# An Asynchronous and Concurrent Producer Consumer Problem

**CSE-506 (HW-3)**

Submitted by:

Mohit Goyal (110349552)

Arjun Bora (110310134)

## 1. INTRODUCTION

This assignment is about an asynchronous and concurrent processing of the producer consumer problem. It involves kernel locking and callbacks. This project has been developed as a system call in the form of loadable/unloadable kernel module. The system call deals with performing costly file operations concurrently and send results back to the user in an asynchronous way. This project is about processing jobs asynchronously and proper handling of shared data between jobs.

## 2. FEATURES

This project supports below operation to be performed on the input files:

a. Encryption/Decryption

- It performs encryption and decryption of the files using the given key and initialization vector.
- While performing decryption it checks for the validity of the key which was preamble into the encrypted file during encryption.
- The key is MD5 hashed before saving into the encrypted file to increase security.
- It supports AES and BLOWFISH encryption algorithms.
- It created temporary file using the ID of the job to avoid any inconsistency as many jobs can be running concurrently and then deletes or rename it to output file depending upon the options.
- Output file depends upon the options given by user, whether she wants the file to be renamed to output file (-n) or overwrite the input file (-o) with encrypted/decrypted file.

b. Compression/Extraction

- It performs compression and extraction.
- While performing extraction it checks for the size of the compressed buffer which was stored during compression in the starting of each buffer of size PAGE_SIZE.
- It works with DEFLATE compression algorithm.
- Temporary file is created using the ID of the job as a prefix to avoid any inconsistency as many jobs can be running concurrently and can overwrite other job's output file.
- Output file depends upon the options given by the user; if they want the file to be renamed to output file or overwrite the input file with compressed/extracted file.

c. Calculate Checksum

- A checksum is small-size information which is used for detecting errors or validation of files.
- It calculates checksum of the files using MD5 and SHA1 algorithms.
- Calculated checksum is sent back to the user on shell by netlink callback after finishing the job.

d. Concatenate 2 or more files into output file
    - Given 2 or more files it concatenates the files into one after another into to output files. The maximum number of files that can be concatenated is 10.

## 3. USAGE

This project is developed as loadable kernel module. The user is required to insert the sys_submitjob.ko module in the kernel using below command to start the module:

```
# insmod sys_submitjob.ko
```

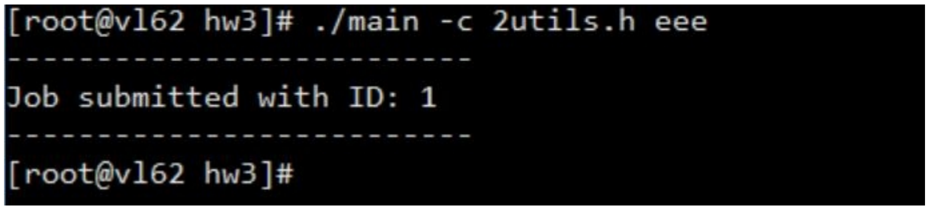
User program "main" takes the input from the user from the shell. It performs exhaustive validation check regarding the syntax of the input arguments and return an appropriate error if the arguments specified by user are not valid. The required format of the inputs to "main" program is:

```
# ./main [operation] infile(s) outfile [key] [options]
```

Following are the operations supported in this project on the given files:

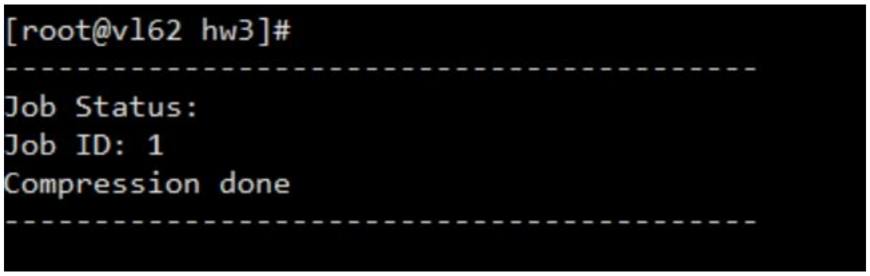
a. **Encryption [-e]:** This option is used to perform encryption on given file with key using crypto APIs. If key is not given with this option, it will return error.
b. **Decryption [-d]:** This option is used to perform decryption on given file with key. Key should be same which was used to encrypt file, otherwise it will return error message through netlink as the decryption is not valid.
c. **Compression [-c]:** This option is used to perform compression operation on given file.
d. **Extraction [-x]:** To perform extraction operation on given file, this option can be used. Only compressed file can be extracted. If uncompressed file is given to extract, it will return error.
e. **Checksum [-s]:** To find the checksum of the file, this option can be used. It will calculate the checksum and return it to user on netlink callback.
f. **Concatenation [-a]:** This option is used to perform concatenation of files into output file. If number of input files are less than 2 it will return error.

Below is the screenshot of Job Submission:



```
[root@vl62 hw3]# ./main -c 2utils.h eee
-------------------------
Job submitted with ID: 1
-------------------------
[root@vl62 hw3]#
```

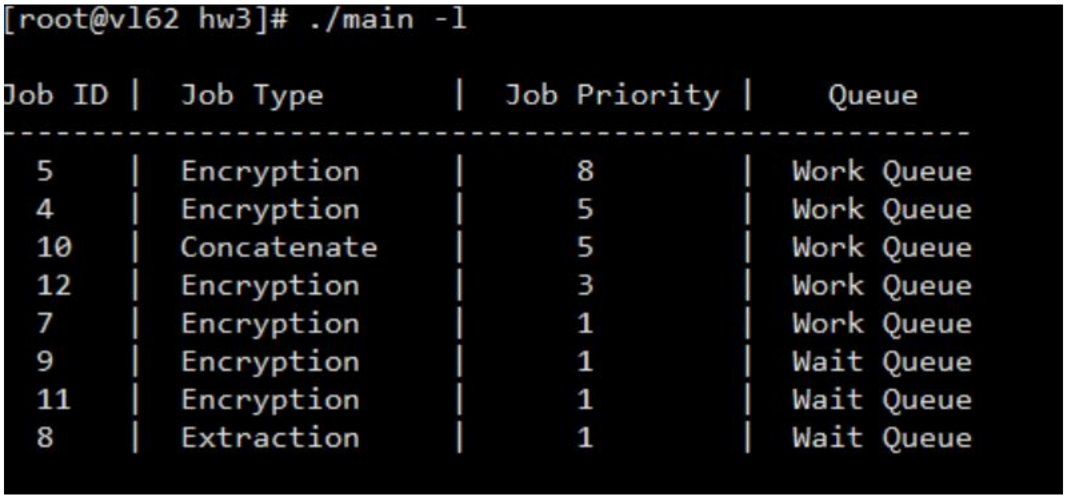
After Successful completion of the job by the consumer threads, it will send back a success or failure message:



```
[root@vl62 hw3]#
------------------------------------------
Job Status:
Job ID: 1
Compression done
------------------------------------------
```

g. **List [-l]:** To list all the jobs in the queue, this option can be used. It will return all the jobs pending to be executed in the queue. It will return the jobs with ID, Operation, Priority and status of job whether it is “work queue” or “wait queue”. The system call does not use netlink to send back this list of all the queued jobs. It writes this information in one of the field of the job struct, which user program reads.

   Below is the screenshot of list operation:



```
[root@vl62 hw3]# ./main -l

Job ID |  Job Type        |  Job Priority |    Queue
----------------------------------------------------
  5    |  Encryption      |      8        |  Work Queue
  4    |  Encryption      |      5        |  Work Queue
  10   |  Concatenate     |      5        |  Work Queue
  12   |  Encryption      |      3        |  Work Queue
  7    |  Encryption      |      1        |  Work Queue
  9    |  Encryption      |      1        |  Wait Queue
  11   |  Encryption      |      1        |  Wait Queue
  8    |  Extraction      |      1        |  Wait Queue
```

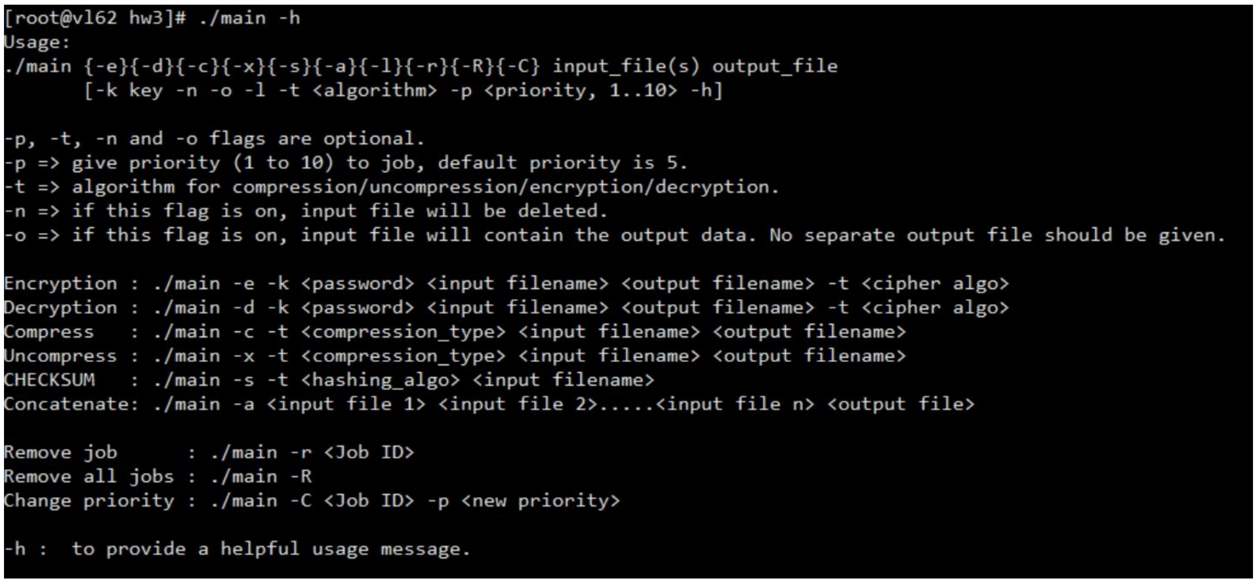
h. **Change Priority [-C]:** This option is used to change priority of any submitted, yet to be executed by the consumers. User need to provide the ID of the job whose priority needs to be changed.

i. **Remove Job [-r]:** This option is used to remove job which was submitted before, but yet to be executed by the consumers. User need to provide the ID of the job which needs to be removed. Before removing the job, kernel sends back a “removed job <job id>” message to the process which is waiting for some message about the completion of the requested job. This message is displayed to the user.

j. **Remove All Jobs [-R]:** If users want to remove all jobs, which are yet to be pending to be executed in queue. They can use this option; it will remove all the jobs.

Key [-k] is required only for encryption/decryption operations. The length of the key should be more than 6 characters. If key is given with other operation the user program will return an appropriate error.

Below is the screenshot of help of this project. Everything is well explained in help.



```
[root@vl62 hw3]# ./main -h
Usage:
./main {-e}{-d}{-c}{-x}{-s}{-a}{-l}{-r}{-R}{-C} input_file(s) output_file
       [-k key -n -o -l -t <algorithm> -p <priority, 1..10> -h]

-p, -t, -n and -o flags are optional.
-p => give priority (1 to 10) to job, default priority is 5.
-t => algorithm for compression/uncompression/encryption/decryption.
-n => if this flag is on, input file will be deleted.
-o => if this flag is on, input file will contain the output data. No separate output file should be given.

Encryption : ./main -e -k <password> <input filename> <output filename> -t <cipher algo>
Decryption : ./main -d -k <password> <input filename> <output filename> -t <cipher algo>
Compress   : ./main -c -t <compression_type> <input filename> <output filename>
Uncompress : ./main -x -t <compression_type> <input filename> <output filename>
CHECKSUM   : ./main -s -t <hashing_algo> <input filename>
Concatenate: ./main -a <input file 1> <input file 2>.....<input file n> <output file>

Remove job      : ./main -r <Job ID>
Remove all jobs : ./main -R
Change priority : ./main -C <Job ID> -p <new priority>

-h :  to provide a helpful usage message.
```

Following are the options supported in the system call:

a. **Priority [-p]:** User can optionally give priority (1 to 10) while submitting jobs, "10" being the *highest priority* and "1" being the *lowest priority*. If user does not give any priory, then default priority of the job is DEFAULT PRIORITY, which will be 5. If the same priority is given for two concurrently executing jobs, then the one that was submitted earlier gets executed first in FCFS order.

b. **Type [-t]:** This flag is used to take input from user about which type of algorithms to be used to file operations like encryption/decryption, compression/extraction, and computing checksum. The default algorithm for encryption/decryption is AES, for compression/extraction is DEFLATE, and for checksum it is MD5.

c. **Rename [-n]:** If this option flag is set while submitting job, then it will delete the input file and rename the encrypted file with output file.

d. **Overwrite [-o]:** If this option flag is set while submitting job, then it will overwrite the input file with the encrypted file. Only one file need to be given with this option. If both infile and outfile are given with this option, it will return an appropriate error in the user level.

**Few Examples:**

a. # ./main –e infile outfile –k “key”
   This command will encrypt infile with key using AES as encryption algorithm and set the priority of this job to Default Priority.

b. # ./main –e infile outfile –k “key” –t “blowfish” –p 8
   This command will encrypt infile with key using BLOWFISH as encryption algorithm and set the priority of this job to 8.

c. # ./main –c infile outfile –t “deflate” –p 2
   This command will compress infile using “deflate” as compression algorithm and set the priority of this job to 2

d. # ./main –s infile –t “sha1”
   This command will calculate the checksum of the infile using “sha1” as hashing algorithm and set the priority of this job to Default Priority.

e. # ./main -l
   This command will print all the jobs in the queue.

f. # ./main –r 4
   This command will remove job with job id 4 from the queue.

g. # ./main –c infile –t “deflate” -o
   This command will compress infile using “deflate” as compression algorithm and set the priority of this job to Default Priority. It will overwrite the infile with compressed file.

## 4. DESIGN

**User Level**:

For operations which are to be done asynchronously (all except List/RemoveJob/ RemoveAllJobs/ChangePriority) we are creating a child process which creates a netlink socket and listen for message to be sent from kernel about the status of the job sometime later.

Kernel threads have their CWD set to root, that is why if the user gives relative file names are given by user, we add the CWD to the filename, is the user has provided the absolute filename, so that kernel threads access the correct file.

Kernel thread have their CWD set to root, that is why if the user gives relative file names are given by user, we add the CWD to the filename, so that kernel threads access the correct file.

**Kernel level:**

In kernel, we are running NUMBER_OF_CONSUMERS (3) number of threads, and maintaining two queues, one is main_queue, and other is wait_queue.

There are two kind of operations:

1. List/RemoveJob/RemoveAllJobs/ChangePriority :

When the system call receives request for these jobs, they are served right away. Appropriate message is sent back to the user process. In case of List operation message is written into the job->key field which the user process reads and display to the user.

2. Encryption/Decryption/Compression/uncompression/Checksum/AppendFiles :

System call first tries to put the job into the main_queue, and if it is full, it puts it into the wait_queue. A warning message is sent to user to slow down speed, if main_queue is full. Size of each queue is MAX_LEN (5). If even the wait_queue is full, user process is throttled and put to sleep until some job gets completed and some place is available in the wait_queue. A job ID is sent back to the user for every accepted job. User need this job ID for operations like list, remove, change priority.

Consumers start on loading the module. They went to sleep state until some new job comes in the queues. We lock the queues before removing/adding a job in the queue, to avoid inconsistency. The lock used is a mutex lock. Consumers call respective functions defined in utils.h to operate the jobs.

When the operation completes, consumer sends a netlink success/failure message to the user mode. In case of failure, user also receives the error code which is translated to the error message. Consumer then fetches a new job from the queue. If it finds no job in the queue, it goes to the sleep state again.

On unloading the module, we set a flag which let all consumers skip the job fetching task and they all exit.

## 5. Flow in the Kernel Mode :

**Producer**:

System call receives a job struct from the user program. It checks for the validity of the parameters, eg, if the job has correct number of filenames, if the struct location is readable or not. We are using kernel function ‘getname’ for this. It then calls the respective function (remove, remove all, change Priority, encryption, decryption, concatenation, checksum, compression, uncompression) according the job type.

a. **List:** It is defined in sys_submitjob.h. It acquires a mutex lock and fetches all the entries from main_queue and wait_queue. It stores this information into the job->key field which user program can read and display to the user.

b. **Change Priority:** It is defined in sys_submitjob.h. It acquires a mutex lock and search for the given job in main_queue and wait_queue, if found it removes the job from the queue, changes its priority to the new one and add it back to the queue using add_to_queue() function. This function puts the job into the correct queue on the basis of its new priority.

c. **Remove Job:** It is defined in sys_submitjob.h. It acquires a mutex lock and search for the given job in main_queue and wait_queue, if found it removes the job and send a netlink message to the process which requested this job.

d. **Remove All Jobs:** It acquires a mutex lock and free all the entries from both the queues. If the use process has requested any other operations, it copies the job data to its internal struct with same time, and add it to a main qork queue. If main_queue is full, it adds it to the wait_queue, and sends back to the user a warning code. This is done by setting the job's priority to 0. User program knows that if the priority is set to 0, it means the job it requested for could not found a place in the main_queue, and it displays a warning message "to request it to slow down its speed" to the user. If even the wait_queue is full, it throttles the user process using "wait_event_interruptible()" and user process hang there till some job gets finished and this job find a place in the wait_queue. As soon as the job finds a place in any of the queue system call (producer) wakes up all the consumers to read this job from the queue. The user level process now reads the job_id which system call has stored in the struct jon and displays it to the user.

**Consumer:**

As consumers wake up, they try acquiring the mutex and removing a job form the main_queue to operate on it. It then releases the mutex. Jobs are shifted within/among the queues when some job is removed for processing. Consumer then calls the respective functions according to the requested job:

a. **Encryption/Decryption:** It is defined in utils.h. Job parameters are passed to the function and encryption/decryption is done. Default algorithm is AES; it is used is user has not provided any algorithm; another supported algorithm is Blowfish. Hashed user password is encrypted and added to the preamble of the file, so that during decryption we can verify if the password is same to what was given during encryption. We also use Initialization Vector to get stronger encryption.

b. **Compression/Extraction:** It is defined in utils.h. Job parameters are passed to the function and compression/extraction is done. Algorithm used is “deflate”. During compression we prefix each of the output buffer with the number of bytes that buffer has been compressed into; this is needed during the extraction. In this way extraction function will know how many bytes it has to uncompress into the buffer of PAGE_SIZE.

c. **Checksum:** It is defined in utils.h. Job parameters are passed to the function and Checksum is done. Default algorithm is MD5; it is used is user has not provided any algorithm; another supported algorithm is SHA1. Calculated checksum is send back to the user process which requested the checksum using netlink message.

d. **Concatenate:** It is defined in utils.h. Job parameters are passed to the function and concatenation is done. It supports a maximum of ten input files.

   Success/Failure messages for all of the above operations are sent back to the requesting user process using netlink message. Message is unicasted to the requesting process’s PID which we are storing in the job struct itself.

   After performing the operation it will again try to fetch some job from the queue. If consumers do not find a job in the queue, they go into the sleep mode, until producer add some job to the queue.

## 6. EXTRA CREDIT

This assignment has the following features added:

a. Prioritizing jobs – Every job can be added with a priority. If no priority is given, then the default priority of the job is MEDIUM PRIORITY.
b. Wait queue for jobs besides a work queue. It will warn the user when the work queue is full to slow down.
c. Calculating checksum using MD5 and SHA1
d. Performing Compression/Decompression using DEFLATE
e. Performing Concatenation of input files into output file

## 7. FILES:

a. common.h
b. main.c
c. main.h
d. sys_submitjob.c
e. sys_submitjob.h
f. utils.h
g. Makefile
h. submitjob.sh

i. README
j. HW3
k. kernel.config
l. Test scripts and files

## 8. CONTACT

This project has been developed by Mohit Goyal and Arjun Bora as a course homework and can be reached at mohit.goyal@stonybrook.edu and arjun.bora@stonybrook.edu respectively for feedback and suggestions.

## 9. REFERENCES:

http://lxr.free-electrons.com/source/include/linux/list.h

http://stackoverflow.com/questions/9907160/how-to-convert-enum-names-to-string-in-c

http://stackoverflow.com/questions/3299386/how-to-use-netlink-socket-to-communicate-with-a-kernel-module