“Asynchronous Job Queue and CallBack Mechanism”

Software Design Document

Team Name: hw3-cse506g14

Team Members: Aakriti Mittal
Shubhi Rani
Tanya Tiwari

Date: 12/03/2015

Table of Contents

1. Introduction

1.1 Purpose

This software design document identifies the design and architecture of Asynchronous Job Queue and Callback mechanism which has been implemented as a Linux kernel module in LINUX 4.0.9+ of CentOS Operating System.

1.2 Scope

The project is about asynchronous and concurrent processing of producer-consumer problem. It involves efficient kernel locking using mutex and callback using netlink sockets. The code is portable to any other machine.

1.3 Overview

The document further describes the detailed system overview which explains project`s functionality at a broad level and detailed design architecture of the project module both at user and kernel level.

2. System Overview

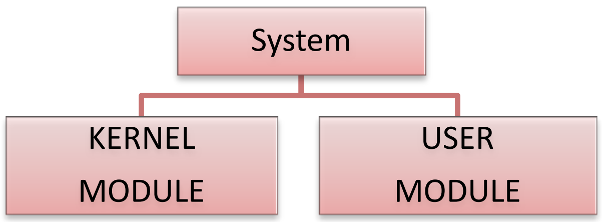
This project aims to simulate the asynchronous job queue mechanism that takes place for executing various jobs. Callback mechanism using NetLink socket has been added to display the status of various jobs back to the user.
The module supports different types of jobs: Encryption, Decryption, Concatenation, Compression, Decompression, calculating Checksum. Various jobs are added to the queue for execution. Two types of Queues have been implemented: Main Queue and Wait Queue. Queues are maintained by the producer; consumer thread consumes jobs based on their priority value. Extra Functionality for the user has also been supported:

1. Listing Jobs at User Level - If a user wants to see which jobs are getting executed.
2. Removing a job from the main Queue- If a user doesn`t want to continue with the processing of a particular job.
3. Changing Priority Value of a particular job- If a user wants to change the priority of any job submitted by him.
4. Removing all jobs from the queue- If a user does not wish to execute jobs further.

The jobs are coming from the user level and passed to system call for execution.
Since some operations can take really long to finish, the system call is designed in such a way that the user program can exit while concurrent operations were being performed in the kernel. In this scenario, the user has the option to submit more jobs to the system call while other operations are underway. The system call keeps adding the newly submitted jobs in its queue to perform when the existing jobs have finished execution.

3. System Architecture

The system consists of kernel module code and code at User level which contains varios files and functions as follows:



3.1 Kernel Module

3.1.1 Modular Design

sys_ submitjob.c

- Sys_submitjob module is initialized. A consumer thread is spawned and a socket for callback is initialized.
- A job structure is populated and passesd to the producer.
- During rmmod, the consumer thread is killed and socket is released.

producer.h

- Holds LOCK
- If shared queue length is less than MAX count, Job is added to shared queue .
- If shared queue length is full and wait queue length is less than MAX count, Job is added to wait queue
- Else Producer is THROTTLED and jobs are discarded.
- Wakes up Consumer
- Releses LOCK

consumer.h

- Holds LOCK on Shared queue
- Consumer removes one job from shared queue and process it further for execution.
- Decreases shared queue length.
- If jobs are pending in wait queue, jobs are removed from wait queue and added to main queue.
- Releases LOCK
- Wakes up producer.

callback.h

- Initialises a struct socket.
- Receives callback for a particular pid and message.
- Unicasts the message received from a job to the user.

utilities.h

- Code for all the utilities :- Concatenation, checksum, encryption, decryption, compression, decompression is present.
- code for handling erroneous condition while file open/read/write.

jobqueue.h, submitjob.h

- Submitjob.h: Job structure and other directives.
- jobqueue.h: Initialize queue,Add job to the queue, remove job from the queue on the basis of priority, change priority of a job.
- functions to support user calls: remove a job from the queue, remove all jobs from the queue, display the jobs at user level.

3.1.2 Functionality

1.) Data Structures:

The job structure includes all the fields in which the required values for the execution of a utility is needed:

```
struct job{
	char *infile;
	char *outfile;
	int job_id;
	int job_type;
	int priority;
	char *algo;
	unsigned char *key;
	pid_t pid;
};
```

Enum data structure defining values for all the utilites:

```
enum job_type {
	ENCRYPT,
	DECRYPT,
	CHECKSUM,
	CONCAT,
	COMPRESS,
	DECOMPRESS,
	REMOVE,
	REMOVEALL,
	DISPLAY,
	CHANGEPR
};
```

Structure needed for the job queues:

```
struct job_queue {
	struct job *job;
	struct list_head head;
	struct mutex mq;
};
```

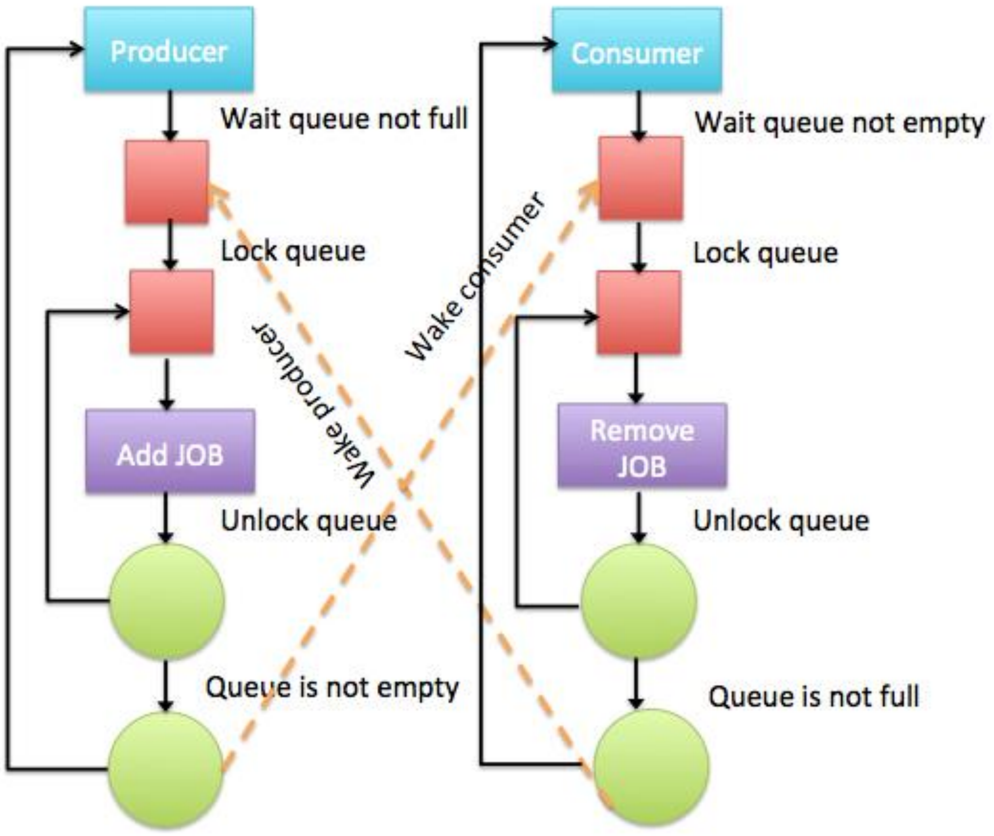
The main functionality of the module can be divided into following parts as follows:

1.) **Queue Operations:**

Two queues are maintained. One is the main queue that is shared between the producer and consumer and other is the wait queue, where jobs are added if the main queue is full.
Mutex locks are held while executing any operation of the queue, i.e while adding a job to the queue, while removing a job from the queue etc.
The mutex locks are released after the completion of the queue operation.
The execution of any job from the main queue is done in the order of priority (1,2, 3) of the job. 1 considered as the highest and 3 as the least.

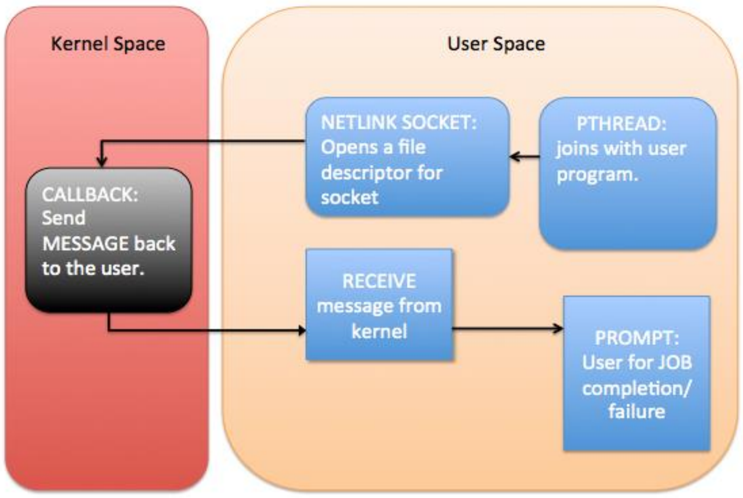
2.) Producer - Consumer:

The producer consumer mechanism can be explained with the help of diagram as follows:



3.) Callback:

Netlink socket mechanism has been used for sending information back to the user.
A pthread has been created at user level which listens to message from the callback function. This makes sure that the user does not wait for the user process and can execute other tasks until kernel execution gets completed.



Kernel Space
User Space
CALLBACK: Send MESSAGE back to the user.
NETLINK SOCKET: Opens a file descriptor for socket
PTHREAD: joins with user program.
RECEIVE message from kernel
PROMPT: User for JOB completion/ failure

4.) Additional Operations:

There are additional operations that are supported by the kernel module. These are for the convenience of the user so that he gets the idea of execution of jobs at the kernel level. The operations are as follows:

1.) Listing Jobs at User Level:- If user gives –l option, the list of jobs (job_id,job_type, infile) presently in the main queue will get printed at the user level.
2.) Removing a job from the main Queue: In order to remove a job from main queue, the user gives –r option along with the job id of the job. The job would be removed from the main queue and the rest of the jobs would be executed.
3.) Removing all jobs: When –R option is given, all the jobs are removed from the main queue and the queue length becomes zero.
4.) Change Priority: The priority of the existing job can be changes using –C option along with the job_id and the new priority.

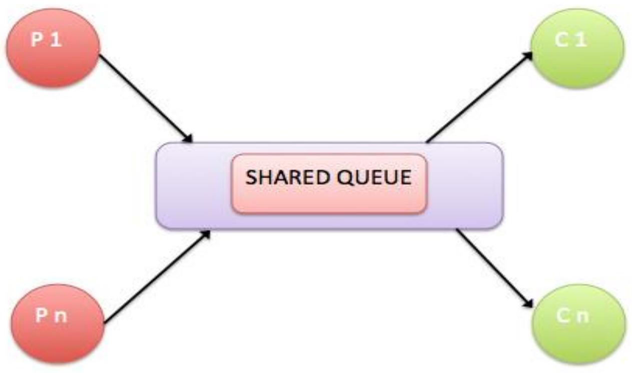
5.) Utilities:

The following utilities are supported by the module:

1.) Checksum: Computing checksum computing two algorithms: SHA1, MD5. User has to provide the input file.
2.) Encryption/Decryption: User has to provide the key for encryption, error conditions are handled in case of erroneous key inputs. The key is hashed and appended along with the encrypted file. For decryption, the correct key has to be given. Callbacks to user are provided for success and failure cases.
3.) Compress/Decompress: Compression and decompression of files can be done using this option.
4.) Concatenation: Concatenation of n files can be done using this utility.
   In case of Encryption/Decryption, Compress/Decompress and concatenations, a temporary file is being created which is deleted in case of any failure and renamed in case of success.

6.) Extensibility:

Our project can be extended to include multiple consumer threads.



3.2 User Module:

The user module takes input from the user using getopt(). The User has to supply the input option for various job types as follows:

./submitjob [operation] [flag] [priority]infile outfile
Where operation can be:
-e: Encrypt the file
-d: Decrypt the file.
-h: Perform checksum.
-c: Concat files
-z: Perform Compression
-u: Perform Decompression
-r: Remove job from Queue
-R: Remove all jobs
-l: List the jobs in the queue
-C: Change the priority of the job.

Priority value can be 1,2,3.
The flag is an optional optional parameter which is as follows:
-a : Algorithm for checksum
-k : Key for encryption and decryption

Outfile is also an optional parameter which is passed in case of Encrption, Decryption, Concatenation, Compression, Decompression.
Examples:

```
/usr/src/hw3-cse506g14/hw3/submitjob -l
/usr/src/hw3-cse506g14/hw3/submitjob -z 1 /usr/src/hw3-cse506g14/hw3/test/data_large
                                          /usr/src/hw3-cse506g14/hw3/test/compress_output_large
/usr/src/hw3-cse506g14/hw3/submitjob -r 25
/usr/src/hw3-cse506g14/hw3/submitjob -u 1
                           /usr/src/hw3cse506g14/hw3/test/compress_output_large
                         /usr/src/hw3-cse506g14/hw3/test/decompress_output_large
/usr/src/hw3-cse506g14/hw3/submitjob -l
/usr/src/hw3-cse506g14/hw3/submitjob -C 41 3
```

5.) References

http://www.linuxjournal.com/article/7356?page=0,3
http://lxr.free-electrons.com/