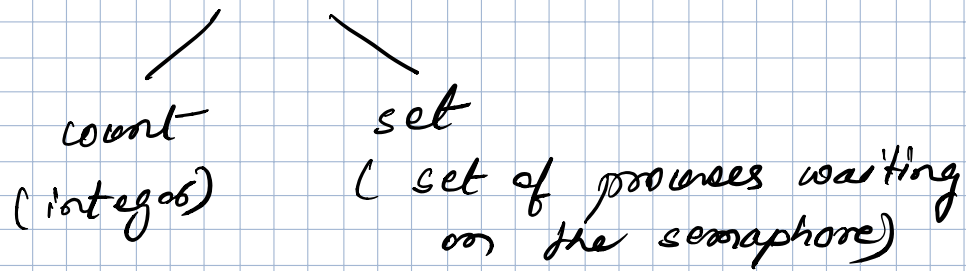


Semaphores

- * Provide high-level mutual exclusion.
- * Permit designers to specify multiple critical sections.
- * Allow independent control of each critical section.
- * Provide an access policy (FIFO)
- * When there are fight for resources.
- * Fairness (counting semaphore)
- * Semaphore = a variable allocated for each item to be protected.
There is semaphore variable for each shared resource.
- * Applications must be programmed in such a way that it uses semaphore before accessing each shared item.
- * Semaphore mechanism guarantees that only one process can access shared item at any given time.
- * No busy-waiting.
- * Semaphores are created dynamically.
- * An id is associated with each semaphore. Typically an integer.
0 - 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99

* Two fields



* Operation on semaphores:-

- ① `semcreate()`: create a new semaphore
- ② `semdelete()`: delete an existing semaphore
- ③ `wait()`:

- /
- Decrements count
- Adds calling process to set of waiting processes if resultant count is negative

④ `signal()`:

- /
- Increments count
- Make a process ready if any is waiting.

* Can complement both:-

- ① Mutual exclusion
- ② Producers / consumers.

Mutual exclusion

- ① Create a mutex semaphore
- ② Only one process can access critical section at any time (others blocked)
 $sid = \text{semcreate}(1);$ → only one process is allowed
 $\text{wait}(sid);$

$\text{signal}(sid);$ (as soon as critical section is over)

Producers - Consumers

For this case there are 2 semaphores.

Each semaphore will hold information for producers and consumers respectively.

Producers: Producer semaphore stores information like no. of resources available for producers to work. Everytime a producer want to execute, it needs to acquire producer semaphore. Producer should only produce as much as fits in buffer.

Consumers: consume amount by consumers. Initialization should not exceed this

$psem = \text{semcreate}(\text{buffer-size});$
 $csem = \text{semcreate}(0);$ → consumers available item in

consume is 0.

Producers
while (1) {
 wait (psem);
 // acquire psem

 signal (csem);
 // signals csem to use resource
}

Consumers
while (1) {
 // acquire csem
 wait (csem);

 signal (psem);
 // signal producer
}

* If there are more consumers and producers.

✓ Producers puts resource inside buffer.

✓ Consumers takes out the resource inside buffer.

* If there are n process waiting for a semaphore that means semaphore count is $-n$.

* If $n > 0$ means queue is empty and there are resources available to use.

XINU Implementation of Semaphore

* XINU implements semaphores, not locks

* There is an array containing semaphore entries.

Each entry:- corresponds to one entry.
 index count

array of pointers

Pointer to list of processes.

- * Semaphore ID is index into array.
- * FIFO (policy)
- * Xtime allows a certain no. of semaphores.
- * When waiting on a semaphore a process
 - not executing
 - not ready
 - not suspended
- * Suspended state is only used by suspend and resume.
- * WAITING :- Process blocked on a semaphore.