Homework Assignment #3
CSE-506 (Fall 2015)

Team Members

Himanshu Sharma: 110453943

Rahul Rishi Sharma: 110347475

Kumar Amit: 110395825

Design

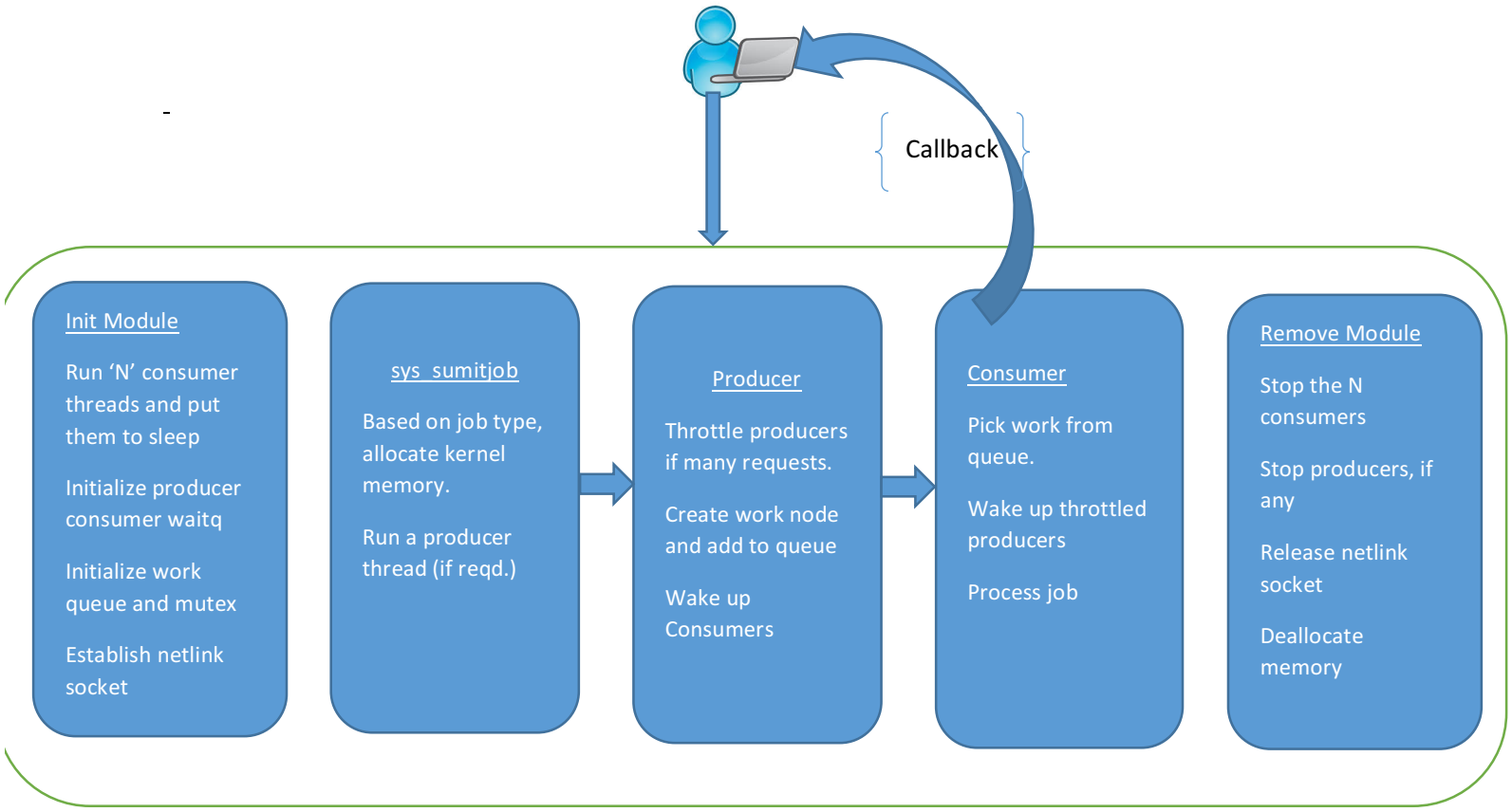


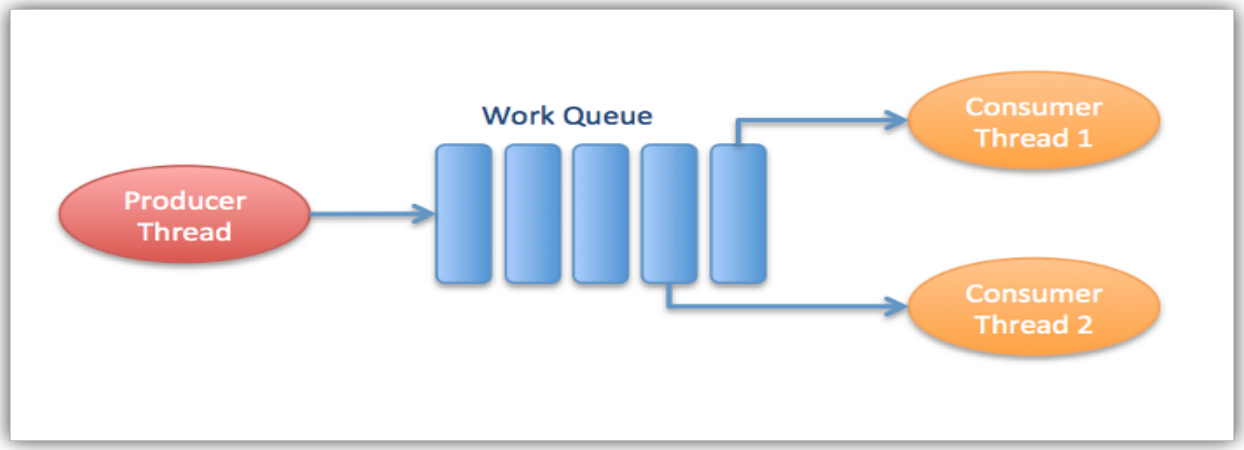
Fig 1

Fig 1 shows the basic design of the project. The user can use the system call sys_submitjob to send request to run various kinds of jobs like compress a file, concat multiple files etc. The user also has the option to choose from two types of callbacks Netlink socket and signal.

Functionality Implemented

Producer Consumer

The problem implements two entities, the producer and the consumer who share a common, fixed sized linked list as work queue.



Below are the steps on how the project implemented.

Init Module

At the time of module init, N consumer threads are instantiated and since the work count at this point is zero, all the threads are put to sleep till there is signal that at least one element is added to the work queue. Below is the list of objects initialized at init time:

Producer and Consumer waitq
Empty work queue.
Mutex to prevent concurrent access to work queue.
N Consumer kernel threads
Netlink socket established using the prefixed socket family

System Call

For any request the user sends to the kernel, the system call allocates the required kernel memory for the objects in the job and runs (creates) a producer thread and passes the job object to the producer.

Producer

The producer is responsible for adding the job to the work queue and notifying the consumer. A mutex is used for protecting the work queue from being accessed by more than one kernel threads simultaneously. The work queue has a MAX_LENGTH, if the number of pending requests exceed the MAX_LENGTH the newly created producers are added to the producer waitq (*producer_queue*).This is how throttling the producers is achieved. As soon as the work count becomes 1 the consumers are notified by calling the wake_up_interruptible routine on the consumer_queue.

Consumer

As already stated, the consumer is responsible for removing the element from the queue and processing it. If there are no jobs to be processed (empty work queue), the consumer goes to sleep. For every job removed from the queue the global work count is decremented and if there are any throttled producers, they are woken up. The de-queued job is then processed based on the job type.

Listed below are the list of features supported:

a. **Encrypt/Decrypt (Job Type = 1)**

   The encrypt/decrypt feature encrypts/decrypts a given file using the Linux Crypto API. It sets a different value of the IV for each chunk of bytes encrypted/decrypted and supports multiple ciphers. It also stores the cipher/passkey used for encrypting in the preamble in an encrypted way used for verification during decryption.
   The user also has an option to rename/rewrite the input file.

   Ex:
   ./xhw3 -j 1 -c aes -p passkey -e -i "/usr/src/hw3-cse506g06/hw3/api-intro.txt" -o "/usr/src/hw3-cse506g06/hw3/api-introe.txt"

   ./xhw3 -j 1 -c aes -p passkey -e -i "/usr/src/hw3-cse506g06/hw3/api-intro.txt"

b. Compress/Decompress (2)

The compress/decompress feature allows to compress/decompress files and give the user an option to either get the output file by overwriting the input file or get a new output file. It uses the Linux Crypto API for the compression/decompression.

Ex:

./xhw3 -j 2 -a deflate -c -i "/usr/src/hw3-cse506g06/hw3/api-intro.txt" -o "/usr/src/hw3-cse506g06/hw3/api-introc.txt"

c. Checksum (3)

Using this feature, user can calculate checksum of given file. As system call is asynchronous so user will submit job and when job will finish using callback (netlink socket) consumer thread will send checksum to user. To know checksum user should start a netlink socket in different thread so call will not be blocked and can received checksum sent by consumer thread.

In checksum we are supporting two hash algorithm MD5 and SHA-1 and if user will give other than this user code will return an error.

E.g. ./xhw3 –j 3 –i f1.txt –a md5 –z 1

E.g. ./xhw3 –j 3 –i f1.txt –a sha1 –z 1

Note: here z is priority you want to assign that particular job.

d. Concatenate (4)

Using this feature, the user can concatenate multiple files of any size into a single file. Errors for cases like if the files provide are regular files, intermediary read/write fails etc. are handled. Provide the input files as a list of comma separated files with the '-i' option, and the output file with the '-o' option.

E.g. ./xhw3 –j 4 –i f1.txt,f2.txt,f3.txt –o out.txt

e. LIST (5)

Lists the pending jobs in the work queue. List will show jobs in their priority order or order in which consumer thread will pick them. First job in work queue will be picked first.

E.g. ./xhw3 –j 5

JOBID | PID | PRIORITY

1 |14353| 2

f. **Remove (6)**

Remove a job from the work queue. Use –i option for jobid and –p for pid.

E.g. ./xhw3 –j 6 –i 1 –p 14353

g. Modify (7)

Modify priority of a job in work queue. If a job is in work queue and waiting for consumer to pick it, in meantime if user wants to change priority of that job user need to send –i jobID, -p pid and –z new priority.

E.g. ./xhw3 –j 7 –i 1 –p 14353 –z 3

Remove Module

At the time of remove module, we make sure that all the memory still allocated, say for pending jobs is freed. The running consumers and the producers are stopped; also, the netlink socket opened for callback is released.

Callback Mechanism

We have implemented callbacks in two ways:

a. Netlink sockets

During the Init module, the kernel creates a netlink socket configured for sending messages to the userspace. The protocol family used is defined in the job.h file which is shared between the user and the kernel.

The user passes its PID as part of the struct job, which is used by the netlink socket to send message to the userspace at the end of each system call.

The user if using this mechanism needs to setup a socket and wait for the message from kernel.

b. Signals

Since signals allows 32 bits of data to be send back to the user, features like compress/decompress which just needs to communicate the success/failure/error to the user space can make use of this

callback mechanism, avoiding the slightly more complicated setting up of a socket.