

CS 370: OPERATING SYSTEMS

[PROCESS SYNCHRONIZATION]

Monitors and their Shiny Armor

Semaphores getting on your nerve?

A missed wait here

A misplaced signal there

Deadlocks, inconsistency we observe

Monitors to the rescue

Encapsulating synchronization

For smooth, correct operation

With nary a miscue

Shrideep Pallickara
Computer Science
Colorado State University

COMPUTER SCIENCE DEPARTMENT



COLORADO STATE UNIVERSITY

1

Frequently asked questions from the previous class survey

- Critical section
 - Should there always be a process in a critical section?
 - Can a process be forced to leave its critical section?
 - How "long" (time) should a critical section be?
- Priority inversion: Why can't the higher priority process preempt?
- Do the discussions about synchronization across processes apply to threads as well?
- Spinlocks
 - What if the code it protects is very computationally expensive?
- Producer consumers
 - Equal numbers of each? Allowed to switch roles?
- Mutex vs semaphore
 - Mutex: provides mutual exclusion; binary state; only one process in the critical section
 - Semaphores: Used for signaling; can be initialized to > 1 ; allows multiple threads to access simultaneously
- Implications for synchronization in programming languages?



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALICKARA
COMPUTER SCIENCE DEPARTMENT

INTER-PROCESS SYNCHRONIZATION

L11.2

2

Topics covered in the lecture

- Classical process synchronization problems
 - ▣ Readers Writers
 - ▣ Dining philosopher's problem
- Monitors
 - ▣ Solving dining philosopher's problem using monitors
- Midterm



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALICKARA
COMPUTER SCIENCE DEPARTMENT

INTER-PROCESS SYNCHRONIZATION

L11.3

3

"Classic." A book which people praise and don't read.

Mark Twain

A photograph of a stack of several books, with a pen resting on top. The books are slightly out of focus, and the pen is in the foreground. The background is dark.

THE READERS-WRITERS PROBLEM

L10.4

4

The Readers-Writers problem

- A database is **shared** among several concurrent processes
- Two types of processes
 - Readers
 - Writers



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALICKARA
COMPUTER SCIENCE DEPARTMENT

INTER-PROCESS SYNCHRONIZATION

L11.5

5

Readers-Writers: Potential for adverse effects

- If *two readers* access shared data simultaneously?
 - No problems
- If a *writer and some other reader* (or writer) access shared data simultaneously?
 - Chaos



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALICKARA
COMPUTER SCIENCE DEPARTMENT

INTER-PROCESS SYNCHRONIZATION

L11.6

6

Writers must have exclusive access to shared database while writing

- FIRST readers-writers problem:
 - ▣ No reader should wait for other readers to finish; simply because a writer is waiting
 - Writers may starve

- SECOND readers-writers problem:
 - ▣ If a writer is ready, it performs its write ASAP
 - Readers may starve



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALICKARA
COMPUTER SCIENCE DEPARTMENT

INTER-PROCESS SYNCHRONIZATION

L11.7

7

Solution to the FIRST readers-writers problem

- Variable `int readcount`
 - ▣ Tracks how many readers are reading the database

- Semaphore `mutex {1}`
 - ▣ Ensure mutual exclusion when `readcount` is accessed

- Semaphore `wrt {1}`
 - ① Mutual exclusion for the writers
 - ② First (**last**) reader that enters (**exits**) critical section
 - Not used by readers, when **other** readers **are in** their critical section



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALICKARA
COMPUTER SCIENCE DEPARTMENT

INTER-PROCESS SYNCHRONIZATION

L11.8

8

The Writer: When a writer “signals” either a waiting writer or the readers resume

```
do {
```

```
    wait(wrt);
```

```
    writing is performed
```

```
    signal(wrt);
```

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

When:

writer in critical section
and if n readers waiting

1 reader is queued on **wrt**
(n-1) readers queued on **mutex**



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALICKARA
COMPUTER SCIENCE DEPARTMENT

INTER-PROCESS SYNCHRONIZATION

L11.9

9

The Reader process

```
do {
```

```
    wait(mutex);  
    readcount++;  
    if (readcount ==1) {  
        wait(wrt);  
    }  
    signal(mutex);
```

```
    reading is performed
```

```
    wait(mutex);  
    readcount--;  
    if (readcount ==0) {  
        signal(wrt);  
    }  
    signal(mutex);
```

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

mutex for mutual
exclusion to readcount

When:

writer in critical section
and if n readers waiting

1 is queued on **wrt**
(n-1) queued on **mutex**



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALICKARA
COMPUTER SCIENCE DEPARTMENT

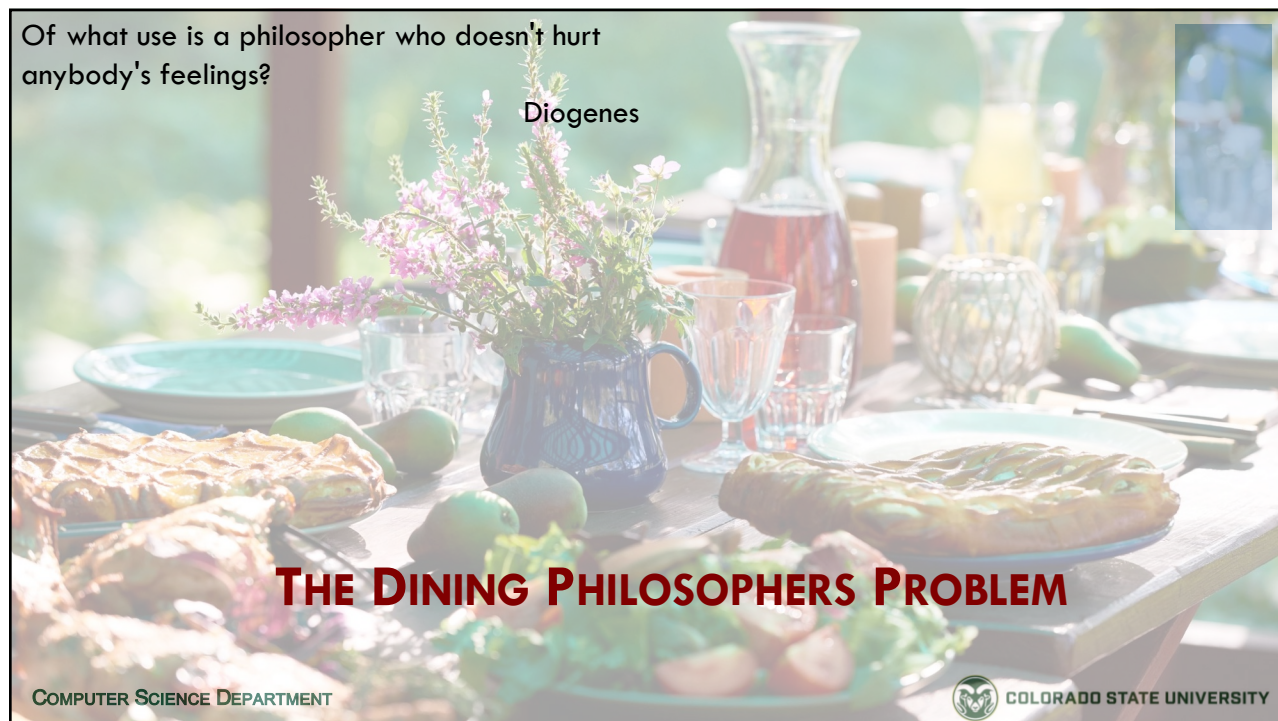
INTER-PROCESS SYNCHRONIZATION

L11.10


10

Of what use is a philosopher who doesn't hurt anybody's feelings?

Diogenes

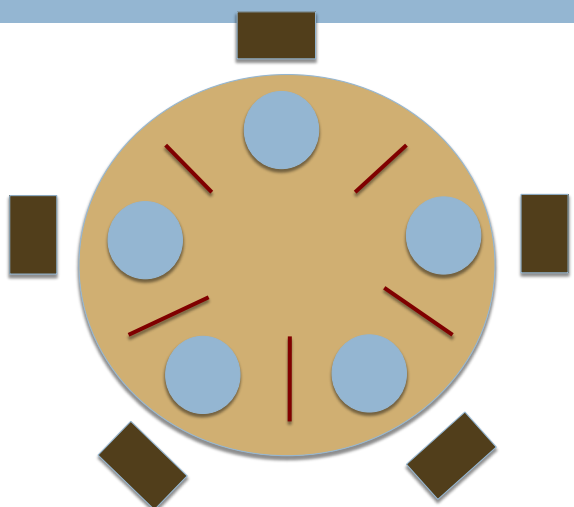



THE DINING PHILOSOPHERS PROBLEM

COMPUTER SCIENCE DEPARTMENT  COLORADO STATE UNIVERSITY

11

The situation



 **COLORADO STATE UNIVERSITY** Professor: SHRIDEEP PALLICKARA
COMPUTER SCIENCE DEPARTMENT INTER-PROCESS SYNCHRONIZATION L11.12

12

The Problem

- ① Philosopher tries to *pick up two closest* {LR} chopsticks
- ② Pick up only **1 chopstick at a time**
 - ▣ Cannot pick up a chopstick being used
- ③ Eat only when you have *both* chopsticks
- ④ When done; *put down both* the chopsticks



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALICKARA
COMPUTER SCIENCE DEPARTMENT

INTER-PROCESS SYNCHRONIZATION

L11.13

13

Why is the problem important?

- ▣ Represents allocation of **several resources**
 - ▣ AMONG **several processes**
- ▣ Can this be done so that it is:
 - ▣ Deadlock free
 - ▣ Starvation free



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALICKARA
COMPUTER SCIENCE DEPARTMENT

INTER-PROCESS SYNCHRONIZATION

L11.14

14

Dining philosophers: Simple solution

[1/2]

- Each chopstick is a semaphore
 - ▣ Grab by executing `wait()`
 - ▣ Release by executing `signal()`
- Shared data
 - ▣ `semaphore chopstick[5];`
 - ▣ All elements are initialized to 1



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALICKARA
COMPUTER SCIENCE DEPARTMENT

INTER-PROCESS SYNCHRONIZATION

L11.15

15

Dining philosophers: Simple solution

[2/2]

```
do {  
    wait(chopstick[i]);  
    wait(chopstick[(i+1)%5]);  
  
    //eat  
  
    signal(chopstick[i]);  
    signal(chopstick[(i+1)%5]);  
  
    //think  
  
} while (TRUE);
```

Deadlock:
If all processes
access chopstick with
same hand

We will look at solution with monitors



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALICKARA
COMPUTER SCIENCE DEPARTMENT

INTER-PROCESS SYNCHRONIZATION

L11.16

16

And still they lead me back
To the long winding road
You left me standing here
A long, long time ago
Don't leave me waiting here
Lead me to your door
The Long and Winding Road, John Lennon/Paul McCartney

MONITORS

COMPUTER SCIENCE DEPARTMENT

COLORADO STATE UNIVERSITY

17

Overview of the semaphore solution

- Processes share a semaphore **mutex**
 - ▣ Initialized to 1
- Each process **MUST** execute
 - ▣ **wait** *before entering* critical section
 - ▣ **signal** *after exiting* critical section

COLORADO STATE UNIVERSITY Professor: SHRIDEEP PALICKARA
COMPUTER SCIENCE DEPARTMENT INTER-PROCESS SYNCHRONIZATION L11.18

18

Incorrect use of semaphores can lead to timing errors

- Hard to detect
 - ▣ Reveal themselves only during specific execution sequences
- If correct sequence is not observed
 - ▣ 2 processes may be in critical section simultaneously
- Problems even if only one process is not well behaved



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALICKARA
COMPUTER SCIENCE DEPARTMENT

INTER-PROCESS SYNCHRONIZATION

L11.19

19

Incorrect use of semaphores: Interchange order of wait and signal

[1/3]

```
do {
```

```
    signal(mutex);
```

```
    critical section
```

```
    wait(mutex);
```

```
    remainder section
```

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

Problem:

Several processes
simultaneously active
in critical section



What if?

NB: *Not always* reproducible



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALICKARA
COMPUTER SCIENCE DEPARTMENT

INTER-PROCESS SYNCHRONIZATION

L11.20

20

Incorrect use of semaphores: [2/3] Replace signal with wait

```
do {  
    wait(mutex);  
    critical section  
  
    wait(mutex);  
    remainder section  
  
} while (TRUE);
```

What if?

Problem: Deadlock!

COLORADO STATE UNIVERSITY Professor: SHRIDEEP PALICKARA COMPUTER SCIENCE DEPARTMENT INTER-PROCESS SYNCHRONIZATION L11.21

21

Incorrect use of semaphores: [3/3] What if you omit signal AND/OR wait?

```
do {  
    wait(mutex);  
    critical section  
  
    signal(mutex);  
    remainder section  
  
} while (TRUE);
```

Omission?

Omission: Mutual exclusion violated

Omission?

Omission: Deadlock!

COLORADO STATE UNIVERSITY Professor: SHRIDEEP PALICKARA COMPUTER SCIENCE DEPARTMENT INTER-PROCESS SYNCHRONIZATION L11.22

22

When programmers use semaphores incorrectly problems arise

- We need a higher-level synchronization construct
 - **Monitor**

- Before we move ahead: Abstract Data Types
 - Encapsulates *private data* with
 - *Public methods* to operate on them



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALICKARA
COMPUTER SCIENCE DEPARTMENT

INTER-PROCESS SYNCHRONIZATION

L11.23

23

A monitor is an abstract data type

- Mutual exclusion provided **within** the monitor

- Contains:
 - Declaration of variables
 - Defining the instance's state

 - Functions that operate on these variables



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALICKARA
COMPUTER SCIENCE DEPARTMENT

INTER-PROCESS SYNCHRONIZATION

L11.24

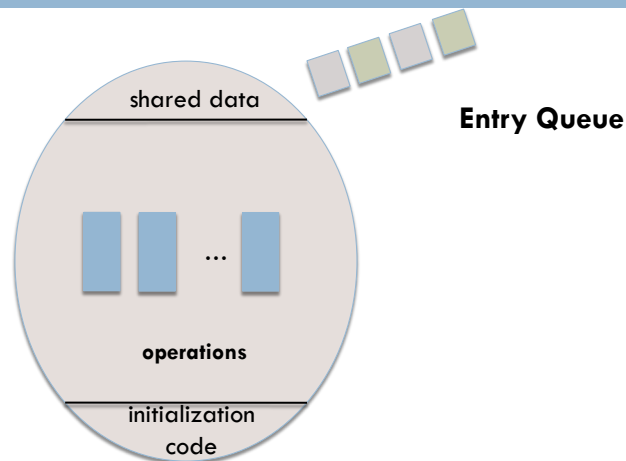
24

Monitor construct ensures that only one process at a time is active within monitor

```
monitor monitor name {  
    //shared variable declarations  
  
    function F1(..) {...}  
  
    function F2(..) {...}  
  
    function Fn(..) {...}  
  
    initialization code(..) {...}  
  
}
```



Programmer does not code synchronization constraint explicitly



Basic monitor scheme not sufficiently powerful

- Provides an easy way to achieve mutual exclusion
- But ... we also need a way for processes to **block** when they cannot proceed



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALLICKARA
COMPUTER SCIENCE DEPARTMENT

INTER-PROCESS SYNCHRONIZATION

L11.27

27

This blocking capability is provided by the condition construct

- The **condition** construct
 - ▣ `condition x, y;`
- Operations on a **condition** variable
 - ▣ `wait: e.g. x.wait()`
 - Process invoking this is suspended UNTIL
 - ▣ `signal: e.g. x.signal()`
 - Resumes exactly-one suspended process
 - If no process waiting; NO EFFECT on state of **x**



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALLICKARA
COMPUTER SCIENCE DEPARTMENT

INTER-PROCESS SYNCHRONIZATION

L11.28

28

Semantics of `wait` and `signal`

- `x.signal()` invoked by process P
- Q is the suspended process waiting on x

- *Signal and wait*: P waits for Q to leave monitor
- *Signal and continue*: Q waits till P leaves monitor

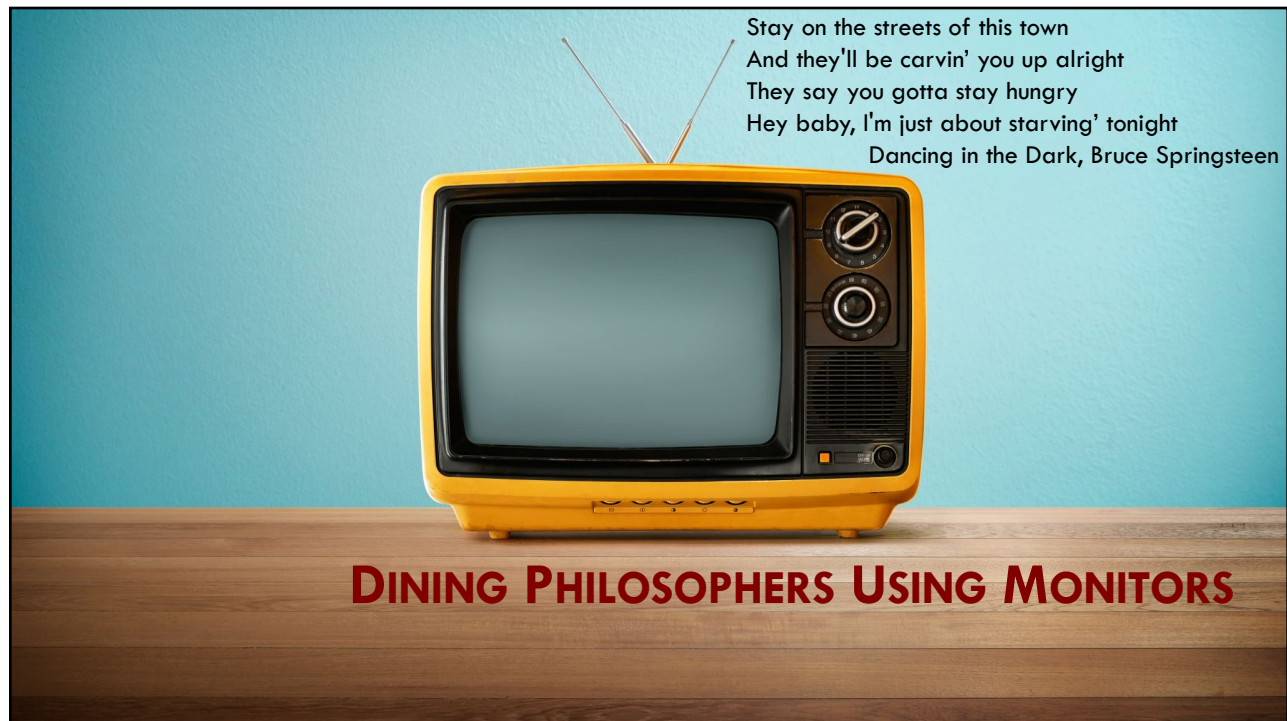
- PASCAL: When thread P calls `signal`
 - P leaves immediately
 - Q immediately resumed



Difference between the `signal()` in semaphores and monitors

- Monitors {condition variables}: **Not persistent**
 - If a signal is performed and no waiting threads?
 - Signal is simply ignored
 - During subsequent `wait` operations
 - Thread blocks
- Semaphores
 - Signal **increments** semaphore value *even if* there are no waiting threads
 - Future `wait` operations would immediately succeed!






31

Dining-Philosophers Using Monitors

Deadlock-free

```
enum {THINKING, HUNGRY, EATING} state[5];
```

- `state[i] = EATING` only if
 - `state[(i+4)%5] != EATING` &&
`state[(i+1)%5] != EATING`
- `condition self[5]`
 - ▣ **Delay** self when *HUNGRY but unable* to get chopsticks

 **COLORADO STATE UNIVERSITY** Professor: SHRIDEEP PALICKARA
COMPUTER SCIENCE DEPARTMENT INTER-PROCESS SYNCHRONIZATION L11.32

32

Sequence of actions

- Before eating, must invoke `pickup()`
 - May result in suspension of the philosopher process
 - After completion of operation, philosopher may eat

```
DiningPhilosophers.pickup(i);  
...  
eat  
...  
DiningPhilosophers.putdown(i);
```



The `pickup()` and `putdown()` operations

```
pickup(int i) {  
    state[i] = HUNGRY;  
    test(i);  
    if (state[i] != EATING) {  
        self[i].wait();  
    }  
}
```

Suspend self if unable
to acquire chopstick

```
putdown(int i) {  
    state[i] = THINKING;  
    test( (i+4)%5 );  
    test( (i+1)%5 );  
}
```

Check to see if person on
left or right can use the
chopstick



test () to see if philosopher can eat

```
test(int i) {  
    if (state[(i+4)%5] != EATING &&  
        state[i] == HUNGRY &&  
        state[(i+1)%5] != EATING) {  
  
        state[i] = EATING;  
        self[i].signal();  
    }  
}
```

Eat only if HUNGRY and
Person on **Left AND Right**
are not eating

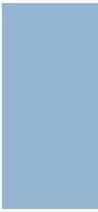
Signal a process that was
suspended while trying to eat



Possibility of starvation


- Philosopher **i** can **starve** if eating periods of philosophers on left and right overlap
- Possible solution
 - Introduce new state: STARVING
 - Chopsticks can be picked up if **no** neighbor is starving
 - Effectively wait for neighbor's neighbor to stop eating
 - REDUCES concurrency!





MID-TERM

COMPUTER SCIENCE DEPARTMENT




COLORADO STATE UNIVERSITY

37

Mid-term on Thursday, March 6th @ 9:30 am

- Held in class
 - Those taking it at the Alternative Testing Center please work with SDC
- Accounts for 20% of your course grade
- Points distribution
 - Processes and Inter-Process Communications: 30 points
 - Threads: 20 points
 - Process Synchronization (including atomic transactions): 30 points



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALICKARA
COMPUTER SCIENCE DEPARTMENT

INTER-PROCESS SYNCHRONIZATION L11.38

38

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]*

