - >= 2.6 kernel: preemptive kernel

April 8, 2021
TCSS422: Operating Systems [Spring 2021]  
School of Engineering and Technology, University of Washington - Tacoma
L4.70

## PREEMPTIVE KERNEL

- Use "locks" as markers of regions of non-preemptibility (non-maskable interrupt)
- Preemption counter (`preempt_count`)
  - begins at zero
  - increments for each lock acquired (not safe to preempt)
  - decrements when locks are released
- Interrupt can be interrupted when `preempt_count=0`
  - It is safe to preempt (maskable interrupt)
  - the interrupt is more important

April 8, 2021
TCSS422: Operating Systems [Spring 2021]  
School of Engineering and Technology, University of Washington - Tacoma
L4.71

# CHAPTER 7- SCHEDULING: INTRODUCTION

April 8, 2021
TCSS422: Operating Systems [Spring 2021]  
School of Engineering and Technology, University of Washington - Tacoma
L4.72

### OBJECTIVES – 4/8

- Chapter 7: Scheduling Introduction
  - Scheduling metrics
    - Turnaround time, Jain's Fairness Index, Response time
    - FIFO, SJF, STCF, RR schedulers

April 8, 2021
TCCS422: Operating Systems [Spring 2021]  
School of Engineering and Technology, University of Washington - Tacoma
L4.73

### SCHEDULING METRICS

- **Metrics:** A standard measure to quantify to what degree a system possesses some property. Metrics provide *repeatable* techniques to quantify and compare systems.
- **Measurements** are the numbers derived from the application of metrics
- Scheduling Metric #1: **Turnaround time**
- The time at which the job completes minus the time at which the job arrived in the system

$$T_{turnaround} = T_{completion} - T_{arrival}$$

- How is turnaround time different than execution time?

April 8, 2021
TCCS422: Operating Systems [Spring 2021]  
School of Engineering and Technology, University of Washington - Tacoma
L4.74

### SCHEDULING METRICS - 2

- Scheduling Metric #2: **Fairness**
  - Jain's fairness index
  - Quantifies if jobs receive a fair share of system resources

$$J(x_1, x_2, \dots, x_n) = \frac{(\sum_{i=1}^n x_i)^2}{n \cdot \sum_{i=1}^n x_i^2}$$

- n processes
- $x_i$  is time share of each process
- worst case =  $1/n$
- best case = 1
- Consider  $n=3$ , worst case = .333, best case=1
- With  $n=3$  and  $x_1=.2, x_2=.7, x_3=.1$ , fairness=.62
- With  $n=3$  and  $x_1=.33, x_2=.33, x_3=.33$ , fairness=1

April 8, 2021
TCCS422: Operating Systems [Spring 2021]  
School of Engineering and Technology, University of Washington - Tacoma
L4.75

### OBJECTIVES – 4/8

- Chapter 7: Scheduling Introduction
  - Scheduling metrics
    - Turnaround time, Jain's Fairness Index, Response time
    - **FIFO** SJF, STCF, RR schedulers

April 8, 2021
TCCS422: Operating Systems [Spring 2021]  
School of Engineering and Technology, University of Washington - Tacoma
L4.76

### SCHEDULERS

- FIFO: first in, first out
  - Very simple, easy to implement
- Consider
  - 3 x 10sec jobs, arrival: A B C, duration 10 sec each

$$\text{Average turnaround time} = \frac{10 + 20 + 30}{3} = 20 \text{ sec}$$

April 8, 2021
TCCS422: Operating Systems [Spring 2021]  
School of Engineering and Technology, University of Washington - Tacoma
L4.77

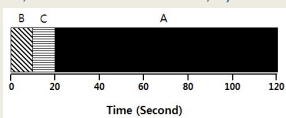
### OBJECTIVES – 4/8

- Chapter 7: Scheduling Introduction
  - Scheduling metrics
    - Turnaround time, Jain's Fairness Index, Response time
    - FIFO, **SJF** STCF, RR schedulers

April 8, 2021
TCCS422: Operating Systems [Spring 2021]  
School of Engineering and Technology, University of Washington - Tacoma
L4.78

### SJF: SHORTEST JOB FIRST

- Given that we know execution times in advance:
  - Run in order of duration, shortest to longest
  - Non preemptive scheduler
  - This is not realistic
  - Arrival: A B C, duration a=100 sec, b/c=10sec

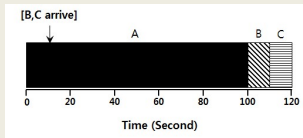


$$\text{Average turnaround time} = \frac{10 + 20 + 120}{3} = 50 \text{ sec}$$

April 8, 2021
TCCS422: Operating Systems [Spring 2021]  
School of Engineering and Technology, University of Washington - Tacoma
L4.79

### SJF: WITH RANDOM ARRIVAL

- If jobs arrive at any time: duration a=100s, b/c=10s
- A @ t=0sec, B @ t=10sec, C @ t=10sec



$$\text{Average turnaround time} = \frac{100 + (110 - 10) + (120 - 10)}{3} = 103.33 \text{ sec}$$

April 8, 2021
TCCS422: Operating Systems [Spring 2021]  
School of Engineering and Technology, University of Washington - Tacoma
L4.80

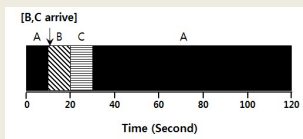
### OBJECTIVES - 4/8

- Chapter 7: Scheduling Introduction
  - Scheduling metrics
    - Turnaround time, Jain's Fairness Index, Response time
  - FIFO, SJF, **STCF**, RR schedulers

April 8, 2021
TCCS422: Operating Systems [Spring 2021]  
School of Engineering and Technology, University of Washington - Tacoma
L4.81

### STCF - 2

- Consider: duration a=100sec, b/c=10sec
- A<sub>len</sub>=100 A<sub>arrival</sub>=0
- B<sub>len</sub>=10, B<sub>arrival</sub>=10, C<sub>len</sub>=10, C<sub>arrival</sub>=10



$$\text{Average turnaround time} = \frac{(120 - 0) + (20 - 10) + (30 - 10)}{3} = 50 \text{ sec}$$

April 8, 2021
TCCS422: Operating Systems [Spring 2021]  
School of Engineering and Technology, University of Washington - Tacoma
L4.82

### SCHEDULING METRICS - 3

- Scheduling Metric #3: **Response Time**
- Time from when job arrives until it starts execution

$$T_{\text{response}} = T_{\text{first run}} - T_{\text{arrival}}$$

- STCF, SJF, FIFO
  - can perform poorly with respect to response time

What scheduling algorithm(s) can help minimize response time?


April 8, 2021
TCCS422: Operating Systems [Spring 2021]  
School of Engineering and Technology, University of Washington - Tacoma
L4.83

### OBJECTIVES - 4/8

- Chapter 7: Scheduling Introduction
  - Scheduling metrics
    - Turnaround time, Jain's Fairness Index, Response time
  - FIFO, SJF, STCF, **RR schedulers**

April 8, 2021
TCCS422: Operating Systems [Spring 2021]  
School of Engineering and Technology, University of Washington - Tacoma
L4.84

## RR: ROUND ROBIN



- Run each job awhile, then switch to another distributing the CPU evenly (fairly)
- Scheduling Quantum is called a time slice
- Time slice is a multiple of the time period.

RR is fair, but performs poorly on metrics such as turnaround time

Process	Burst Time
P1	12

**Round Robin scheduling algorithm Gantt chart**

P1	P2	P3	P4	P5	P1	P2	P4	P1	
0	5	10	14	19	24	29	32	37	39

Scheduling Quantum = 5 seconds

April 8, 2021 TCCS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma L4.85

## RR EXAMPLE

- ABC arrive at time=0, each run for 5 seconds

SJF (Bad for Response Time)

$T_{average\ response} = \frac{0 + 5 + 10}{3} = 5sec$

RR with a time-slice of 1sec (Good for Response Time)

$T_{average\ response} = \frac{0 + 1 + 2}{3} = 1sec$

OVERHEAD not considered

April 8, 2021 TCCS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma L4.86

## ROUND ROBIN: TRADEOFFS

Short Time Slice

Fast Response Time

High overhead from context switching

Long Time Slice

Slow Response Time

Low overhead from context switching

- Time slice impact:
  - Turnaround time (for earlier example):  $ts(1,2,3,4,5) = 14, 14, 13, 14, 10$
  - Fairness: round robin is always fair,  $J=1$

April 8, 2021 TCCS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma L4.87

## SCHEDULING WITH I/O

- STCF scheduler
  - A: CPU=50ms, I/O=40ms, 10ms intervals
  - B: CPU=50ms, I/O=0ms
  - Consider A as 10ms subjobs (CPU, then I/O)
- Without considering I/O:
  - CPU utilization =  $100/140 = 71\%$
  - Poor Use of Resources

April 8, 2021 TCCS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma L4.88

## SCHEDULING WITH I/O - 2

- When a job initiates an I/O request
  - A is blocked, waits for I/O to complete, frees CPU
  - STCF scheduler assigns B to CPU
- When I/O completes → raise interrupt
  - Unblock A, STCF goes back to executing A: (10ms sub-job)

Overlap Allows Better Use of Resources

April 8, 2021 TCCS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma L4.89

## Which scheduler, thus far, best address fairness and average response time of jobs?

Respond at [PollEv.com/wesleylloyd641](https://poll.evl.com/wesleylloyd641)

Text WESLEYLLOYD641 to 22333 once to join, then 1, 2, 3, 4, 5...

First In - First Out (FIFO)	1
Shortest Job First (SJF)	2
Shortest Time to Completion First (STCF)	3
Round Robin	4
None of the Above	5
All of the Above	6

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

### QUESTION: SCHEDULING FAIRNESS

- Which scheduler, this far, best addresses fairness and average response time of jobs?
- First In – First Out (FIFO)
- Shortest Job First (SJF)
- Shortest Time to Completion First (STCF)
- Round Robin (RR)
- None of the Above
- All of the Above

April 8, 2021
TCCS422: Operating Systems [Spring 2021]  
School of Engineering and Technology, University of Washington - Tacoma
L4.91

### SCHEDULING METRICS

- Consider Three jobs (A, B, C) that require:  $time_A=400ms$ ,  $time_B=100ms$ , and  $time_C=200ms$
- All jobs arrive at  $time=0$  in the sequence of A B C.
- Draw a scheduling graph to help compute the **average response time (ART)** and **average turnaround time (ATT)** scheduling metrics for the FIFO scheduler.

Example:

April 8, 2021
TCCS422: Operating Systems [Spring 2021]  
School of Engineering and Technology, University of Washington - Tacoma
L4.92

When poll is active, respond at [PollEv.com/wesleylloyd641](https://poll-ev.com/wesleylloyd641)  
 Text WESLEYLLOYD641 to 22333 once to join

## What is the Average Response Time of the FIFO scheduler?

Start the presentation to see live content. Still no live content? Install the app or get help at [PollEv.com/app](https://poll-ev.com/app)

When poll is active, respond at [PollEv.com/wesleylloyd641](https://poll-ev.com/wesleylloyd641)  
 Text WESLEYLLOYD641 to 22333 once to join

## What is the Average Turnaround Time of the FIFO scheduler?

Start the presentation to see live content. Still no live content? Install the app or get help at [PollEv.com/app](https://poll-ev.com/app)

### SCHEDULING METRICS

- Consider Three jobs (A, B, C) that require:  $time_A=400ms$ ,  $time_B=100ms$ , and  $time_C=200ms$
- All jobs arrive at  $time=0$  in the sequence of A B C.
- Draw a scheduling graph to help compute the **average response time (ART)** and **average turnaround time (ATT)** scheduling metrics for the SJF scheduler.

Example:

April 8, 2021
TCCS422: Operating Systems [Spring 2021]  
School of Engineering and Technology, University of Washington - Tacoma
L4.95

When poll is active, respond at [PollEv.com/wesleylloyd641](https://poll-ev.com/wesleylloyd641)  
 Text WESLEYLLOYD641 to 22333 once to join

## What is the Average Response Time of the Shortest Job First Scheduler?

Start the presentation to see live content. Still no live content? Install the app or get help at [PollEv.com/app](https://poll-ev.com/app)



When poll is active, respond at [PollEv.com/wesleylloyd641](https://PollEv.com/wesleylloyd641)  
Text WESLEYLLOYD641 to 22333 once to join

**What is the Average Turnaround Time of the Shortest Job First Scheduler?**

Start the presentation to see live content. Still no live content? Install the app or get help at [PollEv.com/app](https://PollEv.com/app)

