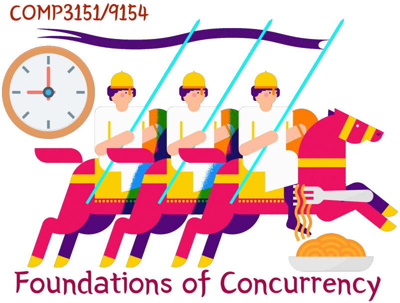


Semaphores

Johannes Åman Pohjola
CSE, UNSW
Term 2 2022

Where we are at

Last week, we saw critical section solutions, and how they are used to implement *locks* (aka *mutexes*).

In this lecture, we will study *semaphores* and the *producer consumer problem*.

Semaphores

First, an abstract view of semaphores:

Definition

A *semaphore* is a pair (v, L) of a natural number v and a set of processes L. A semaphore must always be initialised to some $(v, \emptyset)$.

Semaphores

First, an abstract view of semaphores:

Definition

A *semaphore* is a pair (v, L) of a natural number v and a set of processes L. A semaphore must always be initialised to some $(v, \emptyset)$.

v : how many more processes we can let in without waiting.

L : the processes currently waiting to get in.

Semaphores

First, an abstract view of semaphores:

Definition

A *semaphore* is a pair (v, L) of a natural number v and a set of processes L. A semaphore must always be initialised to some $(v, \emptyset)$.

v : how many more processes we can let in without waiting.

L : the processes currently waiting to get in.

Semaphores

Definition

A process p can do two basic actions on a semaphore S:

wait(S) or $P(S)$, decrements v if positive, otherwise adds p to L and *blocks* p.

Semaphores

Definition

A process p can do two basic actions on a semaphore S:

- **wait**(S) or $P(S)$, decrements v if positive, otherwise adds p to L and *blocks* p.
- **signal**(S) or $V(S)$, if $L \neq \emptyset$, unblocks a member of L. Otherwise increment v.

Semaphores

Definition

A process p can do two basic actions on a semaphore S:

wait(S) or $P(S)$, decrements v if positive, otherwise adds p to L and *blocks* p.

signal(S) or $V(S)$, if $L \neq \emptyset$, unblocks a member of L. Otherwise increment v.

Example (Promela Encoding)

```
inline wait(S) { d_step { S > 0; S-- }}
inline signal(S) { d_step { S ++ } }
```

This is called a busy-wait semaphore. The set L is implicitly the set of (busy-)waiting processes on `S > 0`.

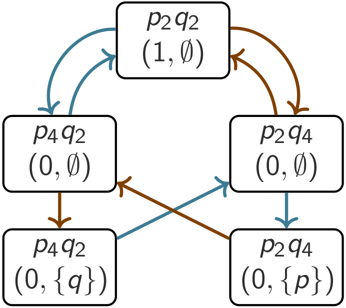
Critical Sections

Locks are just semaphores where the integer starts at 1:

Critical Sections

Locks are just semaphores where the integer starts at 1:

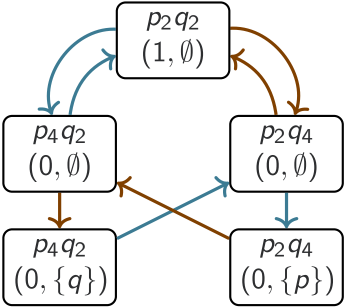
semaphore $S \leftarrow (1, \emptyset)$	
forever do	**forever do**
p_1 *non-critical s.*	q_1 *non-critical s.*
p_2 **wait** (S)	q_2 **wait** (S);
p_3 *critical s.*	q_3 *critical s.*
p_4 **signal** (S)	q_4 **signal** (S);



Critical Sections

Locks are just semaphores where the integer starts at 1:

semaphore $S \leftarrow (1, \emptyset)$	
forever do	**forever do**
p1 *non-critical s.*	q1 *non-critical s.*
p2 **wait** (S)	q2 **wait** (S);
p3 *critical s.*	q3 *critical s.*
p4 **signal** (S)	q4 **signal** (S);



A *weak semaphore* is like our set model earlier. A *busy-wait semaphore* has no set, and implements blocking by spinning in a loop.

Question

What impact does weak vs. busy-wait have on eventual entry?

For N processes

semaphore $S \leftarrow (1, \emptyset)$	
each process i:	
forever do	
i_1	*non-critical section*
i_2	**wait** (S)
i_3	*critical section*
i_4	**signal** (S)

For N processes

semaphore $S \leftarrow (1, \emptyset)$	
each process i:	
forever do	
i_1	*non-critical section*
i_2	**wait** (S)
i_3	*critical section*
i_4	**signal** (S)

Problem 1: With a weak or busy-wait semaphore we don't get eventual entry.
Problem 2: Even with strong fairness, we don't have *linear waiting*.

For N processes

semaphore $S \leftarrow (1, \emptyset)$	
each process i:	
forever do	
i_1	*non-critical section*
i_2	**wait** (S)
i_3	*critical section*
i_4	**signal** (S)

Problem 1: With a weak or busy-wait semaphore we don't get eventual entry.
Problem 2: Even with strong fairness, we don't have *linear waiting*.

Strong Semaphores

Replace the set L with a queue, wake processes up in FIFO order.
This guarantees *linear waiting*, but is harder to implement and potentially more expensive.

Reasoning about Semaphores

For a semaphore $S = (v, L)$ initialised to $(k, \emptyset)$, the following invariants always hold:

Reasoning about Semaphores

For a semaphore $S = (v, L)$ initialised to $(k, \emptyset)$, the following invariants always hold:

1. $v = k + \#\text{signal}(S) - \#\text{wait}(S)$
2. $v \geq 0$

Reasoning about Semaphores

For a semaphore $S = (v, L)$ initialised to $(k, \emptyset)$, the following invariants always hold:

1. $v = k + \#\text{signal}(S) - \#\text{wait}(S)$
2. $v \geq 0$

Definitions

1. $\#\text{signal}(S)$: how many times **signal**(S) has successfully executed.
2. $\#\text{wait}(S)$: how many times **wait**(S) has successfully executed.

Reasoning about Semaphores

For a semaphore $S = (v, L)$ initialised to $(k, \emptyset)$, the following invariants always hold:

1. $v = k + \#\text{signal}(S) - \#\text{wait}(S)$
2. $v \geq 0$

Definitions

1. $\#\text{signal}(S)$: how many times **signal**(S) has successfully executed.
2. $\#\text{wait}(S)$: how many times **wait**(S) has successfully executed.

A *successful* execution happens when the process has proceeded to the next statement. So if a process is blocked on a **wait**(S), then $\#\text{wait}(S)$ will not increase until the process is unblocked.

Reasoning about Semaphores

For a semaphore $S = (v, L)$ initialised to $(k, \emptyset)$, the following invariants always hold:

1. $v = k + \#\text{signal}(S) - \#\text{wait}(S)$
2. $v \geq 0$

Definitions

1. $\#\text{signal}(S)$: how many times **signal**(S) has successfully executed.
2. $\#\text{wait}(S)$: how many times **wait**(S) has successfully executed.

A *successful* execution happens when the process has proceeded to the next statement. So if a process is blocked on a **wait**(S), then $\#\text{wait}(S)$ will not increase until the process is unblocked.

Example (Mutual Exclusion)

The no. of processes in their CS $= \#\text{wait}(S) - \#\text{signal}(S)$. Let's use this to show our usual properties.

Safety Properties

Mutual Exclusion

We know:

1. $v = 1 + \#\text{signal}(S) - \#\text{wait}(S)$ (our first semaphore invariant)

Safety Properties

Mutual Exclusion

We know:

1. $v = 1 + \#\text{signal}(S) - \#\text{wait}(S)$ (our first semaphore invariant)
2. $v \geq 0$ (our second semaphore invariant)

Safety Properties

Mutual Exclusion

We know:

1. $v = 1 + \#\text{signal}(S) - \#\text{wait}(S)$ (our first semaphore invariant)
2. $v \geq 0$ (our second semaphore invariant)
3. $\#\text{CS} = \#\text{wait}(S) - \#\text{signal}(S)$ (our observed invariant)

Safety Properties

Mutual Exclusion

We know:

1. $v = 1 + \#\text{signal}(S) - \#\text{wait}(S)$ (our first semaphore invariant)
2. $v \geq 0$ (our second semaphore invariant)
3. $\#\text{CS} = \#\text{wait}(S) - \#\text{signal}(S)$ (our observed invariant)

From these invariants it is possible to show that $\#\text{CS} \leq 1$, i.e. mutual exclusion.

Safety Properties

Mutual Exclusion

We know:

1. $v = 1 + \#\text{signal}(S) - \#\text{wait}(S)$ (our first semaphore invariant)
2. $v \geq 0$ (our second semaphore invariant)
3. $\#\text{CS} = \#\text{wait}(S) - \#\text{signal}(S)$ (our observed invariant)

From these invariants it is possible to show that $\#\text{CS} \leq 1$, i.e. mutual exclusion.

Absence of Deadlock

Assume that deadlock occurs by all processes being blocked on **wait**, so no process can enter its critical section ($\#\text{CS} = 0$).

Safety Properties

Mutual Exclusion

We know:

1. $v = 1 + \#\text{signal}(S) - \#\text{wait}(S)$ (our first semaphore invariant)
2. $v \geq 0$ (our second semaphore invariant)
3. $\#\text{CS} = \#\text{wait}(S) - \#\text{signal}(S)$ (our observed invariant)

From these invariants it is possible to show that $\#\text{CS} \leq 1$, i.e. mutual exclusion.

Absence of Deadlock

Assume that deadlock occurs by all processes being blocked on **wait**, so no process can enter its critical section ($\#\text{CS} = 0$). Then $v = 0$, contradicting our semaphore invariants above. So there cannot be deadlock.

Liveness Properties

To simplify things, we will prove for only two processes, p and q.

Eventual Entry for p (with weak semaphores)

Assume that p is starved, indefinitely blocked on the **wait**.

Liveness Properties

To simplify things, we will prove for only two processes, p and q.

Eventual Entry for p (with weak semaphores)

Assume that p is starved, indefinitely blocked on the **wait**. Therefore $S = (0, L)$ and $p \in L$.

Liveness Properties

To simplify things, we will prove for only two processes, p and q.

Eventual Entry for p (with weak semaphores)

Assume that p is starved, indefinitely blocked on the **wait**. Therefore $S = (0, L)$ and $p \in L$.
We know therefore, substituting into our invariants:

1. $0 = 1 + \#\text{signal}(S) - \#\text{wait}(S)$

Liveness Properties

To simplify things, we will prove for only two processes, p and q.

Eventual Entry for p (with weak semaphores)

Assume that p is starved, indefinitely blocked on the **wait**. Therefore $S = (0, L)$ and $p \in L$.
We know therefore, substituting into our invariants:

1. $0 = 1 + \#\text{signal}(S) - \#\text{wait}(S)$
2. $\#\text{CS} = \#\text{wait}(S) - \#\text{signal}(S)$

Liveness Properties

To simplify things, we will prove for only two processes, p and q.

Eventual Entry for p (with weak semaphores)

Assume that p is starved, indefinitely blocked on the **wait**. Therefore $S = (0, L)$ and $p \in L$.
We know therefore, substituting into our invariants:

1. $0 = 1 + \#\text{signal}(S) - \#\text{wait}(S)$
2. $\#\text{CS} = \#\text{wait}(S) - \#\text{signal}(S)$

From which we can conclude that $\#\text{CS} = 1$.

Liveness Properties

To simplify things, we will prove for only two processes, p and q.

Eventual Entry for p (with weak semaphores)

Assume that p is starved, indefinitely blocked on the **wait**. Therefore $S = (0, L)$ and $p \in L$.
We know therefore, substituting into our invariants:

1. $0 = 1 + \#\text{signal}(S) - \#\text{wait}(S)$
2. $\#\text{CS} = \#\text{wait}(S) - \#\text{signal}(S)$

From which we can conclude that $\#\text{CS} = 1$. Therefore q must be in its critical section and $L = \{p\}$.

Liveness Properties

To simplify things, we will prove for only two processes, p and q.

Eventual Entry for p (with weak semaphores)

Assume that p is starved, indefinitely blocked on the **wait**. Therefore $S = (0, L)$ and $p \in L$.
We know therefore, substituting into our invariants:

1. $0 = 1 + \#\text{signal}(S) - \#\text{wait}(S)$
2. $\#\text{CS} = \#\text{wait}(S) - \#\text{signal}(S)$

From which we can conclude that $\#\text{CS} = 1$. Therefore q must be in its critical section and $L = \{p\}$.
We know (or rather, assume) that eventually q will eventually finish its CS and **signal**(S).

Liveness Properties

To simplify things, we will prove for only two processes, p and q.

Eventual Entry for p (with weak semaphores)

Assume that p is starved, indefinitely blocked on the **wait**. Therefore $S = (0, L)$ and $p \in L$.
We know therefore, substituting into our invariants:

1. $0 = 1 + \#\text{signal}(S) - \#\text{wait}(S)$
2. $\#\text{CS} = \#\text{wait}(S) - \#\text{signal}(S)$

From which we can conclude that $\#\text{CS} = 1$. Therefore q must be in its critical section and $L = \{p\}$.
We know (or rather, assume) that eventually q will eventually finish its CS and **signal**(S).
Thus, p will be unblocked, causing it to gain entry — **Contradiction**.

Rendezvous

In addition (and perhaps simpler) than the mutual exclusion/critical section problem, the *rendezvous* problem is also a basic unit of synchronisation for solving concurrency problems.
Assume we have two processes with two statements each:

Rendezvous	
P	**Q**
$first_P$ $second_P$	$first_Q$ $second_Q$

Rendezvous

In addition (and perhaps simpler) than the mutual exclusion/critical section problem, the *rendezvous* problem is also a basic unit of synchronisation for solving concurrency problems. Assume we have two processes with two statements each:

Rendezvous	
P	**Q**
$first_P$	$first_Q$
$second_P$	$second_Q$

Problem

How do we ensure that all *first* statements happen before all *second* statements?

In Java

Producer-Consumer

Binary semaphores (aka locks) are not the only use of semaphores!

Producer-Consumer

Binary semaphores (aka locks) are not the only use of semaphores!

Producer-Consumer Problem

A producer process and a consumer process share access to a shared buffer of data. This buffer acts as a queue. The producer adds messages to the queue, and the consumer reads messages from the queue. If there are no messages in the queue, the consumer blocks until there are messages.

Producer-Consumer

Binary semaphores (aka locks) are not the only use of semaphores!

Producer-Consumer Problem

A producer process and a consumer process share access to a shared buffer of data. This buffer acts as a queue. The producer adds messages to the queue, and the consumer reads messages from the queue. If there are no messages in the queue, the consumer blocks until there are messages.

Algorithm 1.3: Producer-consumer (infinite buffer)			
queue[T] buffer $\leftarrow$ empty queue; semaphore full $\leftarrow (0, \emptyset)$			
	producer		**consumer**
	T d		T d
	forever do		**forever do**
p1:	d $\leftarrow$ produce	q1:	**wait**(full)
p2:	append(d, buffer)	q2:	d $\leftarrow$ take(buffer)
p3:	**signal**(full)	q3:	consume(d)

Finite buffer

What if the buffer has finite space, and we don't want to lose messages?

Finite buffer

What if the buffer has finite space, and we don't want to lose messages?

Use another semaphore!

Finite buffer

What if the buffer has finite space, and we don't want to lose messages?

Use another semaphore!

Algorithm 1.6: Producer-consumer (finite buffer, semaphores)	
bounded[N] queue[T] buffer $\leftarrow$ empty queue semaphore full $\leftarrow (0, \emptyset)$ semaphore empty $\leftarrow (N, \emptyset)$	
producer	**consumer**
T d	T d
loop forever	loop forever
p1: d $\leftarrow$ produce	q1: **wait**(full)
p2: **wait**(empty)	q2: d $\leftarrow$ take(buffer)
p3: append(d, buffer)	q3: **signal**(empty)
p4: **signal**(full)	q4: consume(d)

This pattern is called *split semaphores*.

A specific Example

Algorithm 1.7: Producer/Consumer (*b*-place buffer, sem's)			
	integer data[b]		
	semaphore empty $\leftarrow (b, \emptyset)$, full $\leftarrow (0, \emptyset)$		
	producer		**consumer**
	integer i $\leftarrow$ 0		integer k $\leftarrow$ 0, t $\leftarrow$ 0
	loop forever		loop forever
p1:	wait(empty)	q1:	wait(full)
p2:	data[i % b] $\leftarrow g$(i)	q2:	t $\leftarrow$ t + data[k % b]
p3:	i++	q3:	k++
p4:	signal(full)	q4:	signal(empty)

What do we prove?

The crucial properties of this pair of processes include:

safety $S = \left(t = \sum_{j=0}^{k-1} g(j)\right)$ is an invariant

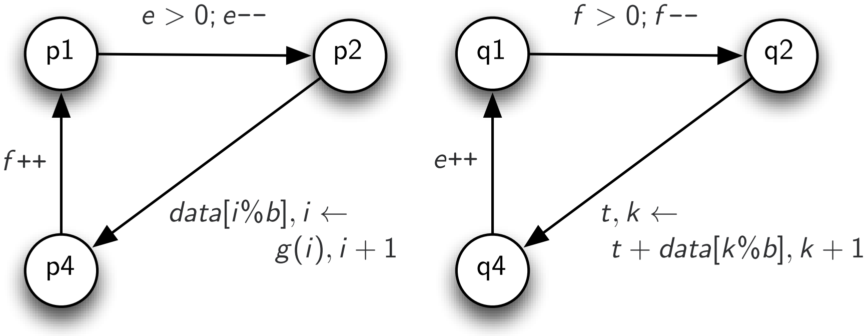
liveness k keeps increasing

How do we prove?

To show the safety property, we

1. translate the pseudo code into transition diagrams,
2. define a pre-condition ϕ
3. define an assertion network Q,
4. prove that Q is (a) inductive and (b) interference-free,
5. prove that the initial assertions Q_{p1} and Q_{q1} follow from ϕ, and
6. prove that each of the consumer's assertions implies the invariant S.

1 Transition Diagrams



2 Precondition

As precondition we collect the initial values of those global and local variables which are read before they are written.

$$\phi = (e = b \land f = 0 \land i = k = t = 0)$$

3 Assertion Network I

We start by collecting further likely invariants.

The consumer can't overtake the producer:

$$0 \leq k \leq i \quad (1)$$

The producer can't lap the consumer:

$$i - k \leq b \quad (2)$$

The buffer shows a subsequence of g's values:

$$\forall j \in a..i-1\,(data[j\%b] = g(j)) \quad \text{, where } a = \max(0, i-b) \quad (3)$$

3 Assertion Network II

semaphore invariants:

$$e, f \in 0..b \tag{4}$$

$$e = b + \#\mathrm{signal}(e) - \#\mathrm{wait}(e) \tag{5}$$

$$f = \#\mathrm{signal}(f) - \#\mathrm{wait}(f) \tag{6}$$

numbers of waits and signals are correlated:

$$\#\mathrm{wait}(e) = \#\mathrm{signal}(f) + 1 - \mathrm{p}_1 = i + \mathrm{p}_2 \tag{7}$$

$$\#\mathrm{signal}(f) = \#\mathrm{wait}(e) - \mathrm{p}_{2,4} = i - \mathrm{p}_4 \tag{8}$$

$$\#\mathrm{wait}(f) = \#\mathrm{signal}(e) + 1 - \mathrm{q}_1 = k + \mathrm{q}_2 \tag{9}$$

$$\#\mathrm{signal}(e) = \#\mathrm{wait}(f) - \mathrm{q}_{2,4} = k - \mathrm{q}_4 \tag{10}$$

3 Assertion Network III

semaphore values are correlated:

$$e + f = b - \mathsf{p}_{2,4} - \mathsf{q}_{2,4} \tag{11}$$

our goal:

$$S \tag{12}$$

Assuming that the invariants (1)–(12) gather all that's going on we may now try to prove that the assertion network consisting of the same assertion,

$$\mathcal{I} = (1) \wedge \ldots \wedge (12)$$

at every location is inductive and interference-free.

4(a) Q is inductive

We need to prove local correctness of each of the 6 transitions. We assume that the auxiliary variables p_1, p_2, p_4, q_1, q_2, and q_4 are implicitly set to 0 resp. 1, depending on the locations.

$$\text{p1} \rightarrow \text{p2:} \models \mathcal{I} \wedge e > 0 \implies \mathcal{I} \circ (e \leftarrow e - 1) \tag{13}$$

$$\text{p2} \rightarrow \text{p4:} \models \mathcal{I} \implies \mathcal{I} \circ (data[i\%b], i \leftarrow g(i), i + 1) \tag{14}$$

$$\text{p4} \rightarrow \text{p1:} \models \mathcal{I} \implies \mathcal{I} \circ (f \leftarrow f + 1) \tag{15}$$

$$\text{q1} \rightarrow \text{q2:} \models \mathcal{I} \wedge f > 0 \implies \mathcal{I} \circ (f \leftarrow f - 1) \tag{16}$$

$$\text{q2} \rightarrow \text{q4:} \models \mathcal{I} \implies \mathcal{I} \circ (t, k \leftarrow t + data[k\%b], i + 1) \tag{17}$$

$$\text{q4} \rightarrow \text{q1:} \models \mathcal{I} \implies \mathcal{I} \circ (e \leftarrow e + 1) \tag{18}$$

4(b) Q is interference-free

Finally it pays off to give such a degenerate assertion network: *interference-freedom comes for free* since we've proved inductivity (local correctness) already.

5 ϕ is strong enough

Since all assertions are the same, we only need to show that (at p1 and q1):

$$\phi \implies \mathcal{I}$$

which is straightforward.

6 S follows from Q

Trivially true since S is the last conjunct of $\mathcal{I}$.

Liveness

Deadlock Freedom

The only global location with a potential for deadlock would be p_1/q_1.
Constant $b > 0$ and invariant (11) ensure that at p_1/q_1, not both semaphores can be 0.

Liveness

Deadlock Freedom

The only global location with a potential for deadlock would be p_1/q_1.
Constant $b > 0$ and invariant (11) ensure that at p_1/q_1, not both semaphores can be 0.

Liveness Property

Suppose one of the processes (say the consumer) is stuck at location 1 forever, and thus k does not increase.

Liveness

Deadlock Freedom

The only global location with a potential for deadlock would be p_1/q_1.
Constant $b > 0$ and invariant (11) ensure that at p_1/q_1, not both semaphores can be 0.

Liveness Property

Suppose one of the processes (say the consumer) is stuck at location 1 forever, and thus k does not increase.
Then, by deadlock-freedom, the producer would have to keep going indefinitely without ever incrementing f—but it does so every round.

What Now?

Next lecture, we'll be looking at Monitors and the Readers and Writers problem.

This week's homework involves Java programming. There's a number of resources (prepared by Vladimir Tosic) on the website to assist you.

Assignment 1 is also coming out this week.