

Name: Ron Tal

ID: 300890191

Mutex – Mutual Exclusion:

A mutex is an object the synchronizes concurrent uses to shared resources , processes use it to control access to a shared resource.

A Mutex can be locked to indicate a resource is in use, then other processes can get blocked by waiting for the resource, or do other tasks.

Mutex is useful for protecting shared data structures from concurrent modifications.

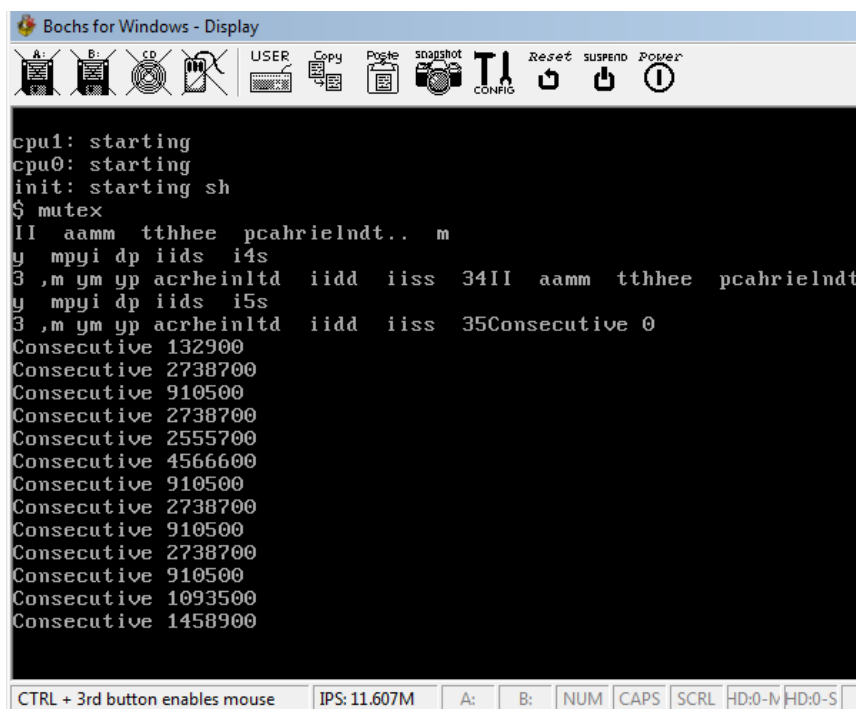
A mutex can be in two possible states: Unlocked (not owned by any process), and locked (Owned by one process).

A Mutex can never be owned by two processes simultaneously.

So after implementing the system calls and adding mutex_acquire(1) and mutex_release, I am getting multiples of 300, although it works real slow and takes a while until it prints one of them.

Furthermore, the numbers are repeated randomly.

(why is the text in the beginning coming out as gibrish???)



```
Bochs for Windows - Display
A: B: CD USER Copy Paste snapshot CONFIG Reset SUSPEND Power
cpu1: starting
cpu0: starting
init: starting sh
$ mutex
II aamm tthhee pcahrielndt.. m
y mpyi dp iids i4s
3 ,m ym yp acrheintltd iidd iiss 34II aamm tthhee pcahrielndt
y mpyi dp iids i5s
3 ,m ym yp acrheintltd iidd iiss 35Consecutive 0
Consecutive 132900
Consecutive 2738700
Consecutive 910500
Consecutive 2738700
Consecutive 2555700
Consecutive 4566600
Consecutive 910500
Consecutive 2738700
Consecutive 910500
Consecutive 2738700
Consecutive 910500
Consecutive 1093500
Consecutive 1458900
CTRL + 3rd button enables mouse IPS: 11.607M A: B: NUM CAPS SCRL HD:0-N HD:0-S
```

Now let's improve the code so each process can hold several mutexes.

First I configured a global variable NOMUTEX=5 in param.h

1. I modified the function 'mutex_release' to accept 'int id' as a parameter, instead of 'void' - as this time it needs to be told which mutex ID to release !

2. I changed in the 'struct proc' the field 'mutex id' to be int* instead of int.

So it will hold an array of the id's currently in use by the process.

defined it to be in size of NOMUTEX, (a global variable equals to 5 as defined in param.h)

3. in 'proc.c' under the mutex_acquire function I added this code:

```
if(!proc->mutex_acquired)
    proc->mutex_acquired = 0;
if (proc->mutex_acquired == NOMUTEX)
    return(-1);
```

It initializes proc->mutex_acquired to 0 if it is the first time it is being called.

It declines only if the process has already NOMUTEX amount of mutexes acquired.

I also changed this part:

int j;

loop:

```
for( p = ptable.proc; p < &ptable.proc[NPROC]; p++){
    for( j = 0 ; j < p->mutex_acquired; j++){
        if(p->mutex_id[j] == id){
            if( proc->killed){
                release (&ptable.lock);
                return(-1);
            }
            sleep((void *)id, &ptable.lock);
            goto loop;
        }
    }
}
```

For each process holding the resource I am interested in, I will run on all of it's mutex objects it holds - which are maximum 5 objects.

Once I find the process that uses the resource I am interested in, I am going to sleep and listening to the channel of the resource (id) ,and then I am passing the control to the scheduler (eventually...).

4. in 'mutex_acquire' I change this line:

```
proc->mutex_acquired++;
```

In order to increment the counter of mutex object's I hold, since I added a new one.

5. in mutex_release I changed the code to be to be:

```
int i;
if( !proc->mutex_acquired || proc->mutex_acquired == 0)
    return(-1);

for(i=0; i < proc->mutex_acquired; i++){
    if(proc->mutex_id[i] == id){
        proc->mutex_id[i] = proc->mutex_id[proc->mutex_acquired-1];
        proc->mutex_acquired--;
        wakeup((void *)id);
        break;
    }
}

return (0);
```

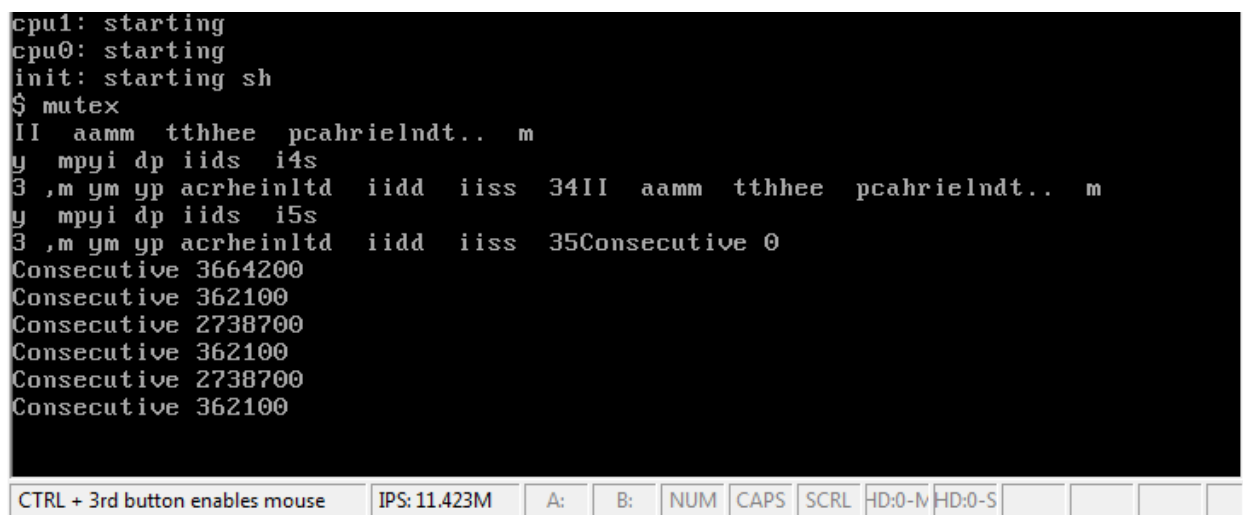
Here I basically first if mutex_acquired is zero or not initialized.

Then, I am running in a loop on the amount of mutex objects the process has, when I find the desired ID to release, I safely release it by preventing any cause of unwanted gaps in my array.

Eventually, I update the mutex_acquired by decreasing it by one.

Finally, I wake up the processes that are waiting for this id.

Eventually after modifying everything this is what I got when running the program in bochs:



```
cpu1: starting
cpu0: starting
init: starting sh
$ mutex
II aamm tthhee pcahrielndt.. m
y mpyi dp iids i4s
3 ,m ym yp acrheintltd iidd iiss 34II aamm tthhee pcahrielndt.. m
y mpyi dp iids i5s
3 ,m ym yp acrheintltd iidd iiss 35Consecutive 0
Consecutive 3664200
Consecutive 362100
Consecutive 2738700
Consecutive 362100
Consecutive 2738700
Consecutive 362100
```

CTRL + 3rd button enables mouse IPS: 11.423M A: B: NUM CAPS SCRL HD:0-IV HD:0-S

Summary

The user program does the following.

First a pipe is initialized with the standard input & standard output.

Then, the parent starts reading from the standard input into a buffer in the size of 300 Bytes, it is being done as long as the read didn't have any failure.

In each iteration, it counts the amount of consecutive identical bytes in the array.

Once it meets a byte that is different from the last consecutive series it prints the number of bytes that were identical & consecutive - and starts a new counting with the last byte read.

The two child processes first initialize a local buffer in the size of 300 bytes.

Each byte will have the same ASCII value, as we are computing into each byte the value ('a' + pid).

Then, each child, in enters an endless loop.

In each iteration, a child tries to acquire the Standard Output resource, which is ID 1.

And Writes from its buffer to the output (which is piped to the standard input that the parent listens to), an identical consecutive series of his bytes.

The reason for the 300 multiplications is because of an ASCII value multiplied by 300, giving us always a multiplication of 300.

After a child writes to the standard output, it releases it with `mutex_release`.

Appendix

When we fork after adding fields to a `proc`, they are being referenced to the child.

When we exit a process, the fields in the child are just being un-referenced.

In the `Exit()` system call, I modified the code to release all mutex objects it holds (wakeup them), in case any other processes were sleeping and waiting for a wakeup for any of these resources:

```
for(int i=0; i < proc->mutex_acquired; i++){  
    mutex_release(proc->mutex_id[i]);  
}  
proc->mutex_acquired = 0;
```