

Name: Ron Tal

ID: 300890191

### **Semaphores:**

Semaphore is a data type that is used for controlling access, by multiple processes, to a common resource.

It basically indicates how many units of a particular resource are available.

Semaphores are counters for resources shared between processes.

A good example would be the producer/consumer problem -

The consumer waits for the producer to produce something if the queue is empty.

The producer must wait for the consumer to consume something if the queue is full.

With Semaphores we would track it with two Semaphores: 'EmptyCounter' & 'FullCounter'.

1st counter indicates the amount of empty places, 2nd counter indicates the amount of full places.

Empty places and Items represent our resources, and the semaphores maintain control over these resources.

If there were several producers and one consumer, we can use a Mutex to represent the Consumers availability, for example: 'useQueue'

This way, we can verify the integrity of the data when concurrent producers are writing to it.

Basically, a mutex is the same thing as a binary semaphore, the main differences between semaphores & mutexes are:

**a.** Mutex can have an owner, which is the process that locked the mutex.

and only the locking process can unlock it - a Semaphore has no concept of an owner.

**b.** Unlike semaphores, Mutexes provide priority inversion safety - so no higher priority task get's locked and lower priority tasks are being done instead of it.

Since the mutex knows its current owner, it can promote the priority of the owner whenever a higher-priority task starts waiting on the mutex.

**c.** Mutexes also provide deletion safety, where the process holding the mutex cannot be accidently deleted - Semaphores do not provide this.

### **Architecture explanation:**

First I added system calls regularly as done in previous assignment.

I created Semaphore.c & Semaphore.h

I added to the make file, under OBJS, the 'semaphore\' value so it knows to compile them.

### **Semaphore.h:**

This contains only the declaration of the semaphore struct:

```
struct semaphore {  
    char name[6];           //a unique name for the "cookie jar"  
    int max;                //indicates the maximum amount of "cookies" possible to be in the "jar"  
    int numOfItems;         //indicates the number of "cookies" in the "jar"  
    int numOfListeners;     //indicates the number of processes connected to the "jar"  
    struct spinlock slock; //personal lock for this "cookie jar"  
};
```

See my documentation of the fields.

The reason I added the 'numOfListeners' is in order to be more efficient - without it, everytime I close against a semaphore, I would need to run on all the process table and check if I am the last one opened against this semaphore.

Instead, I just hold a simple counter on the semaphore, so once the last listener closes his connection, the sTable recognizes it and kills this semaphore.

This way, there is good loose decoupling between a process, a semaphore and the sTable - everyone is responsible for himself !

**Semaphore.c:** This file creates the global stable and allocates for it the NSEM amount of semaphores allowed in the system.

In addition to that, it holds all the implementations of functions relevant for semaphores, such as:

*semaphoreinit* - initializes the stable lock

*semaphorealloc* - allocates an available semaphore from the global stable to a requesting process.

*semaphoreopen* - opens a connection against a semaphore.

*semaphoreclose* - closes a connection against a semaphore.

*semaphoresignal* - signals a semaphore and increases it's value by 1.

*semaphorewait* - decreases a semaphore value by 1 or wait's for it to have a value.

*semnamecmp* - compares a semaphore name with a given string (for some reason I wasn't allowed to directly call fields of semaphores from proc.c, so I checked against file.c and saw they are doing the same...)

*incNumOfListeners* - used only inside 'Fork' in order to increment the number of listeners in a semaphore when they are added through a fork.

*getSemName* - returns a semaphore's name.

**\*Please see my documentation in the functions code - I elaborated a lot about each step.**

**Proc.c:** The system calls in here are just an abstraction for the kernel that really calls the function's in *semaphore.c* - this way, all the integrity checks are being done in proc.c , and only if the data is valid, the functions in 'Semaphore.c' are being called.

As expected, all my functions from semaphore.c were added to defs.h too.

### User Program Example:

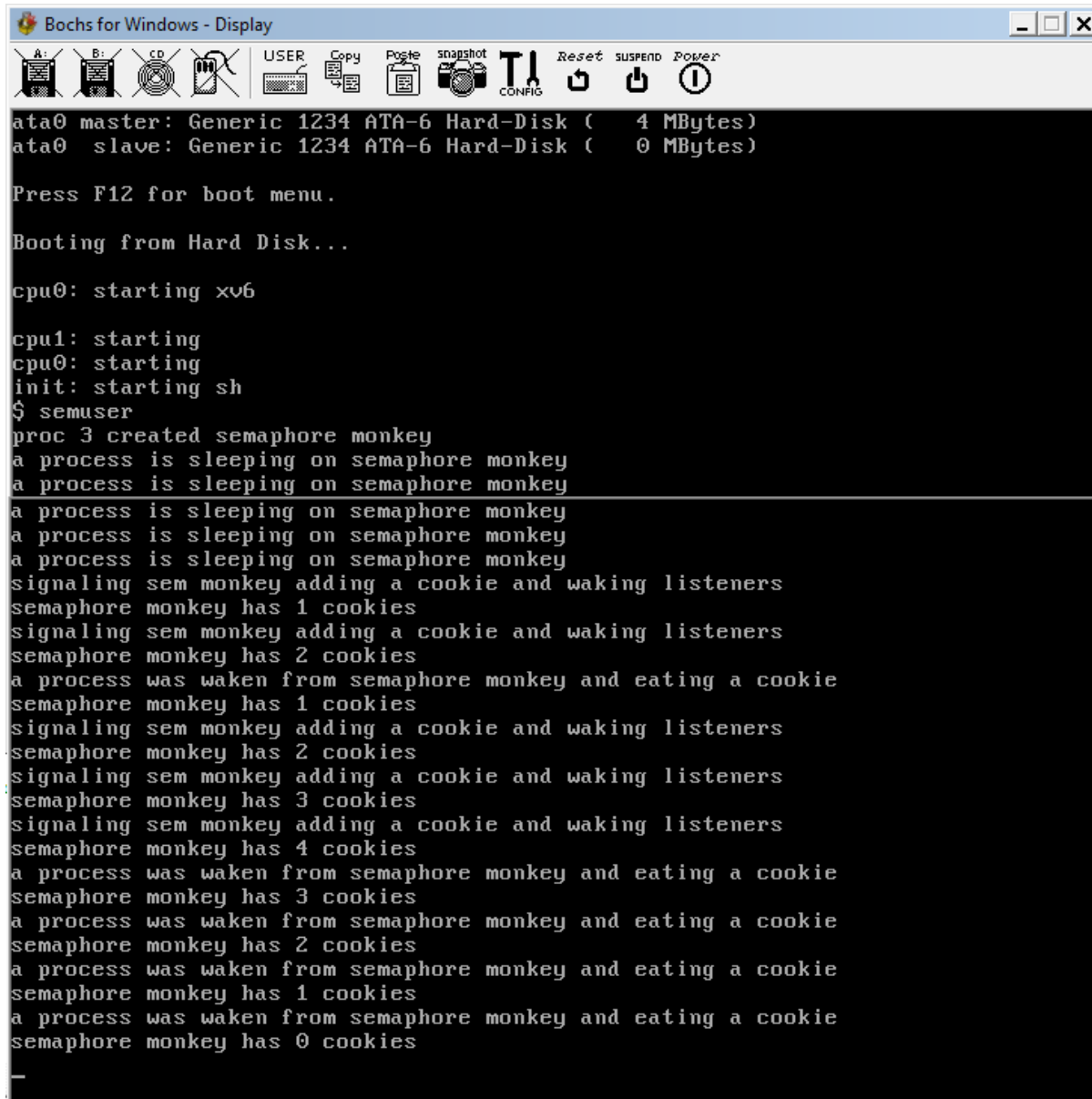
This is the output of my user program 'semuser.c'.

The parent creates a semaphore called 'monkey'.

The parent creates 5 child processes that are opened against 'monkey' due to the 'fork' logic.

Then the parent add's 5 cookies to semaphore 'monkey' and the child's are waiting on semaphore 'monkey'.

You can see how cookies are being added and eaten - exactly as the example above of the producer\consumer scenario.



```
Bochs for Windows - Display
A: B: CD USER Copy Paste Snapshot CONFIG Reset SUSPEND Power
ata0 master: Generic 1234 ATA-6 Hard-Disk ( 4 MBytes)
ata0 slave: Generic 1234 ATA-6 Hard-Disk ( 0 MBytes)

Press F12 for boot menu.

Booting from Hard Disk...

cpu0: starting xv6
cpu1: starting
cpu0: starting
init: starting sh
$ semuser
proc 3 created semaphore monkey
a process is sleeping on semaphore monkey
a process is sleeping on semaphore monkey
a process is sleeping on semaphore monkey
a process is sleeping on semaphore monkey
a process is sleeping on semaphore monkey
signaling sem monkey adding a cookie and waking listeners
semaphore monkey has 1 cookies
signaling sem monkey adding a cookie and waking listeners
semaphore monkey has 2 cookies
a process was waken from semaphore monkey and eating a cookie
semaphore monkey has 1 cookies
signaling sem monkey adding a cookie and waking listeners
semaphore monkey has 2 cookies
signaling sem monkey adding a cookie and waking listeners
semaphore monkey has 3 cookies
signaling sem monkey adding a cookie and waking listeners
semaphore monkey has 4 cookies
a process was waken from semaphore monkey and eating a cookie
semaphore monkey has 3 cookies
a process was waken from semaphore monkey and eating a cookie
semaphore monkey has 2 cookies
a process was waken from semaphore monkey and eating a cookie
semaphore monkey has 1 cookies
a process was waken from semaphore monkey and eating a cookie
semaphore monkey has 0 cookies
```