

CS 370: OPERATING SYSTEMS

[PROCESS SYNCHRONIZATION]

The Critical Section

A segment of code
accessing a shared resource

The segment
worthy of protection
bookended by two sentinels

The entry and exit

The entry gatekeeping
so *only one* may enter

The exit housekeeping
so *someone else* may enter

Shrideep Pallickara
Computer Science
Colorado State University

COMPUTER SCIENCE DEPARTMENT



COLORADO STATE UNIVERSITY

1

Frequently asked questions from the previous class survey

- How different is the actual code for user vs kernel threads?
- Do we decide when to use kernel vs user threads?
- Are threads ever launched without the user ever knowing them?
- Is there a process-level distinction for the main thread vs. other threads?
- When a join is called by a thread, does the number of program counters reduce by one?
- Does every class that needs be a Java Thread need to implement Runnable?
 - Why is the first statement that is executed in a Java thread in the run() method?
- Can only the main-thread join()?
- Say, thread A performs a join() on a thread B
 - Is thread A now running?
 - Is thread B now running?
- What is a thread pool?



COLORADO STATE UNIVERSITY

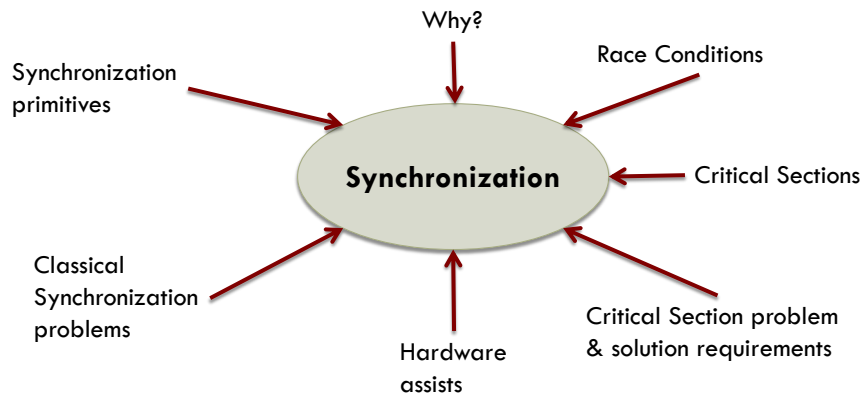
Professor: SHRIDEEP PALICKARA
COMPUTER SCIENCE DEPARTMENT

INTER-PROCESS SYNCHRONIZATION

L9.2

2

Synchronization: What we will look at



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALICKARA
COMPUTER SCIENCE DEPARTMENT

INTER-PROCESS SYNCHRONIZATION

L9.3

3

Topics covered in the lecture

- ☐ Critical section
- ☐ Critical section problem
- ☐ Peterson's solution
- ☐ Hardware assists



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALICKARA
COMPUTER SCIENCE DEPARTMENT

INTER-PROCESS SYNCHRONIZATION

L9.4

4

Reasoning about interleaved access to shared state: Too much milk!

	Roommate 1's actions	Roommate 2's actions
3:00	Look in fridge; out of milk	
3:05	Leave for store	
3:10	Arrive at store	Look in fridge; out of milk
3:15	Buy milk	Leave for store
3:20	Arrive home; put milk away	Arrive at store
3:25		Buy milk
3:30		Arrive home; put milk away
		Oh no!



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALICKARA
COMPUTER SCIENCE DEPARTMENT

INTER-PROCESS SYNCHRONIZATION

L9.5

5

Fairy tales are more than true: not because they tell us that dragons exist, but because they tell us that dragons can be beaten.

G.K. Chesterton by way of Neil Gaiman, Coraline



PROCESS SYNCHRONIZATION

6

Process synchronization

- How can processes **pass information** to one another?
- Make sure two or more processes **do not get in each other's way**
 - ▣ E.g., 2 processes in an airline reservation system, each trying to grab the last seat for a different passenger
- Ensure proper **sequencing** when dependencies are present



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALICKARA
COMPUTER SCIENCE DEPARTMENT

INTER-PROCESS SYNCHRONIZATION

L9.7

7

Applicability to threads

- Passing information between threads is easy
 - ▣ They share the same address space of the parent process
- Other two aspects of process synchronization are applicable to threads
 - ▣ Keeping out of each other's hair
 - ▣ Proper sequencing



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALICKARA
COMPUTER SCIENCE DEPARTMENT

INTER-PROCESS SYNCHRONIZATION

L9.8

8

A look at the producer consumer problem

```
while (true) {  
    while (counter == BUFFER_SIZE) {  
        ; /*do nothing */  
    }  
    buffer[in] = nextProduced  
    in = (in + 1) % BUFFER_SIZE;  
    counter++;  
}
```

Producer



Operators?

```
while (true) {  
    while (counter == 0) {  
        ; /*do nothing */  
    }  
    nextConsumed = buffer[out]  
    out = (out + 1) % BUFFER_SIZE;  
    counter--;  
}
```

Consumer



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALICKARA
COMPUTER SCIENCE DEPARTMENT

INTER-PROCESS SYNCHRONIZATION

L9.9

9

Implementation of ++/-- in machine language

```
counter++  
register1 = counter  
register1 = register1 + 1  
counter  = register1
```

```
counter--  
register2 = counter  
register2 = register2 - 1  
counter  = register2
```



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALICKARA
COMPUTER SCIENCE DEPARTMENT

INTER-PROCESS SYNCHRONIZATION

L9.10

10

Lower-level statements may be interleaved in any order

Producer execute: register1 = counter

Producer execute: register1 = register1 + 1

Producer execute: counter = register1



Correctness?

Consumer execute: register2 = counter

Consumer execute: register2 = register2 - 1

Consumer execute: counter = register2



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALICKARA
COMPUTER SCIENCE DEPARTMENT

INTER-PROCESS SYNCHRONIZATION

L9.11

11

Lower-level statements may be interleaved in any order

Producer execute: register1 = counter

Consumer execute: register2 = counter

Producer execute: register1 = register1 + 1

Consumer execute: register2 = register2 - 1

Producer execute: counter = register1

Consumer execute: counter = register2

The **order** of statements *within* each high-level statement is **preserved**



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALICKARA
COMPUTER SCIENCE DEPARTMENT

INTER-PROCESS SYNCHRONIZATION

L9.12

12

Lower-level statements may be interleaved in any order (counter = 5)

Producer execute: register1 = counter	{register1 = 5}
Producer execute: register1 = register1 + 1	{register1 = 6}
Consumer execute: register2 = counter	{register2 = 5}
Consumer execute: register2 = register2 - 1	{register2 = 4}
Producer execute: counter = register1	{counter = 6}
Consumer execute: counter = register2	{counter = 4}

Counter has **incorrect** state of 4



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALICKARA
COMPUTER SCIENCE DEPARTMENT

INTER-PROCESS SYNCHRONIZATION

L9.13

13

Lower-level statements may be interleaved in any order (counter = 5)

Producer execute: register1 = counter	{register1 = 5}
Producer execute: register1 = register1 + 1	{register1 = 6}
Consumer execute: register2 = counter	{register2 = 5}
Consumer execute: register2 = register2 - 1	{register2 = 4}
Consumer execute: counter = register2	{counter = 4}
Producer execute: counter = register1	{counter = 6}

Counter has **incorrect** state of 6



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALICKARA
COMPUTER SCIENCE DEPARTMENT

INTER-PROCESS SYNCHRONIZATION

L9.14

14



15

Race condition

- Several processes access and manipulate data **concurrently**
- **Outcome** of execution *depends* on
 - ▣ Particular **order** in which accesses takes place
- Debugging programs with race conditions?
 - ▣ Painful!
 - ▣ Program runs fine most of the time, but once in a rare while something weird and unexpected happens



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALICKARA
COMPUTER SCIENCE DEPARTMENT

INTER-PROCESS SYNCHRONIZATION

L9.16

16

Race condition: Example

[1/3]

- When process wants to print file, adds file to a special **spooler directory**
- Printer daemon periodically checks to see if there are files to be printed
 - ▣ If there are, print them
- In our example, spooler directory has a large number of slots
- Two variables
 - ▣ `in`: Next free slot in directory
 - ▣ `out`: Next file to be printed



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALICKARA
COMPUTER SCIENCE DEPARTMENT

INTER-PROCESS SYNCHRONIZATION

L9.17

17

Race condition: Example

[2/3]

- In jurisdictions where Murphy's Law hold ...
- Process A reads `in`, and stores the value 7, in local variable `next_free_slot`
- Context switch occurs
- Process B also reads `in`, and stores the value 7, in local variable `next_free_slot`
 - ▣ Stores name of the file in slot 7
- Process A context switches again, and stores the name of the file it wants to print in slot 7



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALICKARA
COMPUTER SCIENCE DEPARTMENT

INTER-PROCESS SYNCHRONIZATION

L9.18

18

Race condition: Example

[3/3]

- Spooler directory is internally consistent
- But process B will never receive any output
 - ▣ User B loiters around printer room for years, wistfully hoping for an output that will never come ...



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALICKARA
COMPUTER SCIENCE DEPARTMENT

INTER-PROCESS SYNCHRONIZATION

L9.19

19

The kernel is subject to several possible race conditions

- E.g.: Kernel maintains list of all open files
 - ▣ 2 processes open files simultaneously
 - ▣ Separate updates to kernel list may result in a race condition
- Other kernel data structures
 - ▣ Memory allocation
 - ▣ Process lists
 - ▣ Interrupt handling



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALICKARA
COMPUTER SCIENCE DEPARTMENT

INTER-PROCESS SYNCHRONIZATION

L9.20

20



21

Critical Section

- **Concurrent accesses** to **shared resources** can lead to unexpected or erroneous behavior
- Parts of the program where the shared resource is accessed thus need to be protected
 - This protected section is the **critical section**



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALICKARA
COMPUTER SCIENCE DEPARTMENT

INTER-PROCESS SYNCHRONIZATION

L9.22

22

Critical-Section

- System of n processes $\{P_0, P_1, \dots, P_{n-1}\}$
- Each process has a segment of code (**critical section**) where it:
 - **Changes common variables**, updates a table, etc
- No two processes can execute in **their** critical sections at the same time



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALICKARA
COMPUTER SCIENCE DEPARTMENT

INTER-PROCESS SYNCHRONIZATION

L9.23

23

The Critical-Section problem

- Design a **protocol** that processes can use to cooperate
- Each process must **request permission** to enter its critical section
 - The **entry** section



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALICKARA
COMPUTER SCIENCE DEPARTMENT

INTER-PROCESS SYNCHRONIZATION

L9.24

24

General structure of a participating process

```
do {
```

entry section

Request permission
to enter

critical section

exit section

Housekeeping to let
other processes enter

remainder section

```
} while (TRUE);
```



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALICKARA
COMPUTER SCIENCE DEPARTMENT

INTER-PROCESS SYNCHRONIZATION

L9.25

25



**REQUIREMENTS FOR A SOLUTION TO THE
CRITICAL SECTION PROBLEM**

26

Requirements for a solution to the critical section problem

- ① Mutual exclusion
- ② Progress
- ③ Bounded wait

□ PROCESS SPEED

- ▣ Each process operates at **non-zero** speed
- ▣ Make no assumption about the **relative speed** of the n processes



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALICKARA
COMPUTER SCIENCE DEPARTMENT

INTER-PROCESS SYNCHRONIZATION

L9.27

27

Mutual Exclusion

- Only **one** process can execute in its critical section
- When a process executes in its critical section
 - ▣ **No other process** is allowed to execute in *its* critical section



COLORADO STATE UNIVERSITY

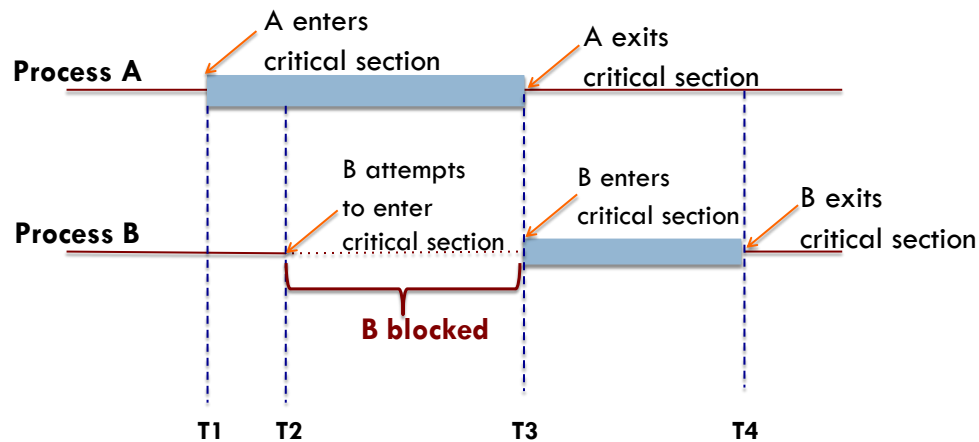
Professor: SHRIDEEP PALICKARA
COMPUTER SCIENCE DEPARTMENT

INTER-PROCESS SYNCHRONIZATION

L9.28

28

Mutual Exclusion: Depiction



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALICKARA
COMPUTER SCIENCE DEPARTMENT

INTER-PROCESS SYNCHRONIZATION

L9.29

29

Progress

- {C1} If **No** process is executing in its critical section, and ...
- {C2} **Some** processes wish to enter their critical sections
- **Decision** on who gets to enter the critical section
 - ▣ Is made by processes that are NOT executing in their remainder section
 - ▣ Selection **cannot be postponed indefinitely**



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALICKARA
COMPUTER SCIENCE DEPARTMENT

INTER-PROCESS SYNCHRONIZATION

L9.30

30

Bounded waiting

- **After** a process has made a **request** to enter its critical section
 - AND **before** this request is granted
- **Limit number** of times other processes are allowed to enter their critical sections



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALICKARA
COMPUTER SCIENCE DEPARTMENT

INTER-PROCESS SYNCHRONIZATION

L9.31

31

Approaches to handling critical sections in the OS

- Nonpreemptive kernel
 - If a process runs in kernel mode: no preemption
 - **Free** from race conditions on kernel data structures
- Preemptive kernels
 - Must ensure shared kernel data is free from race conditions
 - Difficult on SMP (Symmetric Multi Processor) architectures
 - 2 processes may run simultaneously on different processors



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALICKARA
COMPUTER SCIENCE DEPARTMENT

INTER-PROCESS SYNCHRONIZATION

L9.32

32

Kernels: Why preempt?

- Suitable for real-time
 - ▣ A real-time process may preempt a kernel process
- More **responsive**
 - ▣ *Less risk* that kernel mode process will run arbitrarily long



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALICKARA
COMPUTER SCIENCE DEPARTMENT

INTER-PROCESS SYNCHRONIZATION

L9.33

33



34

Peterson's Solution

- **Software solution** to the critical section problem
 - ▣ Restricted to two processes
- No guarantees on modern architectures
 - ▣ Machine language instructions such as `load` and `store` implemented differently
- Good algorithmic description
 - ▣ Shows how to address the 3 requirements



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALICKARA
COMPUTER SCIENCE DEPARTMENT

INTER-PROCESS SYNCHRONIZATION

L9.35

35

Peterson's Solution: The components

- Restricted to two processes
 - P_i and P_j where $j = 1-i$
- **Share** two data items
 - `int turn`
 - ▣ Indicates whose **turn** it is to enter the critical section
 - `boolean flag[2]`
 - ▣ Whether process **is ready** to enter the critical section



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALICKARA
COMPUTER SCIENCE DEPARTMENT

INTER-PROCESS SYNCHRONIZATION

L9.36

36

Peterson's solution: Structure of process P_i

```
do {  
    flag[i] = TRUE;  
    turn = j;  
    while (flag[j] && turn==j) {;}  
    critical section  
    flag[i] = FALSE;  
    remainder section  
} while (TRUE);
```



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALICKARA
COMPUTER SCIENCE DEPARTMENT

INTER-PROCESS SYNCHRONIZATION

L9.37

37

Peterson's solution: Mutual exclusion

```
while (flag[j] && turn==j) {;} 
```

- P_i enters critical section only if
 $\text{flag}[j] == \text{false}$ OR $\text{turn} == i$
- If both processes execute in critical section at the same time
 - $\text{flag}[0] == \text{flag}[1] == \text{true}$
 - **But** turn can be 0 or 1, not BOTH
- If P_j entered critical section
 - $\text{flag}[j] == \text{true}$ AND $\text{turn} == j$
 - Will persist as long as P_j is in the critical section



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALICKARA
COMPUTER SCIENCE DEPARTMENT

INTER-PROCESS SYNCHRONIZATION

L9.38

38

Peterson's Solution: Progress and Bounded wait

- P_i can be stuck only if `flag[j] == true AND turn == j`
 - ▣ If P_j is not ready: `flag[j] == false`, and P_i can enter
 - ▣ Once P_j exits: it resets `flag[j]` to false
- If P_j resets `flag[j]` to true
 - ▣ Must set `turn = i`;
- P_i **will enter** critical section (progress) after at most one entry by P_j (bounded wait)



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALICKARA
COMPUTER SCIENCE DEPARTMENT

INTER-PROCESS SYNCHRONIZATION

L9.39

39



SYNCHRONIZATION HARDWARE



COLORADO STATE UNIVERSITY

40



41

Solving the critical section problem using locks

```
do {  
    acquire lock  
    critical section  
    release lock  
    remainder section  
} while (TRUE);
```



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALICKARA
COMPUTER SCIENCE DEPARTMENT

INTER-PROCESS SYNCHRONIZATION

L9.42

42

Possible assists for solving critical section problem [1/2]

- Uniprocessor environment
 - ▣ **Prevent interrupts** from occurring when shared variable is being modified
 - *No unexpected modifications!*
- Multiprocessor environment
 - ▣ Disabling interrupts is *time consuming*
 - Message passed to ALL processors



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALICKARA
COMPUTER SCIENCE DEPARTMENT

INTER-PROCESS SYNCHRONIZATION

L9.43

43

Possible assists for solving critical section problem [2/2]

- Special **atomic** hardware instructions
 - ▣ Swap content of two words
 - ▣ Modify word



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALICKARA
COMPUTER SCIENCE DEPARTMENT

INTER-PROCESS SYNCHRONIZATION

L9.44

44

Swap ()

```
void Swap(boolean *a, boolean *b ) {  
  
    boolean temp = *a;  
    *a = *b;  
    *b = temp;  
}
```



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALICKARA
COMPUTER SCIENCE DEPARTMENT

INTER-PROCESS SYNCHRONIZATION

L9.45

45

Swap: Shared variable LOCK is initialized to false

```
do {
```

```
    key = TRUE;  
    while (key == TRUE) {  
        Swap(&lock, &key)  
    }
```

Cannot enter critical section
UNLESS lock == FALSE

```
    critical section
```

```
    lock = FALSE;
```

```
    remainder section
```

```
} while (TRUE);
```

lock is a SHARED variable
key is a LOCAL variable

If two Swap () are executed
simultaneously, they will be executed
sequentially in some arbitrary order



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALICKARA
COMPUTER SCIENCE DEPARTMENT

INTER-PROCESS SYNCHRONIZATION

L9.46

46

TestAndSet ()

```
boolean TestAndSet(boolean *target ) {  
  
    boolean rv = *target;  
    *target = TRUE;  
    return rv;  
}
```

Sets target to true and returns old value of target



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALICKARA
COMPUTER SCIENCE DEPARTMENT

INTER-PROCESS SYNCHRONIZATION

L9.47

47

TestAndSet: Shared boolean variable lock initialized to false

```
do {  
  
    while (TestAndSet(&lock)) {;}  
  
    critical section  
  
    lock = FALSE;  
  
    remainder section  
  
} while (TRUE);
```

To break out:
Return value of TestAndSet
should be FALSE

If two TestAndSet () are executed
simultaneously, they will be executed
sequentially in some arbitrary order



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALICKARA
COMPUTER SCIENCE DEPARTMENT

INTER-PROCESS SYNCHRONIZATION

L9.48

48

Entering and leaving critical regions using TestAndSet and Swap (Exchange)

```
enter_region:
    TSL REGISTER, LOCK
    CMP REGISTER, #0
    JNE enter_region
    RET
```

```
leave_region:
    MOVE LOCK, #0
    RET
```

```
enter_region:
    MOVE REGISTER, #1
    XCHNG REGISTER, LOCK
    CMP REGISTER, #0
    JNE enter_region
    RET
```

```
leave_region:
    MOVE LOCK, #0
    RET
```

All Intel x86 CPUs have the XCHG instruction for low-level synchronization



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALICKARA
COMPUTER SCIENCE DEPARTMENT

INTER-PROCESS SYNCHRONIZATION

L9.49

49

The contents of this slide set are based on the following references

- Avi Silberschatz, Peter Galvin, Greg Gagne. *Operating Systems Concepts*, 9th edition. John Wiley & Sons, Inc. ISBN-13: 978-1118063330. [Chapter 5]
- Andrew S Tanenbaum and Herbert Bos. *Modern Operating Systems*. 4th Edition, 2014. Prentice Hall. ISBN: 013359162X/ 978-0133591620. [Chapter 2]
- Thomas Anderson and Michael Dahlin. *Operating Systems Principles and Practice*. 2nd Edition. ISBN: 978-0985673529. [Chapter 5]
- https://en.wikipedia.org/wiki/Critical_section



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALICKARA
COMPUTER SCIENCE DEPARTMENT

INTER-PROCESS SYNCHRONIZATION

L9.50

50