Asynchronous System Calls

DESIGN DOCUMENT

Author(s)	*Kumar Anupam (110614410)* *Parth Kamlesh Dandiwala (110422439)* *Aditi Singh (110285096)*
Group Name	*cse506g02*
Course	*Operating Systems CSE 506*
Assignment	*Asynchronous System Calls and Concurrent Processing*

TABLE OF CONTENTS

1 Document Versions

Date	*Author*	*Description*
12/3/15	Aditi Singh	Added text content
12/4/15	Aditi Singh	Added diagrams

2 System Overview

The aim of this project was to design a system to handle asynchronous processing of tasks. High level of how the system works. The files present are:

- **hw3.c** - user program .

- **sys_submitjob.c** - kernel program – implements the producer and the various consumer processes are also present here.

- **Makefile -** contains the code to make and compile code

- **make_and_load_module.sh –** The system call has been implemented as a loadable module. This shell script file has the code to insert and remove module after doing a make.

- **job_struct.h –** contains the structure which is shared between the user level program and the kernel code

3 Features Implemented

Before running any of the below mentioned features (command given in the [] below the description) , run the following command:

sh make_and_load_module.sh

3.1. Concatenation :

performs concatenation when multiple files are given as input and a single file is specified for the result to be written.
[./hw3 –a output_file input_file1 input_file2…. Input_filen]

3.2. Compression:

performs compression of a single input_file .A single file is specified for the result to be written as output_file.
[./hw3 –c output_file input_file]

3.3. Decompression:

performs decompression of a single input_file. A single file is specified for the result to be written as output_file.
[./hw3 –d output_file input_file]

3.4. Checksum:

performs checksum calculation of a single input_file and a single file is specified for the result to be written as output_file.

[./hw3 –k output_file input_file]

3.5. Encryption:

performs encryption of a single input_file and when a single file is specified for the result to be written as output_file. A passphrase is also supplied by the user to be used as a key. If the output file is not present it gets created.

[./hw3 –x passphrase output_file input_file]

3.6. Decryption:

performs decryption of a single input_file and when a single file is specified for the result to be written as output_file. A passphrase is also supplied by the user to be used as a key. If the output file is not present it gets created.

[./hw3 –y passphrase output_file input_file]

3.7. Listing the items in the job queue:

This lists all the jobs currently in the workqueue

[./hw3 –l]

3.8. Canceling a job from the queue:

Cancels a job with pid given by the user which is present in the workqueue.

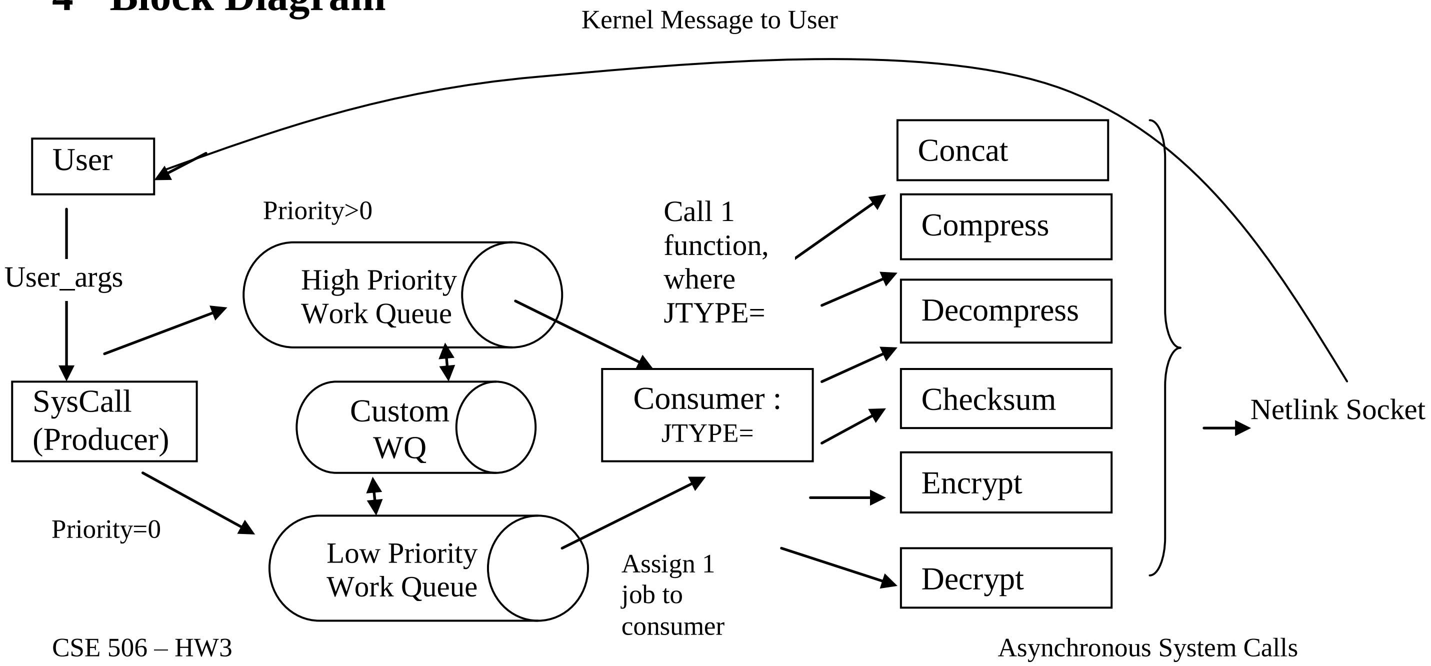
[./hw3 –f pid]

3.9. Changing the priority for a job:

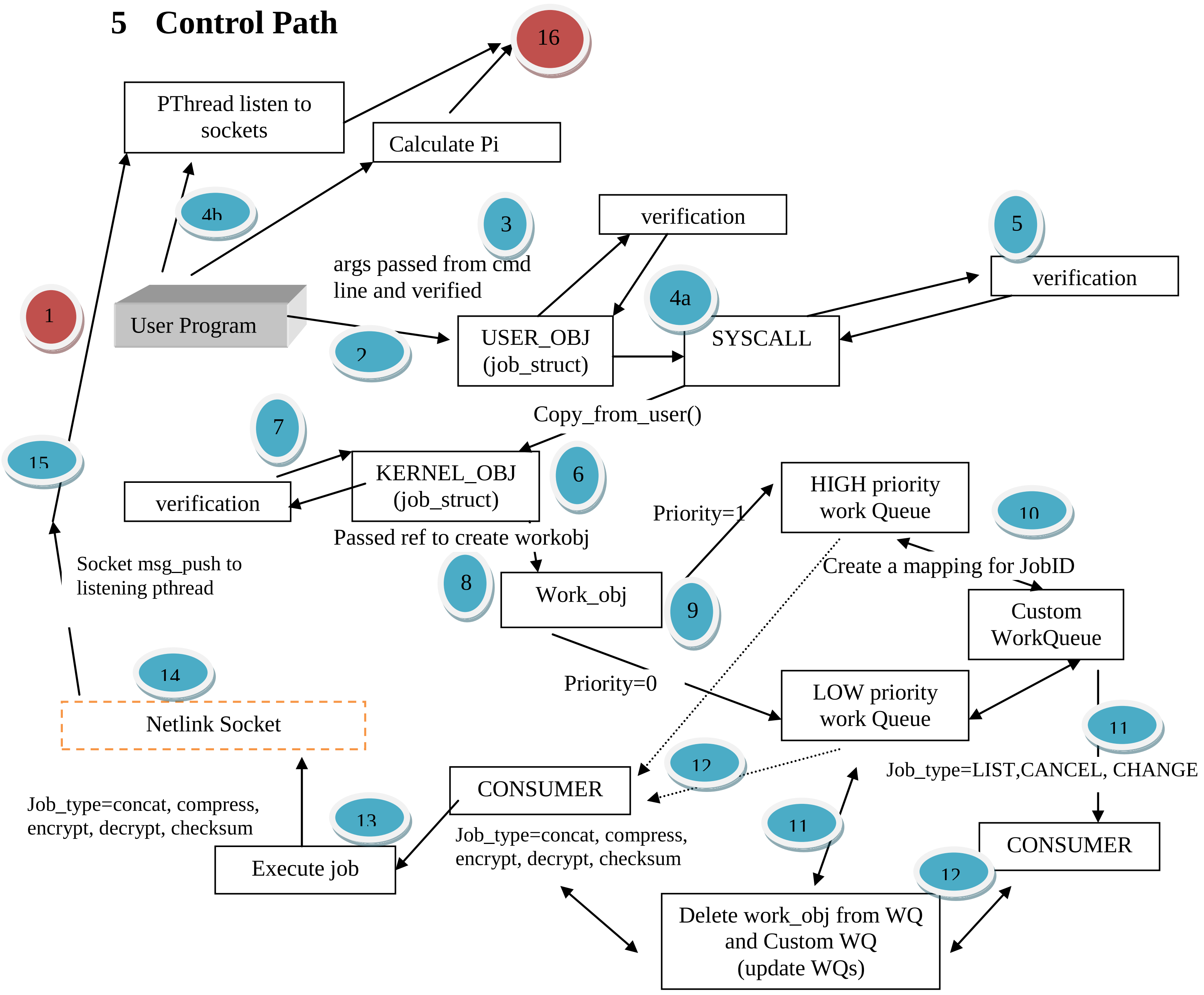
Changes the priority of the job with pid given by the user and values specified as 0 or 1 as the second argument.

[./hw3 –b pid 0/1]

4 Block Diagram



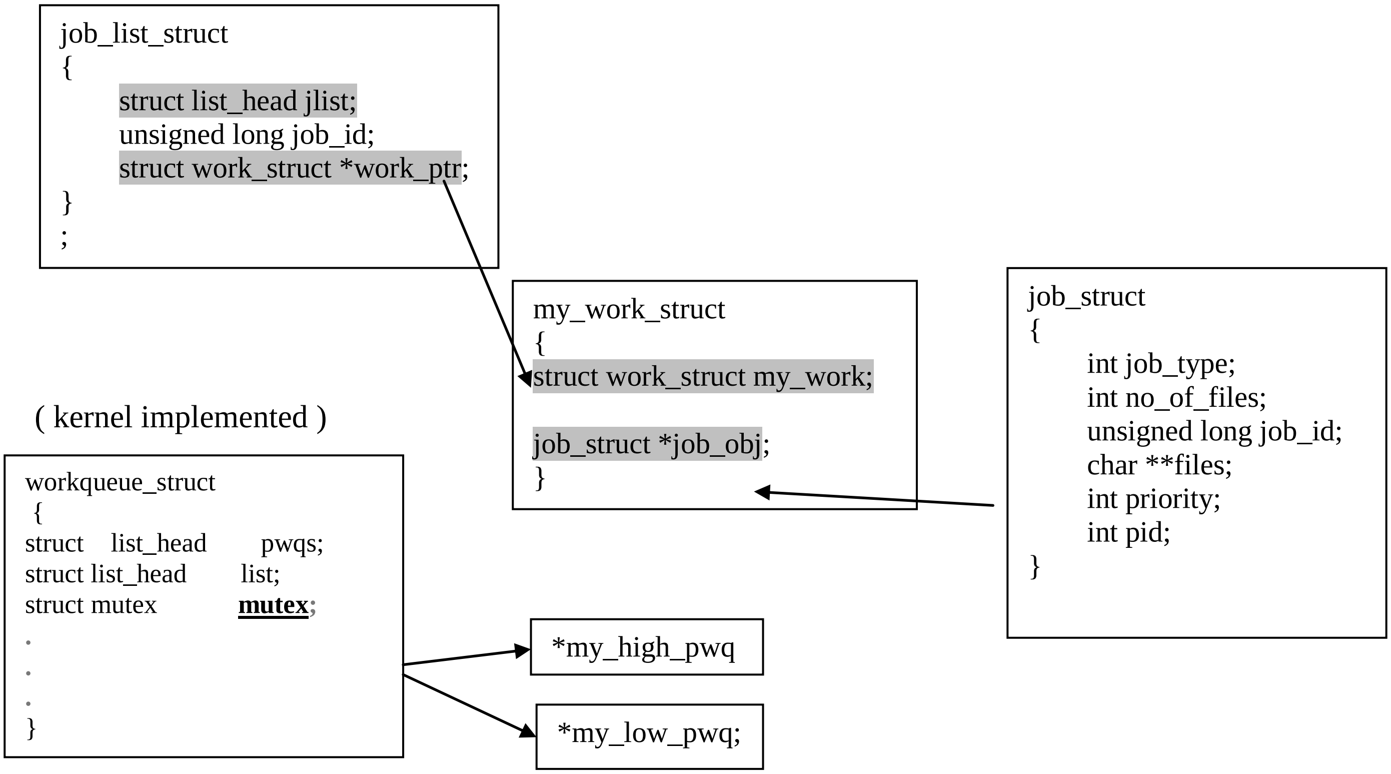
5 Control Path


16
PThread listen to sockets
Calculate Pi
4b
3
verification
5
args passed from cmd line and verified
verification
1
4a
User Program
2
USER_OBJ (job_struct)
SYSCALL
Copy_from_user()
7
15
KERNEL_OBJ (job_struct)
6
verification
HIGH priority work Queue
Priority=1
10
Passed ref to create workobj
Socket msg_push to listening pthread
Create a mapping for JobID
8
Work_obj
9
Custom WorkQueue
14
Priority=0
LOW priority work Queue
11
Netlink Socket
12
Job_type=LIST,CANCEL, CHANGE
CONSUMER
Job_type=concat, compress, encrypt, decrypt, checksum
13
11
CONSUMER
Job_type=concat, compress, encrypt, decrypt, checksum
Execute job
12
Delete work_obj from WQ and Custom WQ (update WQs)

Note :

JID: Job ID

6 Major Data Structures



```
job_list_struct
{
        struct list_head jlist;
        unsigned long job_id;
        struct work_struct *work_ptr;
}
;
```

```
my_work_struct
{
struct work_struct my_work;

job_struct *job_obj;
}
```

```
job_struct
{
        int job_type;
        int no_of_files;
        unsigned long job_id;
        char **files;
        int priority;
        int pid;
}
```

```
workqueue_struct
 {
struct    list_head       pwqs;
struct list_head        list;
struct mutex            mutex;
.
.
.
}
```

7 Functions Implemented

7.1. static void push_msg_to_user(struct sk_buff *skb)

- Kernel uses the netlink socket to communicate with the user.
- It passes the job id to the user when the job is submitted to the kernel
- This job id can be used by the user to cancel or change priority at a later stage
- It is also used by the kernel (consumer process) to communicate the success/failure of the process to the userland.

7.2. static int delete_from_custom_queue(unsigned long job_id)

- takes a job_id as an input and iterates over the list (implemented using kernel datastructure from list.h)
- deletes that list entry
- decrements the job_count

7.3. static void concat(struct work_struct *work_obj)

- This is the concatenation consumer process which takes in multiple input files to be concatenated
- The destination file is appended with the contents of the input files.

7.4. static void compress(struct work_struct *work_obj)

- This is the compression consumer process which takes in one input file to be compressed using CryptoAPI and the “deflate” algorithm
- The destination file is truncated and written with the compressed contents of the input files.

7.5. static void decompress(struct work_struct *work_obj)

- This is the decompression consumer process which takes in one input file to be decompressed using CryptoAPI and the “inflate” algorithm
- The destination file is truncated and written with the decompressed contents of the input files.

7.6. static void checksum(struct work_struct *work_obj)

- This is the checksum consumer process which takes in one input file for which the checksum has to be calculated.
- We have used the MD5 algorithm for the checksum calculation.
- The destination file is truncated and written with the checksum value.

7.7. static void encryption(struct work_struct *work_obj)

- This is the encryption consumer process which takes in one input file which has to be encrypted using a passphrase specified by the user.
- The destination file is truncated or created if does not exists and first written with the hashed passphrase and then with the encrypted data.
- In this case a temp file is created which stores the content of the encrypted data.
- Else if all succeeds, then the contents of the temp file are copied to the destination file and the temp file is deleted.
- If the encryption process fails midway or the copy from temp to destination fails, the temp file is deleted and error is returned.

7.8. static void decryption(struct work_struct *work_obj)

- This is the decryption consumer process which takes in one input file which has to be decrypted using a passphrase specified by the user.
- The destination file is truncated or created if does not exists
- First the passphrase is hashed and checked if the MD5 value written in the input file matches the hashed passphrase.
- Then the destination file is written with the decrypted data.
- In this case a temp file is created which stores the content of the decrypted data.
- Else if all succeeds, then the contents of the temp file are copied to the destination file and the temp file is deleted.
- If the decryption process fails midway or the copy from temp to destination fails, the temp file is deleted and error is returned.

7.9. long validateParams(void *arg)

- This function is used to validate the job_struct arguments (for more details goto section 8.1)

7.10. long copyUserToKernel(job_struct * srcArg, job_struct * destArg)

- copy the priority, job_type , no_of_files and files from the user
- perform validation for each element of the structure (for more details goto section 8.2)
- Malloc memory for each file in the file array, file names
- Copy the values from user structure arguments to designated kernel variables.

7.11. asmlinkage long submitjob(void *arg)

"The submitjob is the producer"

- call the copyUserToKernel() here to copy all arguments passed by the user from the command line.
- Check for each job type and then call the appropriate functions.
- If the job_type is JTYPE_LIST then iterate through the whole list and copy it to a list buffer. Send the whole list buffer to userland using copy_to_user(), along with it's priority.
- For the other appropriate job_type is one of these JTYPE_CONCAT, JTYPE_COMPRESS, JTYPE_DECOMPRESS , JTYPE_CHECKSUM, JTYPE_ENCRYPT or JTYPE_DECRYPT : its appropriate consumer process would be invoked.

  If priority has been set to 0 : then set job priority as LOW, else set it to HIGH
- If the job_type is JTYPE_CANCEL , then iterate through the list and find the appropriate job_id and delete it from the list.
- If job_type is JTYPE_CHANGE then locate the position of the job_id in the queue , delete it from there and copy it to another queue. Copy the corresponding elements for the particular job_id and duplicate it in the other queue too.
- For every job added to the queue update the global variable 'job_id_cnt' by 1.
- If all the above cases succeed proceed to pass the control to the userland, while the consumer proceeds to process the job asynchronously.
- If any of the above cases fail, return an appropriate error message.

7.12. static int __init init_sys_submitjob(void)

- Called when the sys_submitjob module is inserted.
- Allocate a low_priority custom workqueue
- Allocate a high_priority workqueue

7.13. static void __exit exit_sys_submitjob(void)

- called when the sys_submitjob module is removed.
- Delete each entry for the work queue
- Flush the high priority workqueue object
- Flush the low priority workqueue object.

8 Validations

8.1. long validateParams(void *arg):

- check if the structure received from the user is NULL or non accessible using acess_ok(), else return –EFAULT

- check if the job_type is less than 7

  [job_type describes the job that has to be performed on the files specified by the user. eg. concat]

  ```
  if yes then
          if the file pointer is NULL or inaccessible then return -EFAULT
          if the filename address is NULL or inaccessible then return -EFAULT
          if the filename is too long then return -ENAMETOOLONG
  ```

8.2. long copyUserToKernel(job_struct * srcArg, job_struct * destArg):

- check if copy_from_user succeeded for

  ```
  job_type
  priority
  no_of_files
  ```

- check if malloc for the files array succeeded. If not then return –ENOMEM
- check if malloc for the file names succeeded . If not then return –ENOMEM

9 Design Patterns:

- Full path name has to be specified while passing the file name as arguments from the command line from the user program.
- In concatenation operation, the destination file should exist.
- We have two priorities – 1(High) and 0 (low)
- Created a custom work queue in the kernel space to keep a mapping of the job id and the address of the job in the kernel workqueue (high priority WQ and Low Priority WQ).
- We have used the kernel work queue which is populated using the priority submitted by the user (1- high, 0-low) and the kernel scheduler processes the jobs by removing them from the workqueue.
- Before the consumer takes the job from the work queue, the job entry is removed from the workqueue and also the custom workqueue , so that if the user tries to cancel or change priority for the job , he would get an error.
- We have used two spinlocks
  - job_cnt_lock – is used to lock the global count of the job count
  - job_id_cnt_lock – is used to lock the universal count on the job_id
- We have used 1 mutex
  - Used to lock the custom queue list