CSE-506 (Spring 2014) Homework Assignment #3

Submitted By: Kumar Sourav

Date: 08/05/2014

Contents

1. Using the code:

Use sh install_module.sh command to compile the code. It will compile the code and insmod the .ko file.

Basic Flags and their meaning:

The program uses various flags to give user options for running the code. Various flags with their meaning are described below:

- -x : Calculate CRC32 checksum and write to a file
- -e : Encrypt file and write the cipher text to a file
- -d : Decrypt file and write the plain text to a file
- -O : Overwrite the input file (Used with Encryption/Decryption only)
- -R : Rename the input file (Used with Encryption/Decryption only)
- -r : Remove a job from the queue

- -a : Remove all jobs from the queue
- -M : Perform MD5 checksum and write to a file (Extra Credit)
- -S : Perform SHA1 checksum and write to a file (Extra Credit)

Default flag is –x, which is to calculate CRC32 checksum and write to a file with .crc32 appended to its filename.

2. System Design:

High level design is explained in this section. The system uses a user level module called xhw3.c and a kernel level module called sys.xjob.c. There is only one producer thread and there are two consumer threads. Tasks which I have implemented are: Encryption/Decryption (AES blockcipher) and Checksum (crc32, MD5 and SHA1).

Producer and Consumer has been implemented producer and consumer using a FIFO Queue. There are two queues namely main queue and wait queue. Main queue has length 7 and wait queue also has length 7. When the user submits a job it is enQueued into the main queue, if the number of jobs in main queue is 7 then the next job is inserted into the wait queue. This process is explained in the Figure 1.

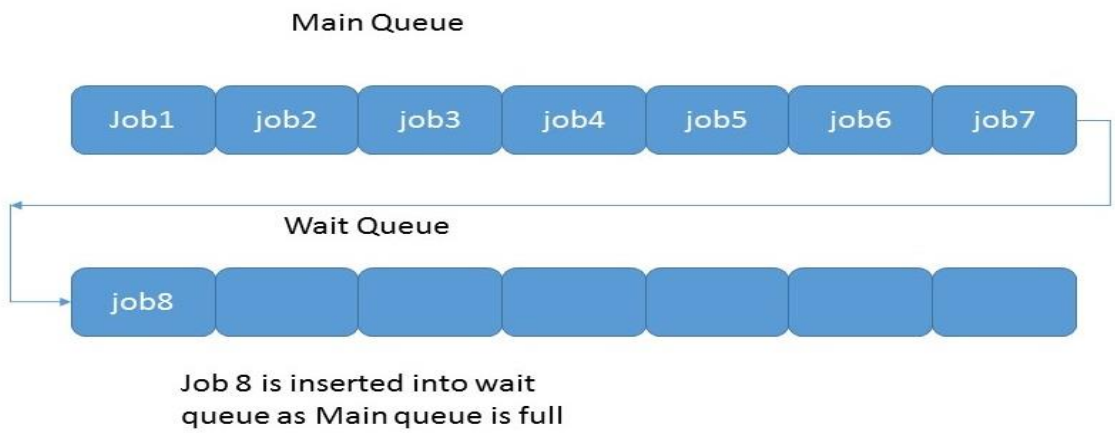

Figure 1

When a job is deQueued from the main queue by consumer qlen is decreased by one and when a job is deQueued from wait queue by consumer qlen_wait is decreased by one. When consumer deQueues a job from main queue, it checks if qlen_wait > 0, if it is true then it deQueues a job from wait queue and enQueues it into the main queue. This phenomenon is described in Figure 2.

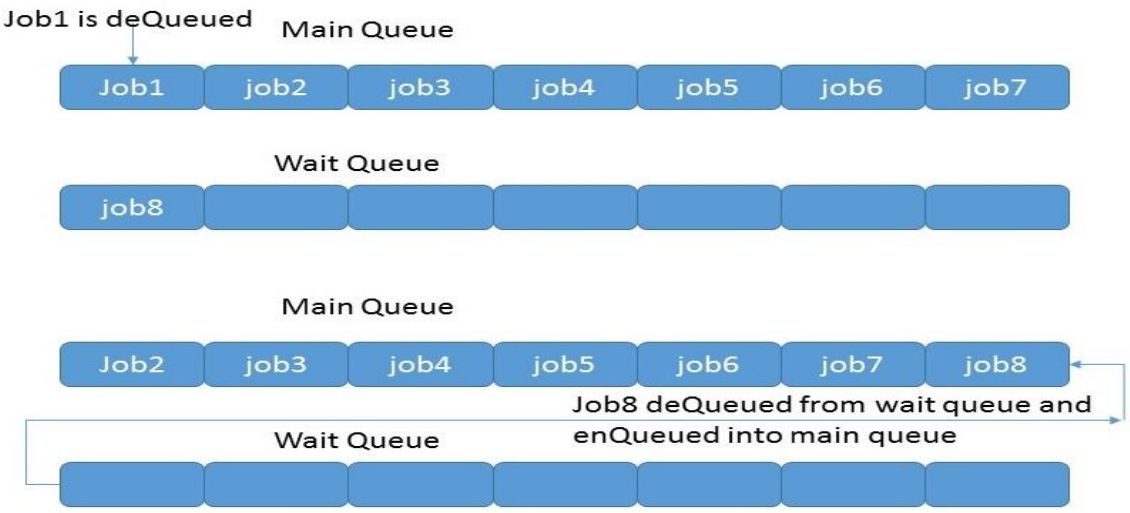

Figure 2

When the main queue and wait queue are both full (that is of length 7 each), the producer is inserted into wait_queue (wait_queue defined in wait.h). Producer is only woke up by consumer when it deQueues a job from queue. Similarly when both main queue and wait queue are empty the consumer is inserted into wait_queue and it is only woke up by producer when the producer enQueues a job.

The jobs which are inserted into the queues are taken by consumer to perform one of the operations from encryption/decryption and checksum. The producer exits after enQueuing the job into the queue returning the error status to the user module whereas the consumer performs the operation on the enQueued job and signals back to user about the completion of the task only if the user program is still running. The overall flow of the system is shown in the Figure 3.

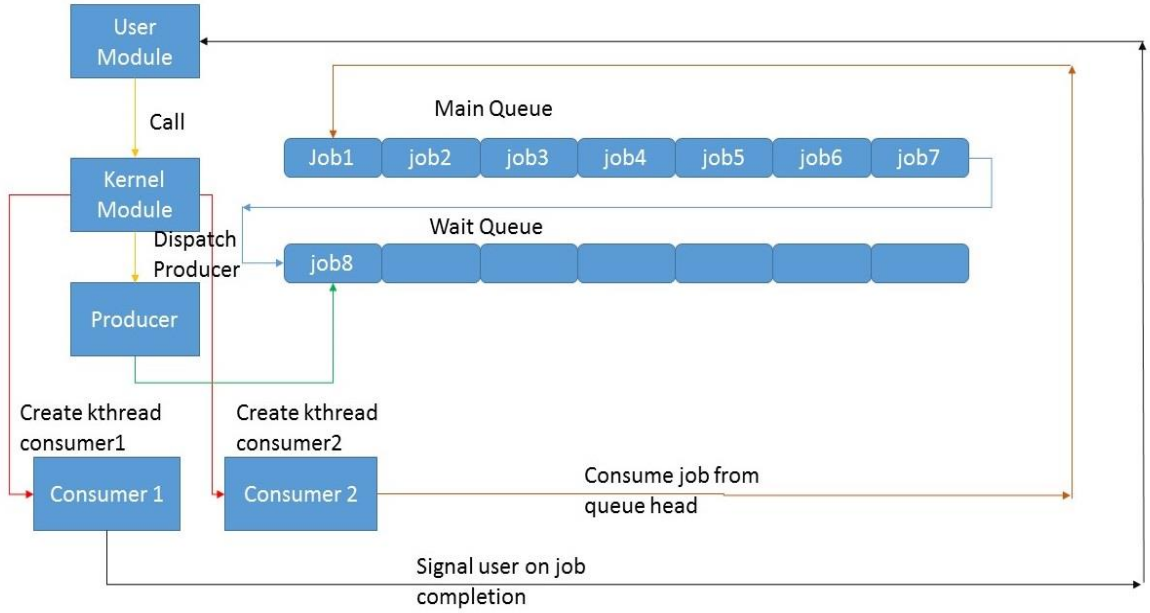

Figure 3. System overall working

3. Major Data Structures:

The major data structures used are: job, node defined in common.h. Details of both data structures are described below:

1. Struct job{
   Char **files;
   Int pid;
   Int type;
   Int count;
   Int out_ind;
   Void *sig_area;
   };

- Char **files: This field stores all the files and key in case of encryption. Consumer performs all the operations on the files stored in this space. A job can contain upto 6 files.
- Int pid: This is a unique process id which makes every process unique.
- Int type: This field stores the type of operation to be performed on the files like: Encryption/Decryption and Checksum.
- Int count: This field displays the count of the files which are in a job.
- Int out_ind: This field tells whether to overwrite a file or rename a file for output.
- Void *sig_area: This field is used for signalling. Consumer checks this variable to check if it can perform copy_to_user on this area. If copy_to_user is successful then Consumer will send signal to the user program.

2. Struct node{
   Struct job job;
   Struct node *next;
   };

   Struct node is a queue element which contains a job structure to store job and a pointer *next to the next queue element.

4. Features Implemented:

4.1 Encryption:

For Encryption I have used Crypto API. The encryption algorithm used is AES block cipher. This algorithm takes file names and Encryption key as input from the user. It reads plain text from the given files and uses the key to encrypt the plain text to form cipher text. This cipher text is overwritten to the same file or it is written to a renamed file depending on the out_ind value

of the job structure. In default case the cipher text is overwritten to the same file.

4.2 Decryption:

For Decryption also I have used Crypto API. The decryption algorithm used is AES block cipher. This algorithm takes file names and Decryption key as input from the user. It reads cipher text from the given files and uses the key to decrypt the cipher text to form plain text. This plain text is overwritten to the same file or it is written to a renamed file depending on the out_ind value of the job structure. In default case the plain text is overwritten to the same file.

4.3 Checksum:

For Checksum I have used three algorithms of Crypto API. I have implemented CRC32, MD5 and SHA1 algorithms. MD5 and SHA1 are part of extra credit. For Checksum the user enters the file names on the console and the consumer performs relevant checksum operation and stores the checksum result into Filename.crc32 or Filename.md5 or Filename.sha1 depending on the algorithm used.

5. Major Functions/Methods:

To modularize the assignment the operations are divided into functions. Some of the important functions are described below:

5.1 Int main (xhw3.c):

This is the main user level function which takes arguments from the console, packs them into the job structure and sends them to system call as void pointer.

5.2 Long xjob (sys_xjob.c):

This is the main kernel level function which receives the user argument, copies them to kernel space and calls the Producer () method. It also returns the error or success message to the user.

5.3 Int Producer (sys_xjob.c):

The producer's responsibility is to enQueue job into the queue after checking length of the queue. If enQueue is successful it returns success to the user, else returns appropriate error message. If both queues are full it goes to sleep.

5.4 Int Consumer (sys_xjob.c):

The consumer's responsibility is to deQueue job from the queue and perform the task on the job. After completion of the task it sends a signal to the user program. If both queues are empty then consumer thread goes to sleep.

5.5 Int perform_task (utils.h):

This function is called by consumer to perform a task. It contains logic to computer checksum, encrypt file and decrypt file. It calls the appropriate function based on the value of job type passed by the user.

6. Locking Semantics and signaling:

Locking has been implemented using mutex lock. Locking is mainly done in two functions namely Producer () and Consumer () in sys_xjob.c. There is one lock for queues for both producer and consumer. Producer and consumer hold this lock in various situations. Some of the situations where lock is used in producer and consumer are:

1. Acquire mutex before producer or consumer checks value of qlen (main queue length) or qlen_wait (wait queue length).
2. Acquire mutex before producer enQueues a job into main queue or wait queue.
3. Acquire mutex before consumer deQueues job from main queue or the wait queue.

For signalling between Kernel space to user space I have used sigaction. User makes a sigaction listener and listens to a signal from kernel. Kernel makes a siginfo object and embeds current job id and error no into it. On receiving a signal from kernel user activates a function which copies value of job id and error no into static variables, user then displays the job id and error no on the terminal.

7. Extra Credit:

For extra credit I have implemented two more algorithms for checksum namely MD5 and SHA1 both using Crypto API. Both are defined in utils.h.

Both algorithms take filename as argument. After calculating the checksum MD5 writes the checksum into a file named as Filename.md5 and SHA1 writes into a file named Filename.sha1. They return error or success to the calling function.

Appendix1: Some basic operations and their syntax:

If you don't specify a flag then default task performed by consumer is checksum.

1. View queue jobs: ./xhw3.c –l
2. Remove all jobs from queues: ./xhw3 –a
3. Remove one job: ./xhw3 –r (job id)
4. Encrypt with overwrite: ./xhw3 -e (key) (file1) (file2)…
5. Encrypt with Rename: ./xhw3 –eR (key) (file1) (file2)…
6. Decrypt with overwrite: ./xhw3 -d (key) (file1) (file2)…
7. Decrypt with Rename: ./xhw3 –dR (key) (file1) (file2)…
8. Checksum crc32: ./xhw3 –x (file1) (file2)…
9. Checksum MD5: ./xhw3 –M (file1) (file2)…
10. Checksum SHA1: ./xhw3 –S (file1) (file2)…