Operating Systems Project Report

Chetan Naik (109930376)
Gowtham Srinivasan (10989072)
Naman Mittal (109888028)

1 Design

1.1 Abstract

We designed a structure having the job metadata and a linked list (to add jobs in the queue) using the list library.

We made the jobs run asynchronously while the user operations such as job removal, changing priority of the jobs and listing of jobs run synchronously. We made this choice for user operations because we feel that if a user wants to perform any of the above mentioned operations, he/she would want it to be done immediately and no lazy approach should be used in it.

Note: Initial Design Choice

Initially we thought of using `work-queue` to consume jobs. The advantage of having a `work-queue` based design is that it'll run the number of threads equivalent to the number of CPUs. But the disadvantage is that we will not be able to change the priority of the job easily and even the listing of jobs would be cumbersome.

1.2 Structures

We have a structure named `job_metadata` which stores the job's metadata and it'll be used by the consumer to call the appropriate function (and then the data will be used by the called function).

```
struct job_metadata {
        /* Store type of job to be performed*/
        int type;
```

```
        /*
         * Job_ID will be generated by producer
         * and will be used to remove a job and
         * it'll also be listed to user when user
         * requests to list all jobs in the list
         */
        unsigned int jobid;
        int job_priority;
        int rename;
        int overwrite;
        int delete_f;
        int operation;
        /* Operation will contain code to
         * encrypt/decrypt or compress/decompress
         * 1 -> Encrypt/ Compress
         * 2 -> Decrypt/ Decompress ( Not implemented )
         * 3 -> Hashing
         */
        char *input_file;
        char *output_file;
        char *key;
        char *algorithm;

        int pid;
        /*
         * int priority;
         */
        /* this variable will store the number of files*/
        int no_of_files;
}((__attribute_packed__));
```

This structure is used by the userland module to fill the job information and send it to the kernel.

```
struct job_queue {
      /* Queue to hold jobs */
      struct list_head job_q;
      /* Job specific data */
      struct job_metadata job_d;
};
```

This structure is used in kernel module to copy the data sent by the user and add the node (job) to the list. We also have a structure which `extern`-s the function for creating the socket and listening to the message

```
extern int createSocket(int);
```

This function creates a socket for the communication between the kernel and the user space. The int passed to the function is pid. This is because the Linux Standard always generate a unique PID for every process so we have directly used this in creation of socket to have one-to-one connection.

```
extern void listen_to_kernel();
```

After creating a socket we created a `pthread` and passed this function so that the main user program can continue with its work and listen to the status report sent by the kernel.

- In our assignment we followed the approach in which a user just submits a job and opens a net-link socket for the kernel to return the status once the job is over.
- We used a wait-queue to make the consumer sleep in case there were no jobs pending.
- We used the list data structure to save about jobs and a wait-queue to make consumer thread sleep.
- Maximum payload size of the net-link socket is 1024 bytes.
- Maximum job queue size is 128.

1.3 Features

Our module supports the user to submit a job for encryption/decryption or for finding checksum, which will be executed and the output will be sent back through the net-link interface. This enables the user to submit work and let it be handled later (asynchronously). We also tried to support compression/decompression but we could not get it to work completely.

- For encryption/decryption we support `aes`, `blowfish` and `des` algorithms.
- For checksum we support `md5` and `sha1` algorithms.
- We have used the mutex locks to give an exclusive access to the linked list and a counter which maintains number of jobs in the queue.

- We also made sure that the job with highest priority is run every-time by finding the highest priority job in the queue and then extracting that job out of queue and calling the appropriate function to complete the job.

1.4 How to run?

- Run make in the hw3 directory to build the kernel and user modules.
- Insert the build module using `sudo insmod sys_submitjob.ko`
- Run the userland module using xsubmit (arguments)

```
Usage:
To Encrypt/Decrypt
./xsubmit -t 1 -e/-d -a "algo name" -k "key" inputfile outputfile

Optional flags:
-x to delete input file
-r to rename
-w to overwrite

Algorithms Supported
1.  aes
2.  blowfish
3.  des

To find checksum
./xsubmit -t 3 -a "algo name" inputfile
Algorithms Supported
1.  md5

To list jobs
./xsubmit -t 4

To remove a job
./xsubmit -t 5 -j jobid

To change priority
./xsubmit -t 6 -j jobid -p new priority
Priority supported between 1-10.  10 being highest.
```

1.5 References

For understanding and implementing netlink: stackoverflow.com link