CSCI 360 - Lab 4 Report

Sami Al-Qusus

Oct 9, 2018

Assignment Goals:

- Strict alternation without blocking.

Assignment Instructions (Dr. LoPinto):

- Strict alternation within the operating system may be bad but strict alternation between application programs may be vital. Create two processes that log entries into a shared memory. Use two semaphores to ensure the two processes make alternating entries. There should never be two consecutive entries from the same process. Think of the producer-consumer system with only one common buffer.
- Be aware that there may be two different semaphore packages on your machine - the POSIX semaphores and the System V semaphores. Don't mix them. Here's an example using POSIX. Run the example more than once and report your findings.
- In addition, each log entry should contain a name for the logging process as well as the number of nanoseconds since it made it's last entry. You can use zero for the first log entries.

Example program explained:

- #include <semaphore.h> //semaphore library
  - Link with *-pthread*.
- #include <fcntl.h> //file control library
- sem_t // Declares the semaphore
- **sem_open**()- creates a new POSIX semaphore or opens an existing semaphore.
  - The first argument is the name by which the semaphore is identified.
  - Second arg is the *oflag* argument it specifies flags that control the operation of the call.
  - If **O_CREAT** is specified in *oflag*, then the semaphore is created if it does not already exist.
  - On success, **sem_open**() returns the address of the new semaphore.
  - On error, **sem_open**() returns **SEM_FAILED.**
- **int sem_getvalue(sem_t *_sem_, int *_sval_)-** places the current value of the semaphore pointed to *sem* into the integer pointed to by *sval*.
  - In the example program semaphore A holds a garbage value. We place that into variable “value” and print it:
    - The value of semaphore A is 1869948720
- sem_init(semA,0,5); - sets the value of the semaphore
  - The first argument is for initializing the address pointed to by semA.
  - The second argument indicates whether this semaphore is to be shared between the threads of a process, or between processes.

        - If 0, then the semaphore is shared between the threads of a process, and should be located at some address that is visible to all threads (e.g., a global variable, or a variable allocated dynamically on the heap).
        - If nonzero, then the semaphore is shared between processes, and should be located in a region of shared memory. (e.g . shmget).

        - Note: Any process that can access the shared memory region can operate on the semaphore using sem_post(3), sem_wait(3)...

    - The third argument is the value we wish to initialize it to (in our example a 5).
- sem_wait(semA); - decrements (locks) the semaphore pointed to by *semA.*
    - We can also use sem_post() to increment.
- sem_destroy(semA) – destroy semaphore at the address pointed to by semA.
    - Destroying a semaphore that other processes or threads are currently blocked.
    - Failure to do this can result in resource leaks on some implementations.
    - Returns 0 on success; on error, -1 is returned.

Example code output after first run:

```
alqususs@otter:~/csci360/lab04$ ./aSender
Sender A
The value of semaphore A is 1
Now the value is 5
This is loop 0
This is loop 1
This is loop 2
```

Example code second, third and fourth run gave the following output:

```
alqususs@otter:~/csci360/lab04$ ./aSender
Sender A
The value of semaphore A is 2
Now the value is 5
This is loop 0
This is loop 1
This is loop 2
Semaphore destroyed
```

The reason that the value of semaphore A goes from 1 to 2 when running the second time is because its not getting destroyed. That's because **sem_destroy** or **sem_close** only closes the semaphore, also done when a process exits. But the semaphore still remains in the system. So what's happening is that the value starts as 2 because sem_open doesnt create a new semaphore since it already exists and it just uses the last sem value after all the increments.

- When I increased the i condition to 7 instead of 3 and ran it multiple times I got the following:

```
alqususs@otter:~/csci360/lab04$ ./aSender
Sender A
The value of semaphore A is 0
Now the value is 5
This is loop 0
This is loop 1
This is loop 2
This is loop 3
This is loop 4
```

  - This shows that now the value is 0 because of the same problem.
  - Note that the program blocks and waits after the fifth loop, it waits for the semaphore to be incremented to proceed.

- To completely destroy it, I used sem_unlink("/lab4a") which removes the named semaphore referred to by *name*. The semaphore is destroyed once all other processes that have the semaphore open close it.

What I did:

- I first observed and understood the example program, then explained piece by piece as shown above.
- I then replicated my sender/receiver programs from last week and changed them up to work with semaphore instead of busy waiting.
  - I used two semaphores to insure alternation.
- I made the sender create both semaphores and shared memory and receiver destroys them once the user quits.
- I added code to measure the time each process took since its last entry.
- Note: for more details, please read step-by-step comments made in receiver and sender files.

What I learnt:

- Most of what I learnt is in the "Example program explained" section.
- The most important thing I learnt was how to use semaphores instead of busy waiting.
  - Including how to use the create, open, destroy, unlink, initialize, post, wait functions on semaphores.
- I also learnt how to use the clock function to measure elapsed time in nanoseconds.

Credits:

- Dr. LoPinto lan3 and lab4 examples.
- http://man7.org/linux/man-pages/man3/sem_open.3.html
- https://www.systutorials.com/docs/linux/man/3-sem_unlink/
- https://stackoverflow.com/questions/15164484/when-to-call-sem-unlink