The Concept and Use of Semaphores

Vlad Wojcik, Computer Science

"Every man, wherever he goes, is encompassed by a cloud of comforting convictions, which move with him like flies on a summer day"

(Bertrand Russell, Sceptical Essays, 1928, "Dreams and Facts")

DEFINITION:

Semaphore: A data structure, initialized at boot time of the machine, masquerading as a non – negative integer

PERMISSIBLE OPERATIONS:

Given a semaphore s, two non-divisible operations are defined:

```
signal(s) // increments s by one
wait(s) // decrements s by one as soon as it is possible
```

Notes:

PURPOSES:

- 1. Enforcement of mutual exclusion
- 2. Synchronization (between loosely coupled processes)

ENFORCEMENT OF MUTUAL EXCLUSION:

Critical section: section of program code not simultaneously available to several processes

Wrong (naïve) solution:

while (gate == closed) continue; gate := closed; // Critical section code goes here; gate := open;

Correct solution using a semaphore named mutex, initialized to 1:

```
wait(mutex);
// Critical section code goes here;
signal(mutex);
```

Practical example: adding / removing items from a queue (mutex initialized to 1):

Adding process	<u>Removing process</u>
•	•
•	•
<pre>wait(mutex);</pre>	<pre>wait(mutex);</pre>
add item to queue;	remove item from queue;
<pre>signal(mutex);</pre>	<pre>signal(mutex);</pre>
•	•
•	•

SYNCHRONIZATION:

We have two processes A and B. We require that A should not proceed beyond point L1 until B reaches point L2. We use a semaphore **proceed** initialized to 0.

<u>Code of A</u>	<u>Code of B</u>
•	•
<pre>L1 : wait(proceed);</pre>	L2 : signal(proceed);
•	•
•	•

PRACTICAL EXAMPLE:

We have a pool of producer processes and another pool of consumer processes. Items of information created by producers are disposed of by consumers. The producers deposit their items in a buffer of capacity N. The consumers remove items in order to dispose of them.

Reasons for synchronization of access to the buffer (of capacity N, contents n):

- It is impossible to extract items if n = 0
- It is impossible to deposit items if n = N
- Buffer access is critical

Semaphores used:	mutex initialized to 1
	<pre>space_available initialized to N</pre>
	item available initialized to 0

Producer processes	<u>Consumer processes</u>
•	•
•	•
repeat forever:	repeat forever:
begin	begin
produce item;	<pre>wait(item_available);</pre>
<pre>wait(space_available);</pre>	<pre>wait(mutex);</pre>
<pre>wait(mutex);</pre>	<pre>extract item from buffer;</pre>
deposit item in buffer;	<pre>signal(mutex);</pre>
<pre>signal(mutex);</pre>	<pre>signal(space_available);</pre>
<pre>signal(item_available);</pre>	consume item;
end	end

A NASTY BUG CHALLENGE:

Find the bug in this solution:

Semaphores used:	mutex initialized to 1
	space_available initialized to N
	item available initialized to 0

Producer processes	<u>Consumer processes</u>
•	•
•	•
repeat forever:	repeat forever:
begin	begin
<pre>produce item;</pre>	<pre>wait(mutex);</pre>
<pre>wait(space_available);</pre>	<pre>wait(item_available);</pre>
<pre>wait(mutex);</pre>	<pre>extract item from buffer;</pre>
deposit item in buffer;	<pre>signal(mutex);</pre>
<pre>signal(mutex);</pre>	<pre>signal(space_available);</pre>
<pre>signal(item_available);</pre>	consume item;
end	end

GLOBAL SEMAPHORE:

A semaphore available to a number of processes.

Each process is allowed to perform both **wait** and **signal** operations on this semaphore.

TYPICAL USE: to protect mutually exclusive operations.

PRIVATE SEMAPHORE:

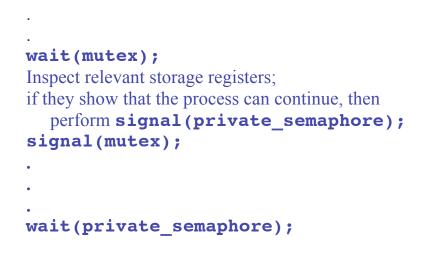
A semaphore available to a number of processes.

Each process is allowed to perform **signal** operations on this semaphore, but only one process is allowed to perform the **wait** operation.

TYPICAL USE: by processes wishing to check if they may proceed.

EXAMPLE:

Whenever a process has to decide if it can continue, the sequence of operations is:



NOTE: **mutex** - global semaphore protecting the examination of registers (initially 1); **private_semaphore** - initially 0.

EXAMPLE:

When a process reaches a stage where one or more other processes may have become free to proceed, the sequence of operations is:

wait(mutex); Inspect and modify relevant storage registers; perform signal operations on the appropriate private semaphores; signal(mutex); The semaphore system also formalizes the means whereby a process can safely perform "privileged operation(s)" on behalf of other process(es).

Case study: disk transfers:

Usually the access to the disk is a privilege reserved to the disk manager process. It has its own private semaphore DM, and there is a communication area in which the details of transfers required by client processes are placed. The disk manager can place there the feedback information as well.

This area may constitute a queue of requests of disk transfers. The global semaphore Q protects this queue. It is set initially to *n*-1, where *n* is the maximum number of requests that can be queued.

When a process wants a disk transfer to be performed on its behalf, the sequence of instructions is as follows:

```
wait(Q);
wait(mutex);
record details of transfer on queue;
signal(mutex);
signal(DM);
wait(private-semaphore);
wait(mutex);
read answer-back information from the communications area;
signal(mutex);
```

The sequence of operations of a disk manager process:

```
START: wait(mutex);
read details of transfer from the queue;
pop-up the queue;
signal(mutex);
signal(Q);
perform the requested transfer to/from a disk;
wait(mutex);
record answer-back information in communications area;
signal(mutex);
signal(private_semaphore) -- of the client process;
wait(DM) -- on its own private semaphore;
goto START;
```

WARNING:

These examples illustrate the scheduling and synchronization problems only!