Asynchronous producer consumer Mechanism

(Operating Systems CSE506 Homework-3
3rd Dec 2015)

Design Document

Submitted by:
Atanu Ghosh, Bhargava Mourya, Dhanendra Jain
110280569,110308227,110369635

Introduction:

This assignment is about implmentation of a system call for asynchronous and concurrent processing of files at the kernel level, kernel locking, efficiency, and callbacks. A system call is written in this assignment which can be loaded into the kernel. The types of functions perfomed by this sytem call include encryption/decryption, checksum, compress/decompression and concatenation of files. Some or most of the operations mentioned above take a lot of time to complete. The system call is designed in such a way, that the user need not wait for processing to complete as the system call just enques the task onto the queue and returns. The user is later notified via callback mechanism about the status of the task.

Functionalities supported by the system call:

1) Encryption/Decryption of files
2) Checksum of a file
3) Compression/Decompression of files
4) Concatenation of 2 files
5) List all jobs
6) Remove queued jobs
7) Change priority of a queued job

Working:

The assigment is composed of 2 main parts:

- sys_submitjob.c: The system call to perform tasks at the kernel level.
- produce_job.c: The user code, where the user can submit jobs to the system call.

Steps to compile and load the sys call:

- Git checkout the master branch
- cd ~/hw3-cse506g05 and make the kernel
- cd ~/hw3-cse506g05/hw3/ and run make
- sh install_module.sh

Usage of the user program:
./produce_job operation [options] [infile_1] [infile_2] [outfile]

Where operation can be:

- -s: Compress 1 file
- -m: Decompress 1 file
- -c: Checksum of a file
- -t: Concat 2 files
- -e: Encrypt a file

- -d: Decrypt a file
- -r: Remove a queued job
- -l: List all queued jobs
- -N: Change priority of a queued job

Options can be:

- -p: Password for encryption / decryption
- -P: To specify Priority of this task. A value between [1, 3] with 3 meaning highest priority.
- -C: To be specified when the user requires callback.

And:
infile_2 needs to be specified only during concat.
infile_1, infile_2 and outfile should not be provided with –r, -l and –N options.

The user program performs validations on the input provided and errors out when an invalid input is give.

Examples of valid input:

- ./produce_job –e –C –p mypassword –P 2 infile outfile
  This call will encrypt infile and store it to outfile using password mypassword and callback is required for this. The priority of this job is 2 (medium).
- ./produce_job –c infile infile 2 outfile
  This call will concat infile with infile2 and store the result in outfile. It does not require a callback and default priority is 1.
- ./produce_job –s –P 3 infile outfile
  This call will compress infile and store the result in outfile. It does not require a callback and priority is 3.
- ./produce_job –N 3 2
  This call will change the priority of job with id = 2 (if it exists) and set it to 3.
- ./produce_job –r 3
  This call will change the priority of job with id = 3 (if it exists) and remove it from the queue.
- ./produce_job –l
  This lists all the jobs currently in the queue.

Design:

- INIT:
  As soon as the module is loaded, consumer threads are created and initialized, each of which may perform any of the jobs (1-4) mentioned above in functionalities. A main queue is initialized to hold all the tasks to be processed. The kernel list is used as a queue. A mutex lock is also initialized. Currently 3 consumer threads are created but it is configurable. Also netlink socket is created for sending data to user space (callback mechanism)

- PASSING OF PARAMETERS:
  If a valid combination of paramters are supplied, they are sent to the kernel in a void * structure object as defined below.
  ```
  struct job {
      pid_t pid;
      char *infile;
      char *infile2;
      char *outfile;
      unsigned char *keybuf;
      int job_type;
      int job_id;
      int priority;
      int new_priority;
      void *extra; /* Extensibility */
  };
  ```
  pid: the process id, generated by getpid() function.
  infile: the input file to process
  infile2: the second input file to concat
  outfile: the output file
  keybuf: SHA1 hash of the password supplied by the user
  job_type: the code for the job type
  job_id: Populated in the kernel. In case of change priority, it is sent by the user
  priority: Priority of this job. Default is set to 1 when the user doesn't supply a priority. Max value = 3.
  new_priority: The new priority to be set during change priority
  extra: any extra arguments for extensabilty. Currently holds the pointer to the array to be populated duting list operations

- LOCKING:
  All operation on the queue and the queue length variable is protect by the mutex. This ensures that all operatiosn such as LIST, REMOVE and pop from list are in sync as seen by all the threads thus preventing race conditions.

- SYNC OPERATIONS:
  If the job type is LIST, REMOVE or CHANGE_PRIORITY then it is processed synchronously by the kernel. In case of LIST, all the jobs currently in the queue are populated into an array allocated and given to the kernel by the user. In case of REMOVE and CHANGE_PRIORITY the operation is performed and success/error is returned to the user.

- VALIDATIONS:
  The sys_submit system call receives this void pointer and makes a copy into the kernel using copy_from_user. Getname function is used to copy infile, outfile. Extensive validations are performed for the files such as its presence, whether it is regular and infile & outfile should not be same. The keybuf is again hashed using SHA1. If any of the above validations fail, the job is not enqueued and an appropriate error number is set and is returned to the user space.

- PRODUCER:
  If everything is successful a unique job_id is given to this and it is then enqueud (to the tail) if the length of the queue is less than the MAX allowed queue length. If the queue length is full the producer is made to sleep, using 'wait_event_interruptible', till atleast one job is consumed by the consumer and the consumer then wakes up the producer usinf 'wake_up_all'.

- CONSUMER:
  The consumer threads sleep till the producer wakes it up. The consumer thread just picks a job from the queue based on priority and processes it. (Priority 3 is highest, 2 medium and 1 lowest). After processing, the consumer thread yields to the scheduler using the 'schedule()' call. The consumer thread(s) goes into sleep state on the condition that queue length > 0, using the 'wait_event_interruptible' call.

- CALLBACK (NETLINK):
  The callback mechanism used is netlink. It is enabled if the user has set the –C option while making the syscall (default is disabled). Netlink sockets are created both at user space and kernel space using a common protocol. In user space, user spawns a separate thread and in that it waits for the callback message from the kernel. It's a blocking call, so user thread will block after calling the recvmsg() function until it receives the message from kernel.

- EXIT:
  During unload of the module, all the jobs from queue are removed. All the consumer threads are killed gracefully and netlink socket is released.

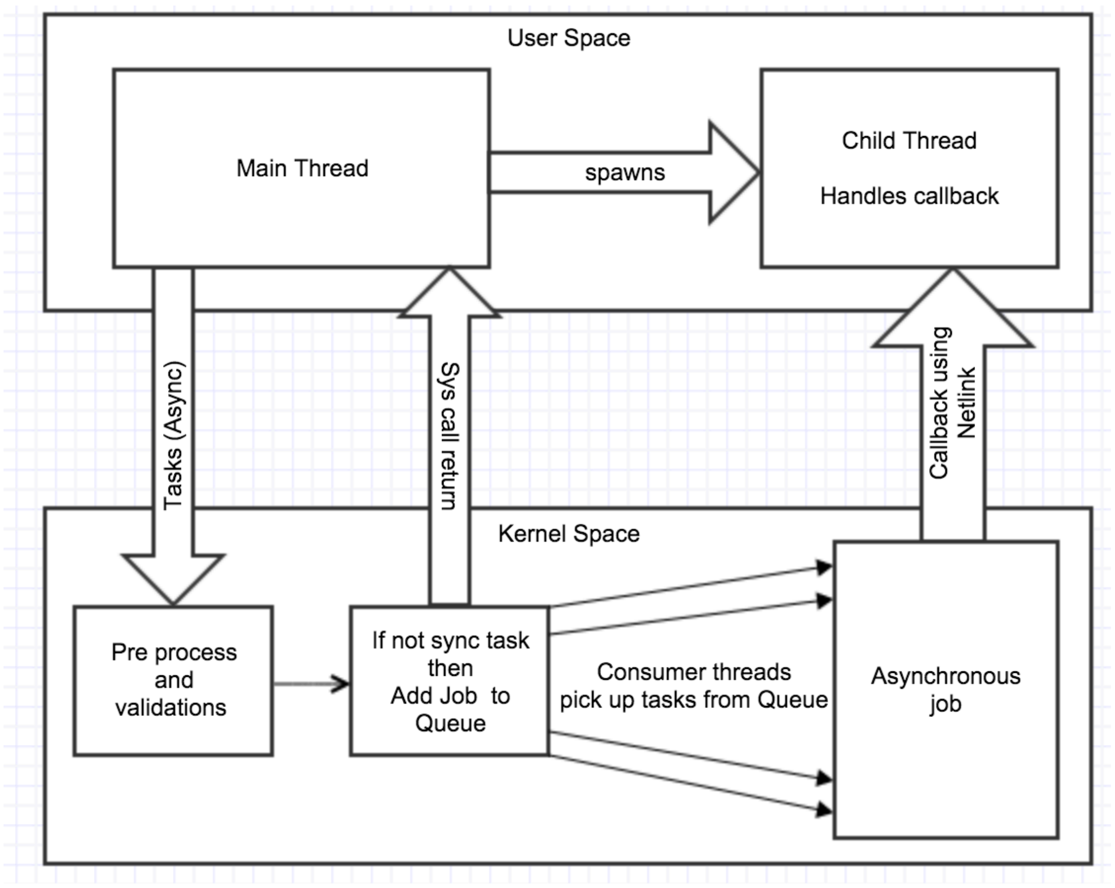

Figure to explain system design and flow

Working of the individual functionalities:

- Encrypt/Decrypt:
  - For encryption/decryption, used AES cipher with CTR mode.
  - AES cipher uses 16 byte cipher key, so truncated the cipher key obtained from userspace to 16 bytes and used that.
  - AES cipher encrypts/decrypts 16 bytes at a time. So if a data block is not a multiple of 16, CTR mode automatically handles it internally, no need of padding.
  - In encryption case, we put double hashed key in preamble of output file.
  - In decryption case, we access hashed key from preamble of encrypted input file and compare it with my double hashed key.
  - If key matches, only then we proceed for decryption, otherwise Key Mismatch error.
  - For computing hash in kernel, we have used CryptoAPI algo of SHA1

- Checksum:
  Code for checksum computation is present in computeChecksum function in checksum.c. It computes crc32 checksum of input file and writes to output file. Data is read page by page and checksum is computed for each page. Page data currently read and Checksum

computed for pages already read are passed as an argument to crc32() to compute new checksum. After file has been completely read, the final computed checksum is converted from hexadecimal to string and written to output file.

- Concat:
  For concatenation, we have read input files one by one. We read buffer from each input file in PAGE_SIZE and write it in output file, this continues till the end of input file. Note: Concat operations work for concatenation of 2 files only.

- Compression/Decompression:
  Compression of the input file is done using crypto_comp_compress api. Decompression is done using the crypto_comp_decompress api. The function signatures for both these operations are same and take in the input and output file names only. Note: Compression and decompression works for small files (< 2MB) as the entire file content is read at once and then compressed / decompressed.

- LIST:
  This function is to list all the jobs currently in the queue. The user sends an array of structs of type
  ```
  struct job_list {
        int job_id;
        int priority;
        int job_type;
  };
  ```
  of MAX size of queue into the kernel. The kernel populates the required data and the user code displays the list.

- REMOVE:
  Here, the user provides the job id to remove. If it is present in the queue then it is removed and success is returned. Else, -ENOENT is returned.

- CHANGE PRIORITY:
  Here, the user provides the job id to change and its new priority. If it is present in the queue then its job id is changed and success is returned. Else, -ENOENT is returned.

Extra Credit:

- Computing checksum using crc32.
- Compression/Decompression using crypto api.
- Callback (netlink) feature is made optional. The user can pass –C option to enable it.
- Priorities of jobs are maintained and they are removed from the queue based on priority. (1-lowest, 3-highest).

- User has the flexibility to change the priority of any job based on job id.
- Concatenation of 2 files.

Files Added / Modified:

All files added in hw3/ folder

- sys_submitjob.c : The system call producer consumer implementation.
- kernel.config : Config file of kernel
- concat.c : Code to concat 2 files
- compress_decompress.c : Compression and Decompression code
- install_module.sh : load or unload the system call
- produce_job.c : User code
- common.h : Common headers betwwen user and kernel
- encrypt_decrypt.c : Encryption Decryption algorithm
- nlink.c : User level program for netlink sockets and receive message
- test_script.sh : Script to fir queries to the system call
- checksum.c : Checksum implmentation
- design.pdf : Deisgn.pdf file
- Makefile : Build files and create libraries for the new system call
- sys_submitjob.h : Headers for system call
- ~/arch/x86/syscalls/syscall_32.tbl: Adding submitjob symbol for 32 bit.
- ~/include/linux/syscalls.h: System call entry added for sys_submitjob.

Requirements:

- This assignment was implemented and tested on linux kernel 4.0.9 on Cent OS.
- A multi core processor is required for robust performance

References:

[1] For using list data structure in list.h, used below links as reference
http://www.makelinux.net/ldd3/chp-11-sect-5
http://isis.poly.edu/kulesh/stuff/src/klist/

[2] For waitqueue, process waiting and waking up, referenced below links
http://www.makelinux.net/ldd3/chp-6-sect-2
http://tuxthink.blogspot.ca/2011/04/wait-queues.html

[3] For Netlink, code referenced from below link
http://stackoverflow.com/questions/3299386/how-to-use-netlink-socket-to-communicate-with-a-kernel-module
http://amsekharkernel.blogspot.com/2012/01/what-are-ways-of-communication-bw-user.html

[4] For Encrypt/Decrypt, referenced below links
http://lxr.free-electrons.com/source/drivers/staging/lustre/lustre/obdclass/capa.c#L292
http://lxr.free-electrons.com/source/drivers/staging/lustre/lustre/obdclass/capa.c#L345
http://www.chronox.de/crypto-API/ch06s02.html

[5] For computing hash in kernel during encryption/decryption
http://lxr.free-electrons.com/source/fs/ecryptfs/crypto.c#L87
http://stackoverflow.com/questions/3869028/how-to-use-cryptoapi-in-the-linux-kernel-2-6

[6] For CRC32 checksum:
http://www.linuxquestions.org/questions/linux-newbie-8/generate-crc-value-for-file-s-199640/

[7] For Compression/Decompression -
https://dev.openwrt.org/browser/trunk/target/linux/generic-2.6/files/crypto/ocf/cryptosoft.c?rev=13288

[8] Referenced - http://lxr.free-electrons.com/source/

[9] Producer-Consumer Mechanism - CSE506 Class notes