Semaphores

* Provide high-level mutual exclusion.
* Permit designers to specify multiple critical sections.
* Allow codependent 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

* Two fields

count (integer) — set (set of processes waiting on the semaphore)

* Operation on semaphores :-

① semcreate() : create a new semaphore
② semdelete() : delete an existing semaphore
③ wait() :
- Decrements count
- Adds calling proces to set of waiting processes if resultant count is negative

④ signal() :
- Increments count
- Make a process ready if any is waiting.

* Can implement both :-
① Mutual exclusion
② Producer / consumer

Mutual exclusion

(1) Create a mutex semaphore

(2) Only one process can access critical section at any time (others blocked)

```
sid = semcreate(1);   → only one process is allowed
wait (sid);

signal (sid);  (as soon as critical section is over)
```

Producer – Consumer

In this case there are 2 semaphores. Each semaphore will hold information for producer and consumer respectively.

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

Consumer : Consume amount by consumer.

Initialization

```
psem = semcreate(buffer-size);   → producer (should not exceed this)
csem = semcreate(0);   → consumer (available item to
```

consumer is 0.

Producer

```
while (1) {
    wait (psem);

    signal (csem);
}
```

acquire psem

signals csem to use resource

Consumer

```
while (1) {
    wait (csem);

    signal (psem);
}
```

acquire csem

signal produces

* If there are more consumers and producers.
  - Produces puts resource inside buffer.
  - Consumer 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.
  - Entries count =

Pointer to list of processes.

* Semaphore ID is index into array.
* FIFO (policy)
* Xinu 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.