We will wait 10 minutes until 10:40 AM for all students to join into the meeting.

We will start the tutorial at **10:40 AM**.

CS 3SD3 - Concurrent Systems Tutorial 5

Mahdee Jodayree

October 19, 2021

Outline

- Announcements / Reminders
- Semaphore
- Binary Semaphore
- Semaphore notes and codes lecture notes.

Announcements

- Assignment 1 is marked.
- Assignment 2 is released, and it is due on Nov 8th, 2021.
- Mid-term is on October 28th.
- Information about the midterm is posted on the course website.
- Common mistakes in assignment 1.

Assignment 1 – common mistakes.

- Students did not use commas and dots at all.
- For any coding questions you **Must** upload your code and not screenshot of your codes.
- Please make sure that your code is runnable.

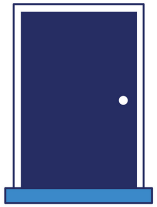
What is a semaphore

Let's understand the concept of semaphores before we study any definitions.

Lets represent

- Every threads or process by 

Washroom

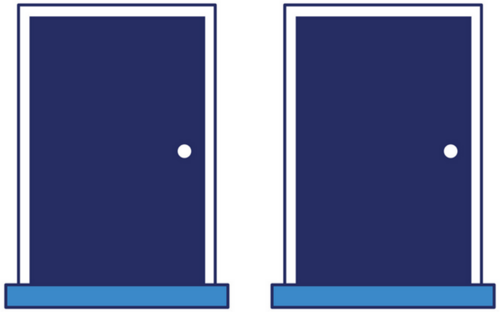
- Critical area by washroom 

- A semaphore by a 

What is a semaphore

- Imagine we have hundreds of people (threads/processes) in an event but we only have two washrooms available and people are lining up to use the washroom (critical area), however we have two washrooms available. One way to solve this problem is to create two keys (semaphores.)
- The key is assigned in a mutual exclusive way (people will never fight over a key or two people cannot share a key.
- Keys represent semaphores.

Two washrooms (Critical section)



Two identical keys for both doors

What is a semaphore

A semaphore is a **variable** (positive integer) or **abstract data** type used to control access to a common resource by multiple processes in a concurrent system such as a multitasking operating system.

A semaphore ensures that a critical section can be accessed by processes in mutual exclusive way (the threads take turn accessing the critical section) and this allows us to achieve process synchronization in the multi processing environment.

Abstract data (in Java, this means object of class). You do not need to understand this for this class.

For example if set our semaphore value into two then only two threads can access the critical section.

Semaphore operations.

The only semaphore operations are

- Down(s) *also known as* wait(s) *also known as* V(s)
- Up(s) *also known as* signal(s) *also know as* P(s)

Semaphores (from lecture notes)

down(s): **if** s > 0 **then**
decrement s
else **//mean s is zero**
blocks execution of the calling process

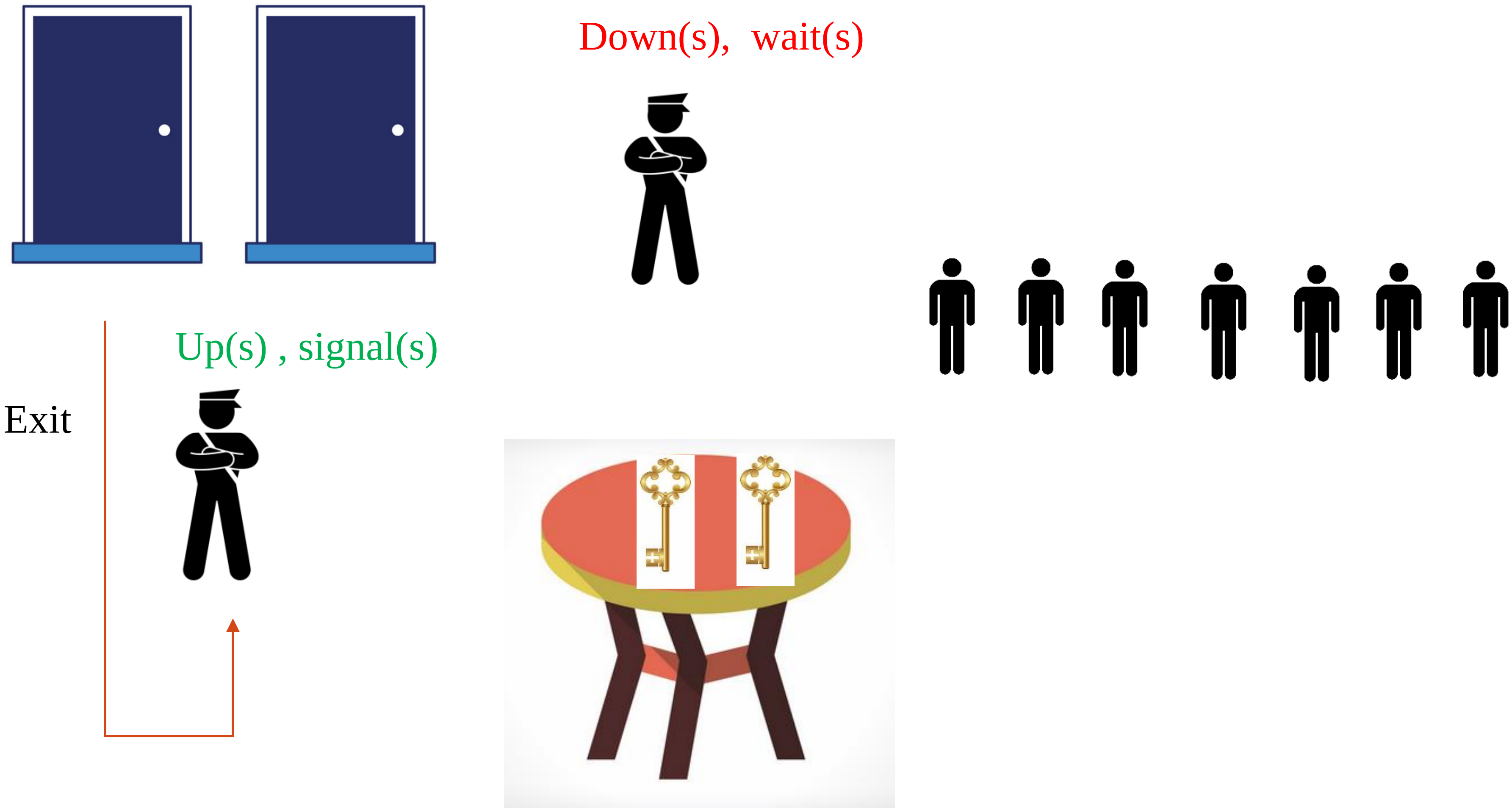
up(s): **if** processes blocked on s **then** **//means s was 0**
awaken one of them
else
increment s

- Binary Semaphores

down(s): **if** s-1 **then** s-0
else blocked execution of the calling process.

up(s): **if** process blocked on s **then** awaken one of them
else s=1

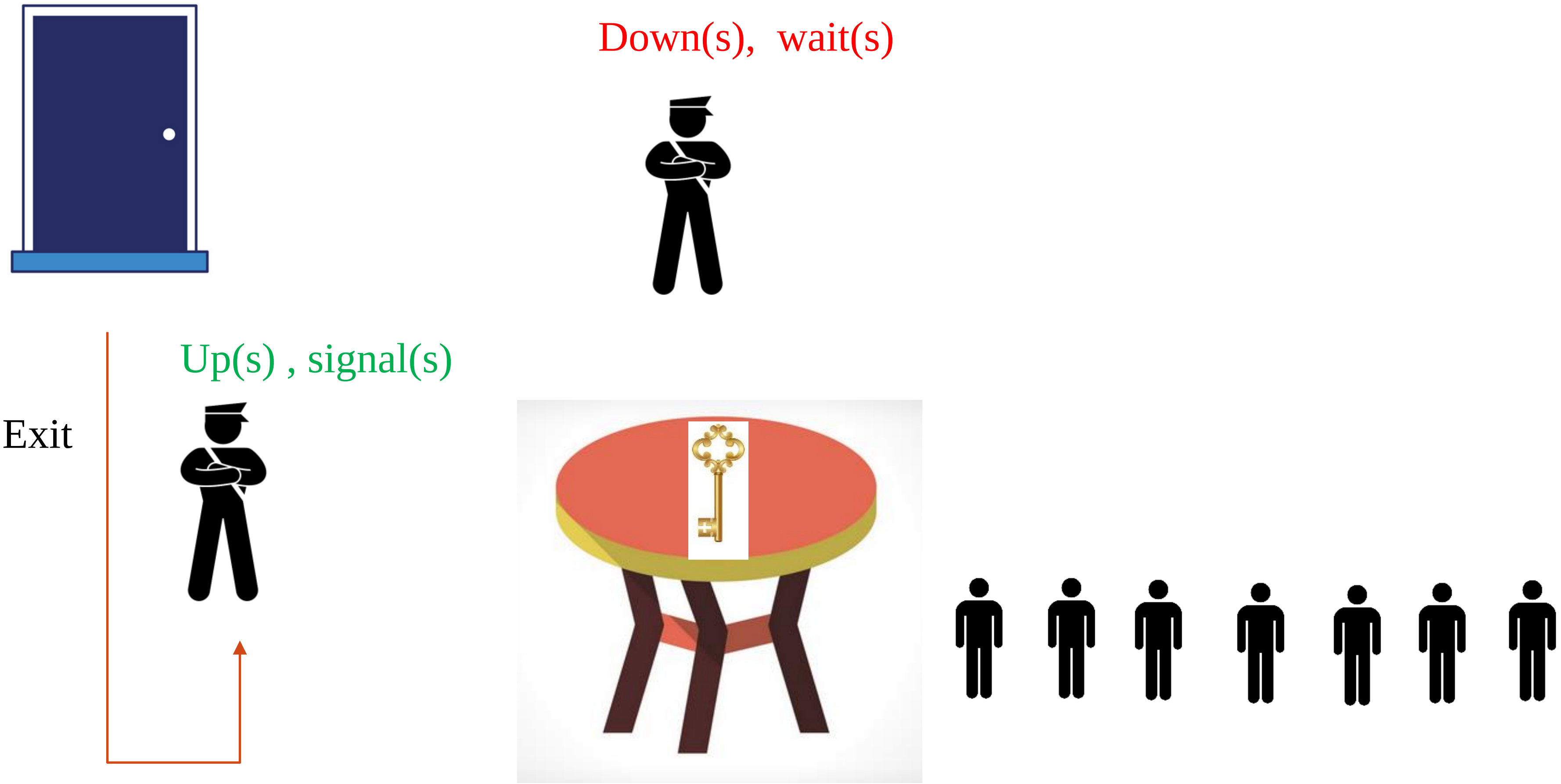
Semaphores



Binary Semaphore

- **Binary Semaphore –** This is also known as mutex lock. It can have only two values – 0 and 1. Its value is initialized to 1. It is used to implement the solution of critical section problem with multiple processes.
- **Semaphore –** Its value can range over an unrestricted domain. It is used to control access to a resource that has multiple instances.
- In other words Binary Semaphore functions as a Mutex but it gives the benefit of using operations of Semaphore.
- So if you need to use Mutex, you can use binary Semaphore and this would allow you to use the Semaphore options.

Binary Semaphore



Mutex and Binary Semaphores

From Lecture 7

Three processes `p[1..3]` use a shared semaphore **`mutex`** to ensure mutually exclusive access (action `critical`) to some resource.

```
LOOP = (mutex.down->critical->mutex.up->LOOP).

||SEMADEMO = (p[1..3]:LOOP
              ||{p[1..3]}::mutex:SEMAPHORE(1)).
```

- For mutual exclusion, the semaphore initial value is 1.
- Binary semaphores are sufficient in this case, actually mutual exclusion provided main motivation to Edsgar Dijkstra for invention of semaphores in 1958 (actually he implemented what was first used for rail tracks in 19 century).

From lecture notes

In the model presented in the textbook, blocked processes are held in FIFO queue. In standard theoretical model they are held in a set (and choice releasing is non-deterministic). In general any protocol for releasing is allowed.

FIFO queue means: First-in/First-out

```
Public static void main (String[] args) throws InterruptedException {
ExecutorService service =  Excecutors.newFixedThreadedPool (nThreads: 50);
IntStream.of(1000).forEach(i -> service.excute(new Task()));
service.shutdown();
service.awaitTermination( timeout: 1, TimeUnit.MINUTES);
}
Static class Task implements Runnable {
Public void run()   {
    //some processing
  semaphore.acquireUninterruptibly(permits: 2);   // Only 2 thread can acquire at a time
// if there are 3 threads the 3rd one will b e blocked here.
  semaphore.release(permits: 2);
   //we must ensure that the number of threads blocked = # of threads // release are equal to
    //IO call to the slow service
   // rest of processing
}
```

Modeling Semaphores with FSP

- Since the semantics of FSP is via LTS, we can only model semaphores that take a finite range of values.
- If this range is exceeded then we regard this as an **error**.
- This may not be true in other models!
- N is the initial value.

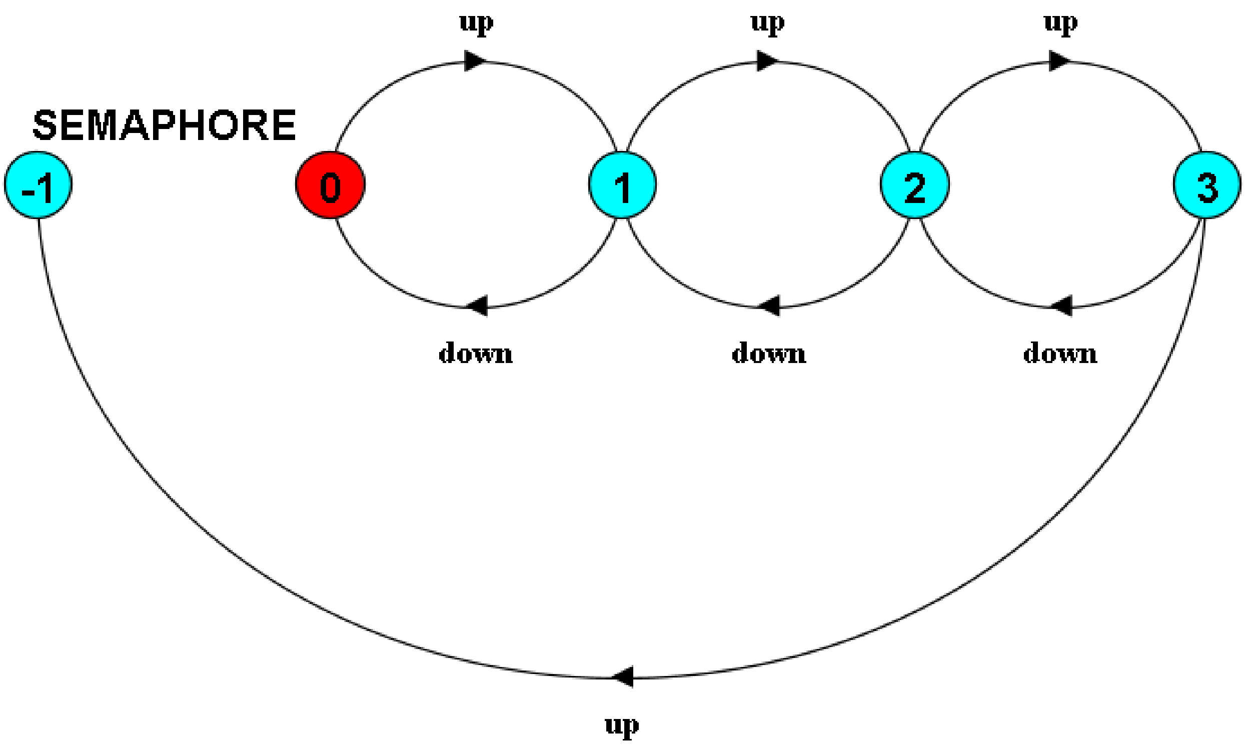
```
const Max = 3
range Int = 0..Max
SEMAPHORE(N=0) = SEMA[N],
SEMA[v:Int]    = (up->SEMA[v+1]
                 |when(v>0) down->SEMA[v-1]),
SEMA[Max+1]    = ERROR.
```

```
const Max = 3
range Int = 0..Max

SEMAPHORE(N=0)  = SEMA[N],
SEMA[v:Int]     = (up->SEMA[v+1]
                  |when(v>0) down->SEMA[v-1]
                  ),
SEMA[Max+1]     = ERROR.
```

```
const Max = 3
range Int = 0..Max
SEMAPHORE(N=0) = SEMA[N],
SEMA[v:Int]     = (up->SEMA[v+1]
                  |when(v>0) down->SEMA[v-1]),
SEMA[Max+1]     = ERROR.
```

```
const Max = 3
range Int = 0..Max
SEMAPHORE(N=0) = SEMA[N],
SEMA[v:Int]    = (up->SEMA[v+1]
                 |when(v>0) down->SEMA[v-1]),
SEMA[Max+1]    = ERROR.
```



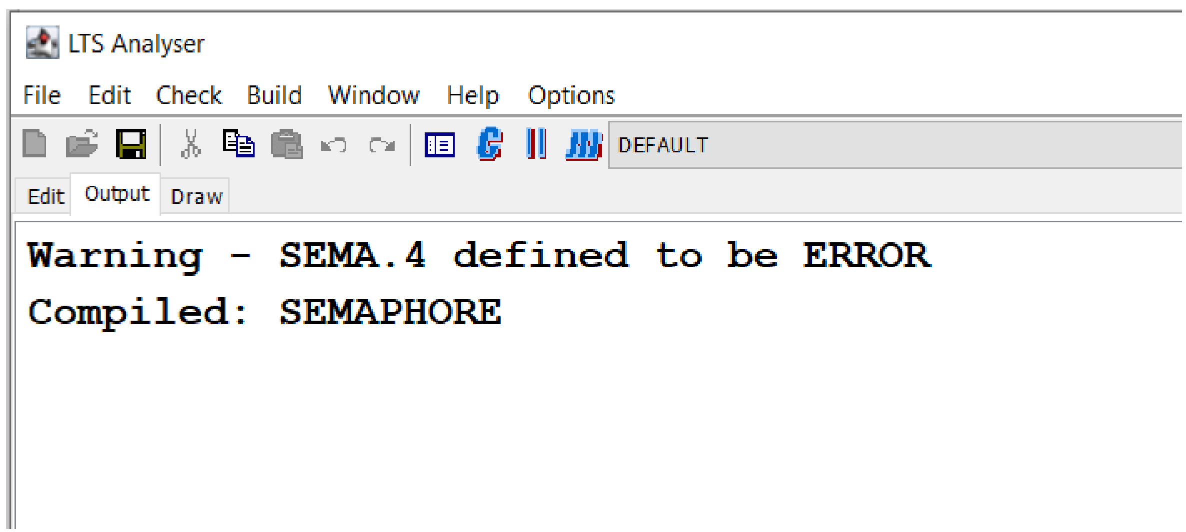
What if

We remove

```
SEMA[Max+1]      = ERROR.
```

If we remove the last line and change the comma if the previous line to a dot.

```
const Max = 3
range Int = 0..Max
SEMAPHORE(N=0)  = SEMA[N],
SEMA[v:Int]     = (up->SEMA[v+1]
                  |when(v>0) down->SEMA[v-1]).
SEMA[Max+1]     = ERROR.
```



- Since the semantics of FSP is via LTS, we can only model semaphores that take a finite range of values.
- If this range is exceeded then we regard this as an **error**.
- This may not be true in other models!
- *N* is the initial value.

```
const Max = 3
range Int = 0..Max

SEMAPHORE(N=0)  = SEMA[N],
SEMA[v:Int]     = (up->SEMA[v+1]
                  |when(v>0) down->SEMA[v-1]
                   ),
SEMA[Max+1]     = ERROR.
```

Modeling Semaphores with FSP

```
const Max = 3
range Int = 0..Max

SEMAPHORE(N=0)  = SEMA[N],
SEMA[v:Int]     = (up->SEMA[v+1]
                  |when(v>0) down->SEMA[v-1]
                  ),
SEMA[Max+1]     = ERROR.
```

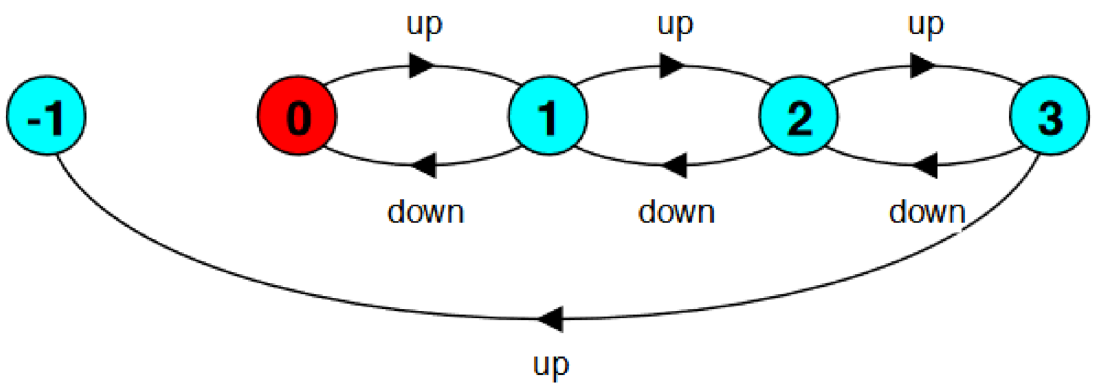
It expands to:

$SEMA[0] = (up \rightarrow SEMA[1])$
$SEMA[1] = (up \rightarrow SEMA[2] \mid down \rightarrow SEMA[0])$
$SEMA[2] = (up \rightarrow SEMA[3] \mid down \rightarrow SEMA[1])$
$SEMA[3] = (up \rightarrow SEMA[4] \mid down \rightarrow SEMA[2])$
$SEMA[4] = ERROR$

- Using *ERROR* is questionable!
- What not: $SEMA[3] = (down \rightarrow SEMA[2])$!!

Modeling Semaphores with LTS

$$
\begin{aligned}
&SEMA[0] = (up \to SEMA[1]) \\
&SEMA[1] = (up \to SEMA[2] \mid down \to SEMA[0]) \\
&SEMA[2] = (up \to SEMA[3] \mid down \to SEMA[1]) \\
&SEMA[3] = (up \to SEMA[4] \mid down \to SEMA[2]) \\
&SEMA[4] = ERROR
\end{aligned}
$$



Action **down** is only accepted when value v of the semaphore is greater than 0.

Action **up** is not guarded.

Trace to a violation:

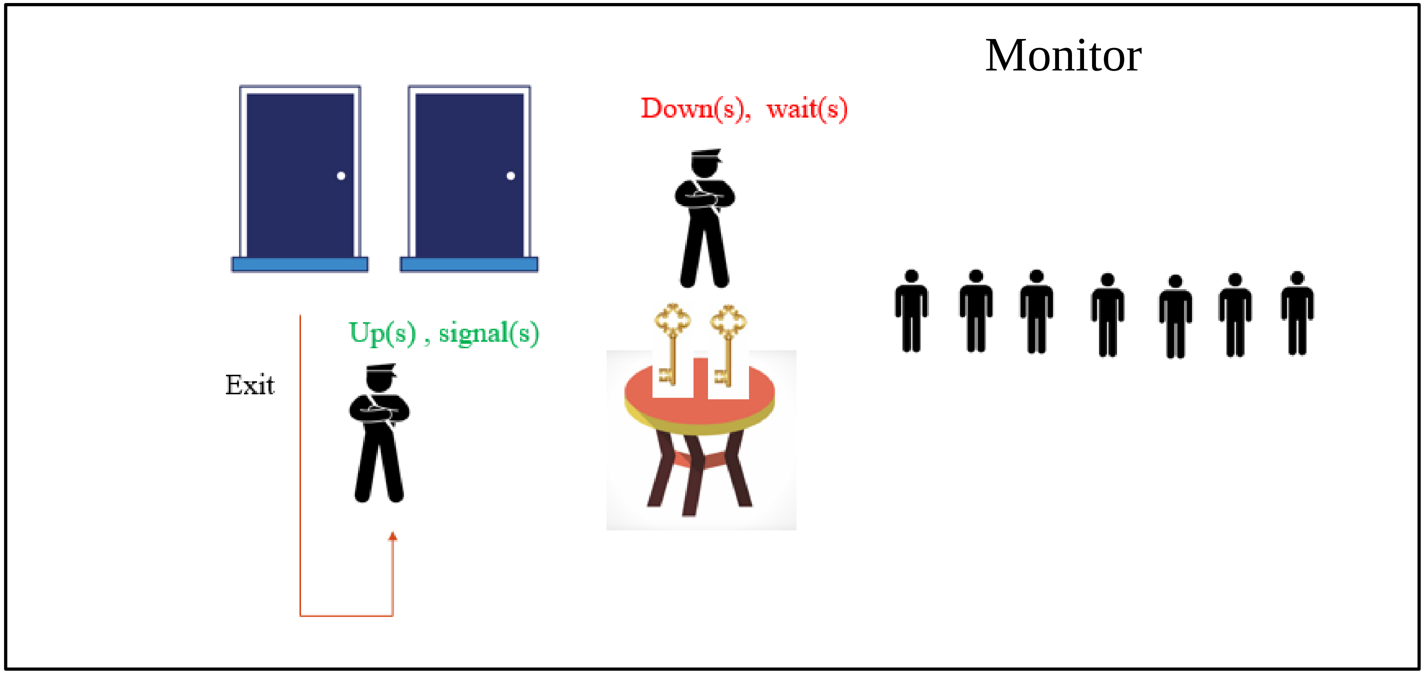
up → up → up → up

More LTSA codes next week.

- Next week more examples of Semaphore codes will be discussed.

We are getting close to understand a monitor

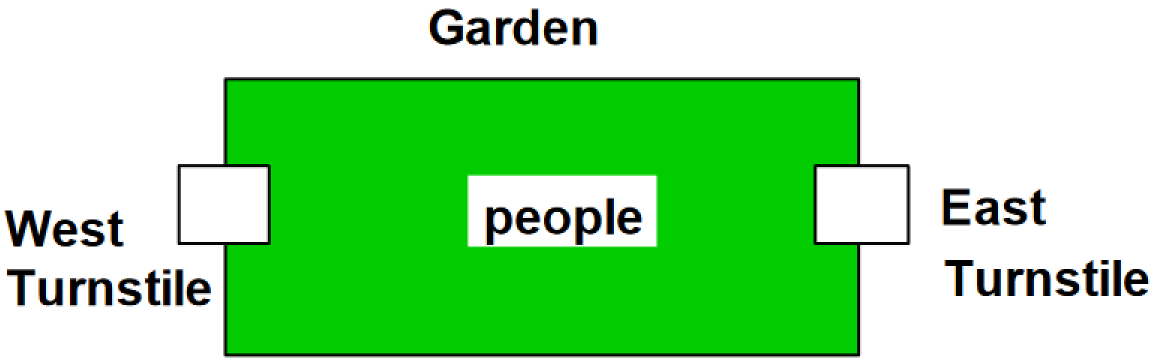
- ❖ Semaphores are part of Monitors but Monitors have more features.



From lecture 6 slides.

Ornamental garden problem:

- People enter an ornamental garden through either of two turnstiles. Management wish to know how many are in the garden at any time.



- Suppose that movement of people is modeled by two concurrent processes and a '*shared*' counter.

Acuire and release lock

- To add locking to our model, define a LOCK, compose it with the shared VAR in the garden, and modify the alphabet set :

```
LOCK = (acquire->release->LOCK).
||LOCKVAR = (LOCK || VAR).

set VarAlpha = {value.{read[T],write[T],
                acquire, release}}
```

- Modify TURNSTILE to acquire and release the lock:

```
TURNSTILE = (go     -> RUN),
RUN       = (arrive-> INCREMENT
            |end    -> TURNSTILE),
INCREMENT = (value.acquire
             -> value.read[x:T]->value.write[x+1]
             -> value.release->RUN
            )+VarAlpha.
```

- Old INCREMENT:
  $INCREMENT = (value.read[x : T] \rightarrow value.write[x + 1] \rightarrow RUN)$
  $+VarAlpha$

- Consider a trace:

$go \rightarrow east.arrive \rightarrow east.value.read[0] \rightarrow west.arrive \rightarrow west.value.read[0] \rightarrow east.value.write[1] \rightarrow west.value.write[1] \rightarrow end \rightarrow display.value.read[1]$

- We have **two** people in the garden but the counter displays number 1!
- $west.value.read[0]$ was executed *before* $east.value.write[1]$, so *VAR* did not update storage!
- The trace below is OK.

$go \rightarrow east.arrive \rightarrow east.value.read[0] \rightarrow east.value.write[1] \rightarrow west.arrive \rightarrow west.value.write[1] \rightarrow west.value.write[2] \rightarrow end \rightarrow display.value.read[2]$

- Trace:

$go \rightarrow east.arrive \rightarrow east.value.acquire \rightarrow east.value.read[0] \rightarrow east.value.write[1] \rightarrow east.value.release \rightarrow west.arrive \rightarrow west.value.acquire \rightarrow west.value.read[1] \rightarrow west.value.read[2] \rightarrow west.value.release \rightarrow end \rightarrow display.value.read[2].$

- We can test it similarly as previously using TEST process and LTSA.
- But tests cannot prove correctness, only can find errors!

Any Questions?