Synchronization: Semaphores

Harrison Vuong (hvuong@ucsc.edu)
David Zou (dzou@ucsc.edu)
Derek Frank (dmfrank@ucsc.edu)

1 Goal:

The goal of this assignment is to create a kernel service to support Semaphores and to use the created Semaphores to implement locks and condition variables.

2 Available Resources:

Minix Documentations
Interprocess Communications and Synchronization Slides

3 Design:

***file location instructions are located in the README*

struct semaphore
The semaphore structure and its members

```
    Variable to hold the value
    Variable to hold the id
    Variable to keep track of number of sleeping processes
    An array to keep track of processes that are asleep
    Variable index to keep track of next process to wake up
    Variable index to keep track of most recently put to sleep process
```

struct semaphore semarray[100]
An array to store all of the active semaphores

int semcount
A count to keep track of of how many active semaphores

void init_sem(void)
When the initial semaphore array is created, initialize all slots to have an initial value of 0 and id of -1.

```
For every item in the semaphore array
    every member of the semaphore struct is initiate to 0 except id where id is -1
The count of the number of semaphores in the semaphore array is made to 0
```

Int sem_exist(int sem)
The purpose of this function is to check and see if a semaphore already exists in the semaphore array before creating a new one.

```
    If the sem id is less than 0 or the id is greater than 2^32-1, then return 0
    if there is for some reason the semCount becomes negative then return 0
    for every semaphore in the semaphore array
        check to see if the id already exists
        if it does return 1
    return 0
```

Int do_semvalue (void)

semvalue will return the value for the specified sem id as well as do all necessary error checking prior to returning a value.

```
Set the exit status to error, assume an error unless all error checks pass
Receive the message and store sem id
if sem id is less than or equal to 0 or is greater than 2^31-1 or is negative
    return ESEMVAL
Set the index by sem id modulo the size minus 1
For all the semaphores in the array that is less than 100 and exit status is still error
    if the id is found, return the value and the loop is done
    else continue the loop until it's found

if the exit status is still and error status
    return the exit status

return the exit status, where exit status is the value if the id was found
```

Int do_semit (void)

calling this function will create a new semaphore a new semaphore and if successful, store it in the global semaphore array

```
If the semCount is 0 call init sems
Retrieve the messages and store the id and value variables
If there are more than 100 semaphores
    return EAGAIN
if the sem id is not 0 and the sem already exists
    return EEXIST
Variable to hold the index where the new semaphore will be
if sem id given is 0
    Iterate through the semaphore array and look for an empty array index
        if the semaphore id at the current index is -1, then the semaphore is unallocated
            pick an id based on current index that does not conflict with any current
            sem id in use
            after inserting the semaphore, increase semaphore count
            set exit status to the picked sem id
            allocate and initialize the semaphore with the id and value
            and the rest of the members to default values of 0 (ie. Process list is null)

else if the given sem id is not a 0
    since index range from 0 to 99, sem id range from 1 to 2^31-1, will need to subtract to
    find the index since after sem id modulo 100 returns a value between 1 and 100, subtract
    to estimate an index.
    Iterate through and find an empty index to store semaphore
        after finding an available space
        increment the number of semaphore for semcount
        set all the available members inside the semaphore to 0
        increment index when necessary
return the status
```

Int do_semup (void)

This functions wakes up a waiting process if there are more than one process in the process list queue for the semaphore

```
    variable to assume exit status is fail unless its set later in the code
    retrieve the message and set sem
    if the sem id is less than 0
        return EINVAL
    if the sem id is greater than the size limit for an id
        return EINVAL
    if the semCount is for some reason a negative
        return EINVAL

    locate the semaphore by general index region by using modulo
        iterate through region to find the matching sem
        note: this method is similar to binary search given that we start at around the
    region, and at worse case, have to iterate through the entire semaphore array

    if the semaphore can't be found
        return error

    if the sem value is greater than 10^6
        return EOVERFLOW

    increase the value for the semaphore

    if the process count for the semaphore is not 0
        variable to keep track of  first process of process list
        set proc number to be first process in process list
        If process number is less than or equal to 0 or is greater than 1000
            return error
        if the first process (stored as next) is the last proces in process list
            clear the members and make 0
        if reaches the end of array
            loop around and set begin and end accordingly
        set next to 0
        increment next index
        set begin to 0
        decrease the process list counter for semaphore
        get the pointer to mproc structure using proc number as index
        set reply  so the process can stop waiting
    return the exit status
```

Int do_semdown (void)

This function controls the semaphore by adding the process to a process list when the value goes below 0.

```
    set the default exit status
    retrieve the message
    if sem id is less than 0 or greater than limit size or less than 1
        return EINVAL
    find the closest region where the index is
        for the size of the semaphore array
            find the sem id in the cloest region
    if the sem id can't be found
        return error
```

```
if the semaphore value is as large as the negative max value
    return EOVERFLOW
check to see if the process will fit in the process list
decrement the value for the semaphore
if the value for the semaphore is less than 0
    find the first available space in process list
    set the process number
    do the necessary error checking before adding to process list
    after adding to list increment the process list counter
suspend the process
return exit status
```

Int do_semfree (void)

the purpose of this function is to free a semaphore that is not currently being in use

```
set defaul exit
retrieve the message
if sem id is less than 0 or greater than limit size or less than 1
    return EINVAL
do necessary error checking before index
    go through indexes to find a match

if exist status is still error then return the error
if the process count for semaphore is not 0
    return EBUSY
decrease the semCount
clear the semaphore
clear the process
return exit status
```

PUBLIC int seminit(int sem)

the purpose of this is to send a message to the kernel. This would be the same on all of the other system calls.

```
Create a message;
package the message
send the message to the desired system call
```

4 Testing:

First tested that seminit allocated a semaphore when passing a sem id of 0. Next, implemented separated processes that can up and down a semaphore given an id. Specifying an order to run such processes, we could determine whether semaphores were working correctly because a process would print a message or stall depending on the number of up and down semaphore processes created.

The main goal is to test if each of the individual calls worked and did not fail the error checking that is necessary prior to executing the system call.