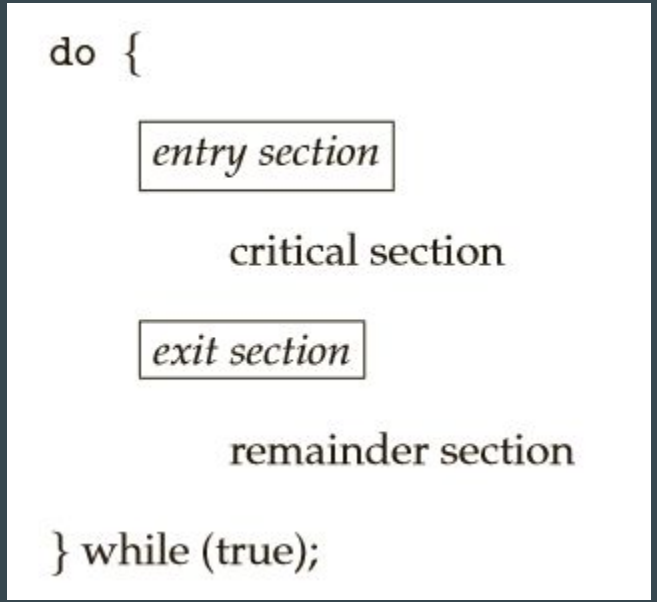
Presentation on Synchronization

By Erik Johansson, Robin Larsson, Sofia Lindström, Marcus Mueller, Adell Tatrous and Helena Tévar Hernández.

The Critical Section Problem



Race Condition

$register_1 = \texttt{counter}$	$\{register_1 = 5\}$
$register_1 = register_1 + 1$	$\{register_1 = 6\}$
$register_2 = \texttt{counter}$	$\{register_2 = 5\}$
$register_2 = register_2 - 1$	$\{register_2 = 4\}$
$\texttt{counter} = register_1$	$\{counter = 6\}$
$\texttt{counter} = register_2$	$\{counter = 4\}$

Rules for solutions:

1. Mutual exclusion
2. Progress
3. Bounded Waiting

Solving Critical Section Problems in OS

- Preemptive kernel
- Non preemptive kernel

Peterson’s Solution

- Software based solution
- Might not work

Hardware Synchronization for:

- Single Processor
- Multi Processor

Mutex Locks

- Hardware solutions are complicated

  ➡ Software tools

- Simplest and easiest to apply
- One process can use its critical section at a time

❌ Busy waiting

❌ Wasting CPU cycles

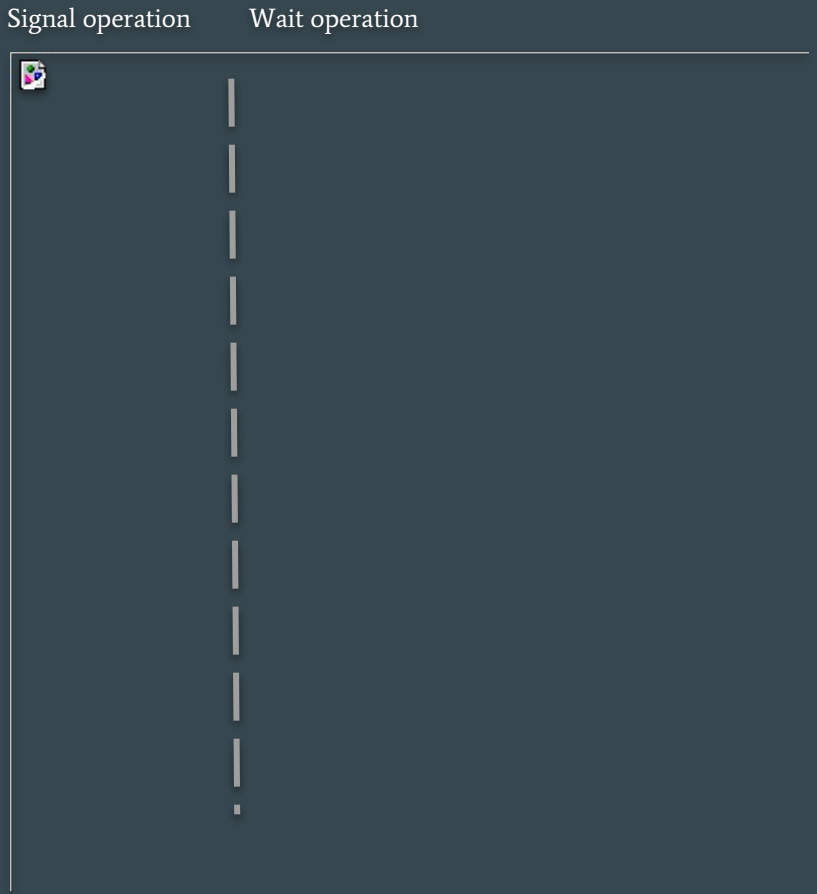
- Spinlocks

Semaphores

- Similar to mutex locks
- Semaphore is an integer

1. Binary semaphores: 0 or 1

2. Counting semaphores: ∞ of integers



Semaphores

- Busy waiting
- *Semaphore, wait() and signal()*
- Implemented atomically → interrupts
- Busy waiting → critical section
- Deadlocks
- Starvation

Monitors

When semaphores don’t work

Monitor

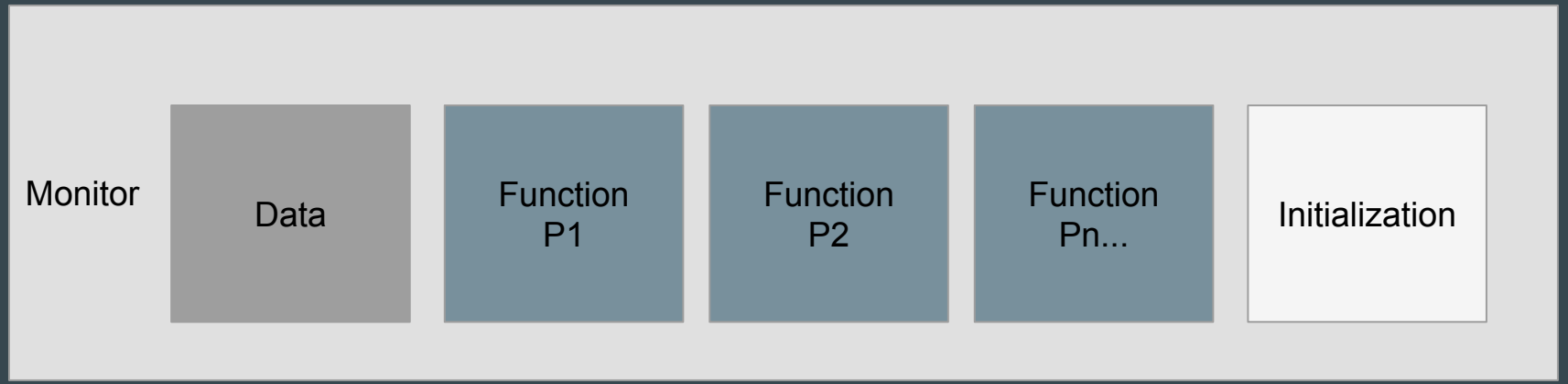
Abstract Data Type

Encapsulates data and functions

Defined operations ⟶ mutex

Declare variables ONLY accessible by its functions

This is not enough, we need more synchronization!



Conditions

Programmer define variables type condition

Their only operations are:

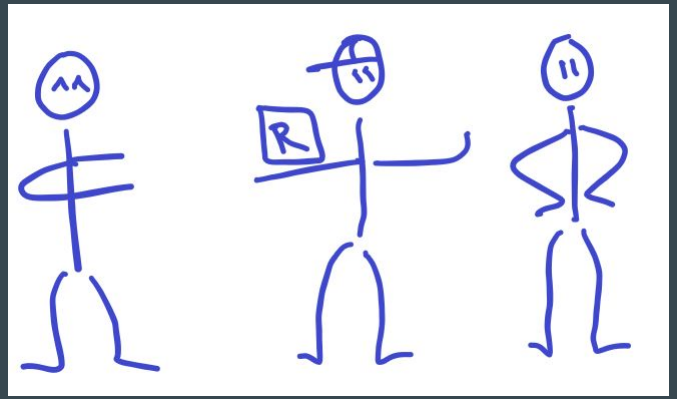
- Wait
- Signal

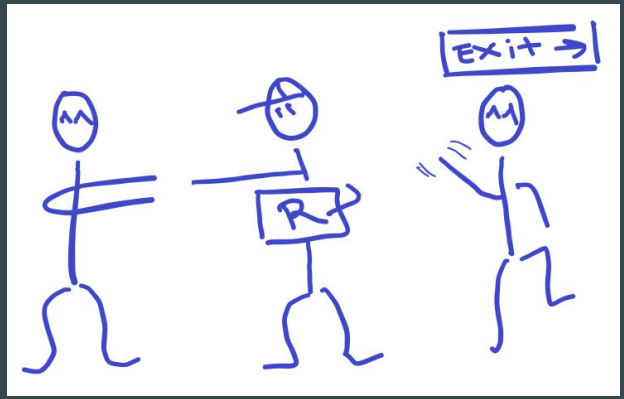
Difference with semaphores: If no process waits, signal does nothing.

Resuming processes:

Signal ⟶ Wait

Signal ⟶ Continue





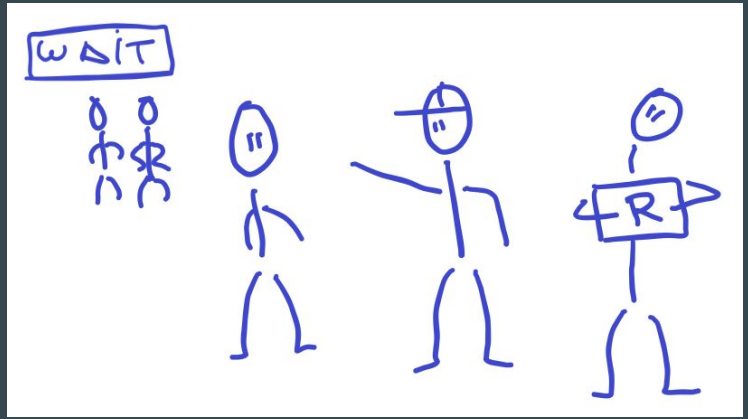
Monitors + Semaphores

Semaphore MUTEX

- Waits before entering Monitor
- Signals after leaving Monitor

Semaphore Next

- Wait to suspend process
- Signal to “wake” process



Resuming processes - Who's next?

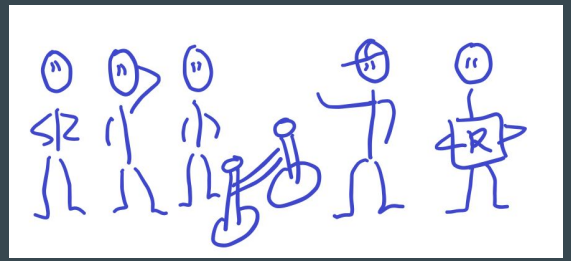
- FIFO Queue
- Conditional Wait

Process call wait with a priority number

Problems

Wrong use of semaphores, operations, compiler cannot help us.

Java synch mechanism in java.langObject



Synchronization in...

Windows

```
if (system == single-processor) {
    “Other interrupt handlers are temporarily masked out.”
}
```

- spinlocks
- dispatcher objects
  - mutex Lock
  - semaphores
  - Events (condition variables)
  - Timers
- critical-section objects
  - User-mode (long wait)-> kernel mutex

Synchronization in...

Linux

- atomic integer
- mutex_lock()
- spinlocks
  - reader–writer version
- semaphores
  - reader–writer version

single processor	multiple processors
Disable kernel preemption.	Acquire spin lock.
Enable kernel preemption.	Release spin lock.

Synchronization in...

Solaris

locking mechanisms are implemented both for kernel and user-level threads

- adaptive mutex locks
- condition variables
- semaphores
- reader–writer locks
- turnstiles

Synchronization in...

Pthreads

available for programmers at the user level and is not part of any particular kernel

- mutex locks
- condition variables
- read–write locks

EXTENSIONS

- semaphores
- spinlocks
- + more

change... changes things...

Transactional memory

can be added to a programming language

system and not the developer is responsible for atomicity

OpenMP

a set of compiler directives and an API

system and not the developer is responsible for atomicity

critical-section compiler

Functional programming

do not maintain state

do not have most of the problems talked about

clojure.org/about/concurrent_programming

Thank you!

“All materials and illustrations used in this presentation are taken from the book “Operating System Concepts” by Abraham Silberschatz, 2012”

- Group H