Inter Process Communication Design Document

POSIX Libraries used

* Semaphore.h

Shared Memory Design

There will be 2 pointers to Shared Memory each of which points to one of the following:

* Queue of pid
* Binary Semaphore for queue
* Byte variable.
* Binary Semaphore for byte variable.

Service Daemon

The service daemon will do the following from within an infinite loop:

1. It will lock semaphore on shared memory queue and dequeue first pid from shared memory queue. Then it will unlock semaphore and callback the first async process function. If no pid was returned from dequeue, it will go to sleep for 30 seconds and try again.
2. If pid was retrieved from queue, service daemon will try to acquire semaphore lock on byte variable, if successful, It will place some random bytes here and unlock the semaphore on this byte variable.
3. Then it will raise SIGUSR2 using kill function to pid.

Synchronous Processes

The synchronous call processes will follow the following steps:

1. They will try to acquire semaphore lock on queue using **sem\_wait()**.. They will wait till they have acquired the lock.
2. Once they have acquired the lock, they will put their pid on the queue and go to sleep.
3. They will receive a SIGUSR2, upon which their SIGUSR 2 handler will try to acquire the semaphore lock on the Byte variable. They will wait until they have acquired the lock.
4. Once they have acquired the lock, they will retrieve the random bytes, unlock the byte semaphore and exit.

Asynchronous Processes

The asynchronous call processes will follow the following steps:

1. They will try to acquire semaphore lock on the queue using **sem\_tryandwait()**. If they won’t they will get an error. At this point they will register a callback function with the semaphore and continue with their next task.
2. When the semaphore is next free , it will use the callback function to allow the process to enqueue its pid. The process will then be able to lock the semaphore.
3. Once, the async process has enqueued its pid it will continue on its next task.
4. They will receive SIGUSR2, upon which their SIGUSR2 handler will try to acquire the semaphore lock on the byte variable. If they won’t they will get an error. At this point they will register a callback function with the semaphore and continue with their next task.
5. When the semaphore on the byte variable is next free, it will use the callback function to allow the process to access the byte variable. The process will then be able to lock the semaphore.
6. Once the async process has retrieved the byte variables, it will unlock the byte semaphore and exit.

Issues:

* What does it mean to register a callback function?
  + If the process was not able to acquire the lock, then it will call a registerCallback((void) \* callMe()) function which belongs to a service daemon. This process maintains an array of function pointers. It loops through all of them, everytime the lock is removed from the queue.
* All processes will need to maintain a struct which contains the SIGUSR2 handler.
* Processes need to have an indicator telling whether they are async or sync.

References

<http://pubs.opengroup.org/onlinepubs/009695399/functions/sem_trywait.html>

<http://linux.die.net/man/3/sem_trywait>

<http://www.advancedlinuxprogramming.com/alp-folder/alp-ch05-ipc.pdf>