CSE 506 Operating Systems

Homework 3

Asynchronous and Concurrent Processing with Callback Functionality

By

Name	SBU ID
Jasmit Kaur Saluja	110463904
Harshkumar Patel	110371010
Nirmit Desai	110351469
Shubhada Patil	110284246

Table of Contents

1. About This Document

The purpose of this assignment is to learn about asynchronous and concurrent processing with callback functionality inside linux kernel. For this assignment we have implemented code in vanilla linux kernel 4.0.9. This document briefly describe the various approaches used along with their purpose. It also describes different data structures, locking mechanism, callback functions etc. used for this assignment.

2. Introduction

Concurrency is at the heart of kernel's fast processing nature. Allowing processes to do their work asynchronously will boost up this concurrency and processing speed. Most of the times, tasks given by users are time consuming and calling process should wait till the execution of that lengthy processes. To avoid such scenario, we've implemented functionality that will do these processing tasks asynchronously.

Here, user can submit "job" (specific task to be performed) for processing and continue his work till he wants. Users are not required to wait until that job is finished. Notifications regarding the status of that job will be passed to user using callback functionality. User can execute jobs till he wants and then give "exit" to indicate end of user input. After submission of each job, user will be prompted with the message containing different operation options available to him. After choosing an option, user will be prompted with further options for selected operation.

To allow submission of such jobs, we've added new system call which will accept appropriate arguments from user. We have also implemented user code that will make a call to this system call and prints appropriate callback results.

3. Functionalities

List of functionalities implemented are as below.

3.1 Encryption

Encryption of any file is required to make it secure. User can select option for "Encryption" and then he can give filename for encryption along with key to encrypt it. For encryption, we're using Advance Encryption Standard **(AES)** algorithm. Output file will be generated with ".enc" extension. i.e. for encryption of file foo, generated encrypted file will have name "foo.enc".

3.2 Decryption

In order to decrypt previously encrypted file, user can select option for "Decryption" and then he can give filename for decryption and key to decrypt it. Here, output file will be generated with ".dec" extension. i.e. for decryption of file foo, generated decrypted file for input file "foo" will have name "foo.dec". Here, user should give the same key to decrypt as it was used for encrypting it otherwise error will be given back to user.

3.3 Concatenation

One can concat one or more files into one output file. User can select option for “Concat” and then he can give as many files as he wants for concatenation. Here number of files should be at least two and last filename will be taken as output file. Format for giving files for concatenation is as below.

Infile1 [infile2] [infile3] …… outfile

In case of only one input file, this option will works as copy command. Output file will be created if it doesn’t exists or if it is already there then it will be overwritten on successful concat operation. Here, all input files should have read permission and output file (if exists) should have write permission.

3.4 Checksum

Checksum for any file can be calculated using different types of hash functions. User can select option for “Checksum” and then can give filename and hash function to calculate checksum. File should be smaller than page size. Following hash functions are supported.

- MD5
- SHA1
- SHA512

3.5 Compression

Compression is a useful technique when we want to store/transfer larger files especially over the network. User can select option for “Compression” and then can give filename and compression algorithm to compress the file. Below are the types of compression algorithm supported:

- Deflate
- lz4
- lz4hc

Output file will be created with same name as input file with “.x” extension where ‘x’ is type of algorithm used for compression. If user tries for compression using any options other than above four then error will be returned. File should be smaller than page size.

3.6 Decompression

User can decompress files by selecting option for “Decompression” and then he can give filename to decompress. Output file will be created with same name as input file with ‘.dcmp’ extension. i.e. if “foo.deflate” is given for decompression using Deflate algorithm then output file will be “foo.deflate.dcmp”. If user try for decompression of a file with different algorithm than what it is used for compression or try to decompress uncompressed file then it will give error.

3.7 List Jobs

User can list all queued jobs by selecting option for “List Pending Jobs”. Jobs will be shown with their job id, job type and job priority. At the time of submitting job, user needs to provide job priority. If no priority is specified then it will be 1 (default) for that job.

3.8 Change Priority

User can change the priority of queued jobs by selecting option for “Change Priority” and then he can give job id (for which he wants to change priority) and new priority. Priority can range from [1-256] and follows- higher the number, higher the priority.

3.9 Remove Job by Job ID

User can also remove any particular pending job from the Job Queue by selecting option for “Remove job by id”. User can list pending jobs first and then can look for job ID to remove that particular job. Appropriate error message will be returned in callback if there is no job pending with this ID or if job ID is invalid. Also the thread which requested to submit the job initially will be notified that the job has been removed from the queue (by signaling waiting user threads).

3.10 Remove all Jobs

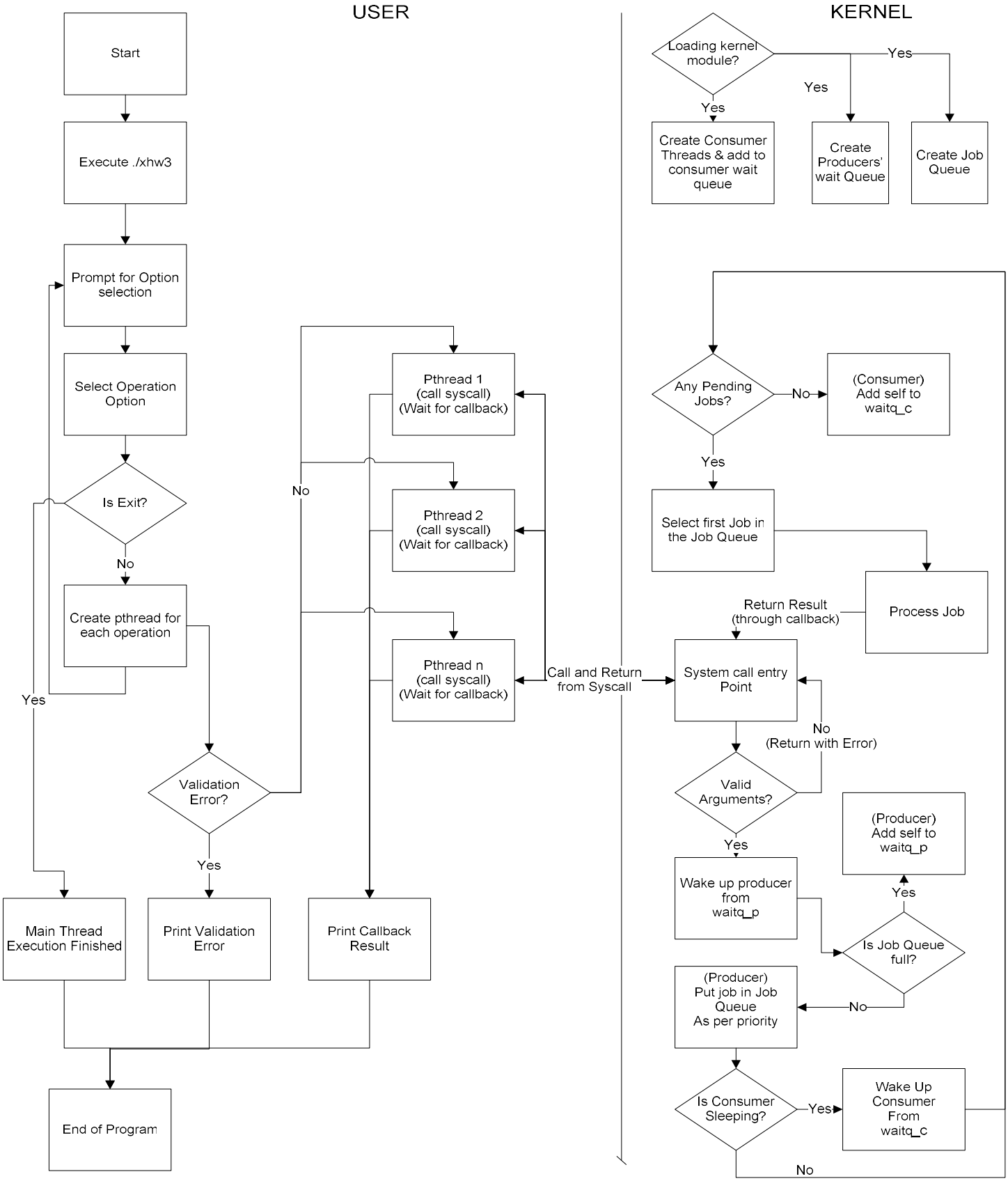
Alternatively, user can also remove all the jobs currently queued by selecting option for “Remove all Jobs”. Appropriate error/success message will be returned through callback if there are no pending jobs. Also threads which requested to submit the jobs initially will be notified that the corresponding jobs have been removed from the queue (by signaling waiting user threads).

(**Note:** Operations- List Jobs, Change Priority, Remove Job by Job ID and Remove all Jobs- require immediate response from kernel. In order to enforce this, we are not executing these operations via consumer threads.)

3.11 Exit

User can terminate execution of program by selecting option “Exit”. Callback functionality will print appropriate status for all previously submitted jobs.

4. Flow of Execution



5. Description

This section will describe about different types of modules and other policies used for the purpose of this assignment.

5.1 Creation of Producer-Consumer and initialization Queues

Whenever kernel module 'sys_submitjob.ko' is installed, it will create three (can be increased) kernel threads (consumer threads) and a job queue. This job queue is basically a link list in which jobs will be added (based on the priority of jobs) and removed. Two separate queues- Producer wait queue and Consumer wait queue- will also be created. Other than threads and queues, Netlink socket will also be created which will be used for callback functionality. On removing module, mutex will be destroyed, Netlink socket will be released, all the queued jobs (if any) will be removed and both producer-consumer wait queues will also be released.

5.2 Calling system call and argument validation

At user level, for each job, one dedicated pthread will be created for each operation (if job data is valid). This pthread will make a call to system call with necessary data required for that operation. Here for different types of operation, data will be stored in appropriate structure defined in the common user header file ('xhw3.h'). Arguments will be passed as 'void *'. At kernel level, before adding job to Job Queue, arguments are validated based on operation and related structure. If arguments are not valid then in that case system call will not add that job in the queue and immediately returns to user. Appropriate error message will be printed on user side. Argument validation is also done on user side before making a call to system call.

5.3 Job Queue Operations

Producer and Consumer will add and remove jobs from the Job Queue respectively. Mutex lock is required to perform these operations as multiple producer/consumer can update this queue at the same time. This lock is also required at the time of changing priority of any job or listing and removing jobs from queue. Jobs are added to the queue by producer based on job priority. Consumer will always take first job from the job queue so every time job with higher priority will get executed.

5.4 Working of Producer and Consumer

Consumer threads are created at the time of module loading. These threads will go to wait queue if there are no jobs pending. Two additional queues named producer wait list and consumer wait list are also created to keep track of waiting producer and consumer. If Job Queue is currently full then producer will go to producer wait queue and remains there until job queue is full. Maximum number of jobs that can reside in job queue is defined as MAX_JOBS and maximum number of consumer threads is defined as MAX_CONSUMER_THREADS in header file (common_utility.h).

5.5 Callback Functionality

With asynchronous operations based on producer/consumer it is require to provide callback mechanism so that user can be notified for outcome of previously submitted jobs. Here we have used **Net link** based user-kernel communication. At the time of module loading, netlink socket will be created for communicating appropriate data/job status back to the user. The data transmitted through this socket depends on the type of operation requested i.e. for checksum operation this reply message will contain calculated checksum for the given file and for other operations that don't require data to be sent back to user it will contain only information for status (success/failure) of that operation. At user level, each pthread will establish new socket, bind it to the same port where kernel has created socket and after executing system call it will wait in blocking mode ('recvmsg'). After consumer finishes its work, it unicasts status of job (or checksum in case of checksum operation) to respective process. As soon as pthread receives message from kernel it will finish its execution and will exit.

5.6 Locking Mechanism

Locking mechanism becomes necessary when it comes to concurrency. Whenever two or more processes want to operate on a same source they require some locking mechanism to operate as per expectation. Here we have Job Queue which is shared between consumers/producers and to do operations on such data structure we require locking. So we have used a mutex lock for this job queue which will make sure that operations on job queue such as adding job to queue, removing job, changing priority and listing job items etc. are done without any interference between calling threads.

5.7 User Code

File "xhw3.c" contains user code which is needed to be executed to perform all operations such as encryption, decryption, concatenation, checksum calculation, compression and decompression of a file. User code is designed in such a way that user can submit jobs without waiting for execution of previously submitted jobs. User will be prompted with several options and after he makes choices, appropriate message will be shown for further input. Once user submits request with valid deta, one process thread (pthread) will be created for that job and this process thread will call system call. After making system call, this pthread will wait for callback message from kernel. Simultaneously, main thread can continue its execution and submit further requests. Main thread can be terminated by selecting "exit" or pressing "Ctrl + d". Program execution will end after all created threads finish their execution. Main thread is not required to wait for other threads to finish resulting in saving CPU cycles. Appropriate job status will be printed on user side. Callback messages will interfere with the execution of main thread. User can see outcome of previously submitted job while submitting new jobs as requests are processed aynchronously.

6. Files and their Usage

Below is the list of files used for this assignment along with their purpose/usage:

File Name	Usage
xhw3.h	This file contains task specific structures and a job structure which encapsulate this task specific structure. It also contains declaration of different type of operations to be performed.
xhw3_k.h	This file includes xhw3.h and contains signature of different task specific methods which are defined in operation related .c files.
common_utility.h	This file contains declaration of structure that will be used in link list implementation of queue. It also specifies structure called job item which will contain job related info. This file contains implementation of methods related to job queue.
common_func.c	This file is created to make code more modular. It contains common methods that is needed by multiple operational related files such as read from file, write to file etc.
xhw3.c	This file contains user code which is responsible for taking operation type and related input from user. This file is responsible for making a call to system call. It creates pthread for each operation and wait for callback functions.
submitjob.c	This file contains code that will be executed whenever a system call is invoked. It takes data from user and validate it. It also contains method that will be called by consumer thread and it is responsible for processing job. Actions on module loading and unloading are also defined in this file.
encryption.c	This file contains implementation of methods related to encryption and decryption. It also perform validation for files related to these operations.
concat.c	This file contains implementation of methods related to concatenation. It also perform validation for given input files, output files related to this operation.
checksum.c	This file contains three methods that computes checksum for given file. These methods are for computing checksum using SHA512, SHA1 and MD5.

compression.c	This file contains implementation of methods related to compression and decompression. It also contains one method which call one of these methods with appropriate algorithm to use.
Makefile	This file contains command that will compile above operation related files as well as user code. It can also be used to clean files.
install.sh	It contains script that will install and remove module - 'job.ko'.
kernel.config	.config file used to build kernel
log.txt	This file contains log about different types of jobs who have finished their execution along with their start time, end time, time to run, job type, priority etc. details.

7. Important Features

- Keeping track of performed operations via a log file. A log file named "log.txt" is created under "hw3/" directory. Log file can be useful to evaluate performance of the system as one can see execution time for different operations and status of all the jobs. This is additional extra credit functionality and is not requested as part of assignment specifications.

- Use of pthreads: At user level, each task is submitted by different process thread so that each thread can process asynchronously on user side too. Main thread is not require to wait for completion of all created threads resulting in saving CPU cycles.

- Asynchronous Processing: Jobs are executed asynchronously via Producer-Consumer mechanism inside kernel. Proper locking mechanism is used to maintain this concurrency.

- Callback Functionality: To notify user thread which is in waiting state for getting result of particular job, callback functionality is used. Here user and kernel exchange information based on Netlink mechanism. Callback message will be unicasted to particular process through socket by kernel.

- Allowing compression using different types of algorithms and implementation is not restricted to single compression algorithm. Supports deflate, lz4 and lz4hc algorithms.

- Allowing calculations of checksum using three different algorithms – MD5, SHA1, and SHA512.

- File concatenation is not restricted any fix number of input files. User can give any number of input files to concat them into single file.

- Code Modularity: As this assignment includes implementation of various operations, it is a good practice to make different files for different operations so that other functionality can be included efficiently and coding/debugging can be done easily.

8. How to Run This Code

Below are the steps to run this code:

1. Clone our Git Repository (**hw3-cse506g13**).
2. Build kernel using **kernel.config** file located in /usr/src/hw3-cse506g13 as follows:
   2.1 make
   2.2 make modules
   2.3 make modules_install install
   2.4 reboot
3. Do **cd /usr/src/hw3-cse506g13/hw3**.

4. Run **make** command to generate loadable module and executable user code.

5. Do **sh install.sh** to remove previously loaded module (if any), insert new module and generate user code executable.

6. Run user code by command '**./xhw3'**.

7. Use **make clean** for clean up.

9. References

Net link between user and kernel:

http://www.linuxjournal.com/node/7356/print ,
http://stackoverflow.com/questions/15215865/netlink-sockets-in-c-using-the-3-x-linux-kernel

File compression in kernel:

http://lxr.free-electrons.com/source/crypto/testmgr.h

List data structure in kernel:

https://isis.poly.edu/kulesh/stuff/src/klist/