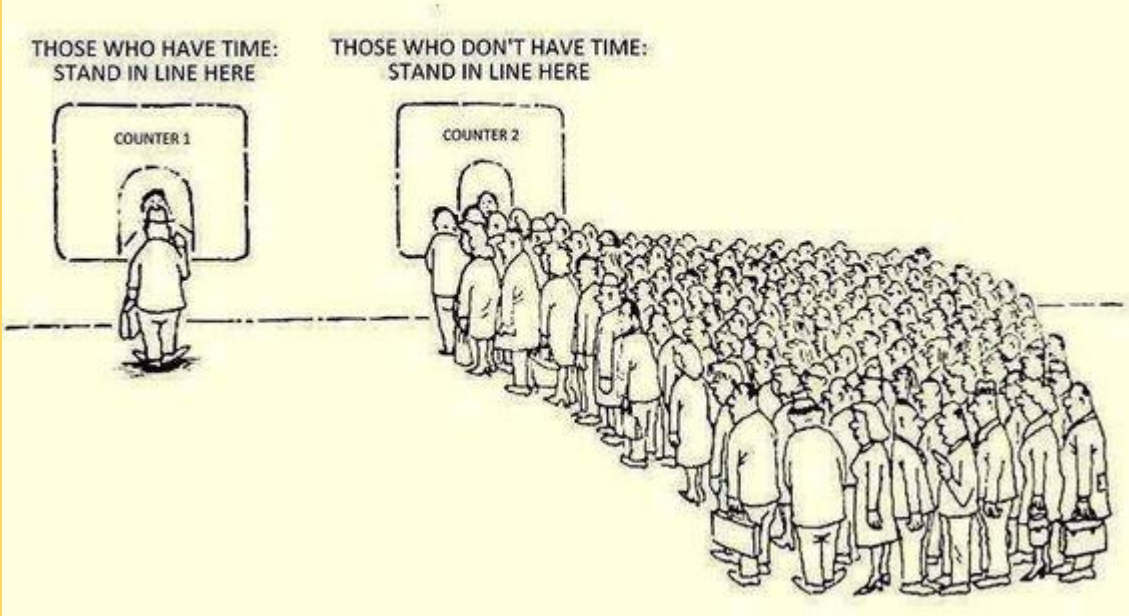

CS 506 Spring 2018
ASSIGNMENT 3

1. Lavanya Kandhibedala - 111493391
2. Vishwatej Reddy Anugu - 111446995
3. Praneeth Gubbala - 11132448
4. Balraj Inder Kaur Mahal - 111498697

Image refireference: https://i.pinimg.com/564x/1c/b7/f5/1cb7f5b188dcd5aebcab0dbbbc698c9f.jpg

Introduction

- This assignment implements a Linux kernel-based system to handle file processing operations like unlinking, encryption and/or compression synchronously in case of small files (4KB and smaller) and ASYNCHRONOUSLY for larger files using work queues.

- When files are unlinked, if the processor file is setup as such, then the file is moved to a special trashbin folder, and optionally compressed/encrypted. Support to undo/recover a lost file using an ioctl, periodic deletion of older files in the trashbin and to purge the trashbin folder completely is provided.

Approach

(i) SYNCHRONOUS PROCESSING :-

Order : Compression, encryption and unlinking

User sets the encryption key through a special ioctl that is stored in the /proc file. The key is stored in kernel memory so that it can be accessed by other kernel programs.

Then, the file is compressed (using crypto_comp_compress()) , and encrypted.

After encryption , the file is unlinked and a copy is moved to the trashbin folder.

Approach

(ii) ASYNCHRONOUS PROCESSING :-

The approach involves maintaining a work queue , spawning a predefined number of worker threads. The user program (producer) submits the job to the kernel which gets enqueued in the work queue. Before enqueueing the jobs, the size of queue is checked, if it is full then the producers are throttled.

The user process then returns. The worker threads remove the job based on first come first serve basis and performs the intended job (encryption/unlinking/compression).

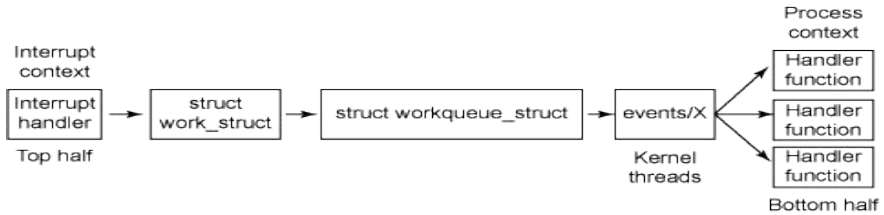

Image reference: https://www.ibm.com/developerworks/library/l-tasklets/index.html

Implementation

1. Userland :-

1)If encryption feature is desired, then store the encryption key entered by the user in proc file. Ex: echo “user_key” > /proc/enc_key_file

```
echo “Hello, this is test file data.” > test1.txt
Echo “12345” > /proc/enc_key_file
./xclone2  -mce  test1.txt
```

2)To recover a lost file:

```
./xclone2 -u <infile in the trashbin folder>
```

Implementation

Kernel land:-

1) Asynchronous :- When a job is enqueued in the queue, if a queue is already full then the producer processes are throttled. There is a limit on the number of producers that can be throttled, if that exceeds then the incoming producers are returned with an error that the job cannot be processed.

   If the job is inserted successfully in the queue, then the worker thread dequeues it based on first come first serve basis, and performs the intended job (encryption/ decryption / compression/ unlinking etc.)

Implementation

2) Synchronous :-

An ioctl is used to set the encryption key that is supplied by the user, and the three steps of compression, encryption and unlinking are performed synchronously.

An ioctl to support the recovery of lost file is provided.

Periodic deletion of files in the trashbin: Runs every 100 ms; checks if number of files in the trashbin directory exceeds the MAX_NUM_FILES , then sort the files according to timestamp and delete the older files.

Purge the trashbin folder completely :- List all the files using iterate_dir and delete.

Implementation

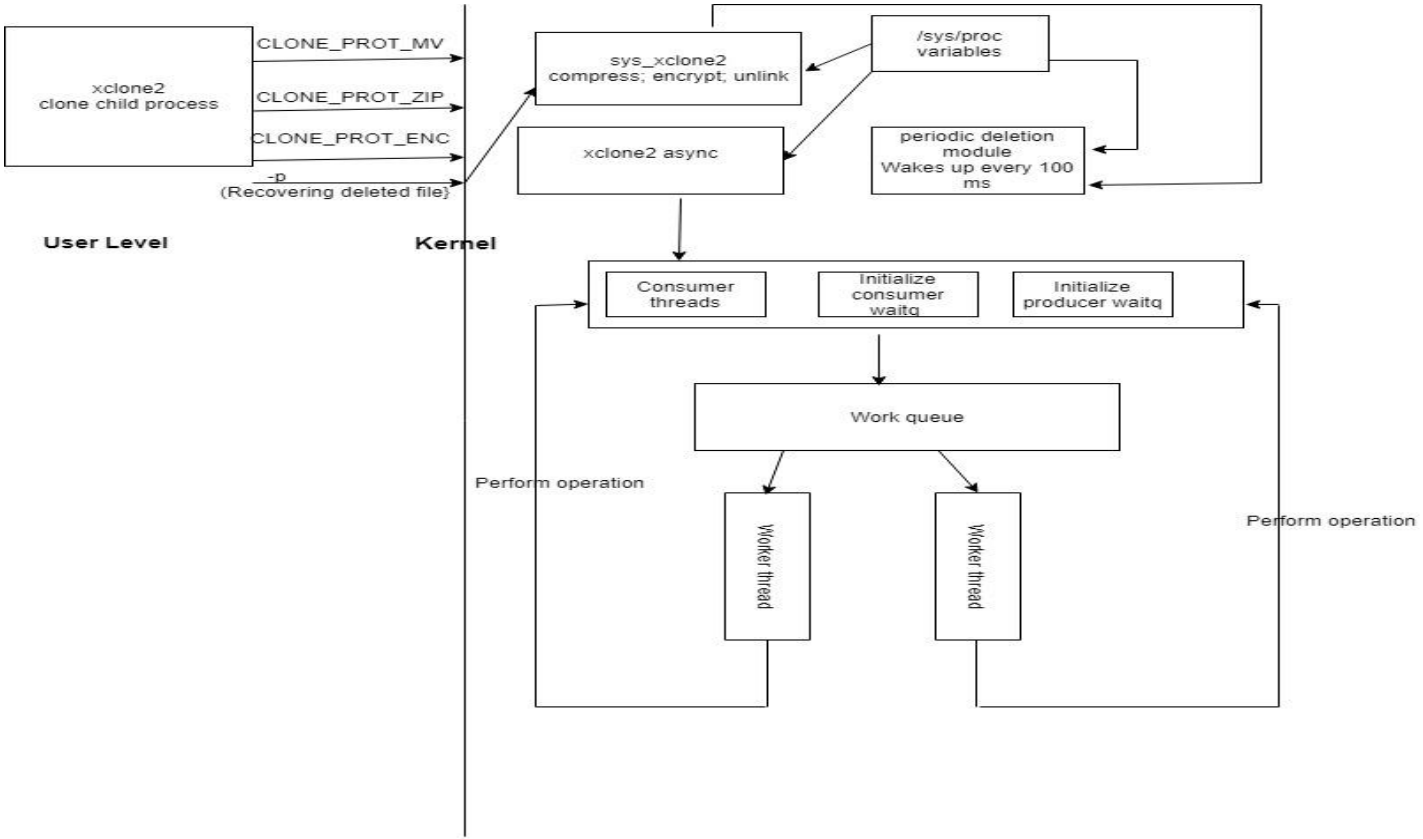

Image source:- Self

Design considerations

1. Asynchronous and synchronous processing are supported through separate syscalls.
2. One work queue for all users
3. Order is kept as compression, encryption and unlinking
4. For recovering a lost file , the order is decryption, decompression and moving.
5. Cleaning up of the work queue while unloading the syscall
6. /sys/proc variables stored as environment variables