A miscellanious device driver will be created with name /dev/numpipe.

This device driver will act like a pipe and support read and write operations.

To solve issues faced in Task B, synchronization mechanism will be implemented in this kernel driver. Kernel primitives such as mutex i.e. mutual exclusion will be used to handle and avoid issue faced in Task B.

Tasks:

1. Producers write numbers to /dev/numpipe.

Solution: Write system call invoked in user-space(producer\_numbers.c) will execute write function written inside the /dev/numpipe code.

2. Consumers read numbers from /dev/numpipe and print it on the screen.

Solution: Read system call invoked in user-space(consumer\_numbers.c) will execute read function written inside the /dev/numpipe code.

3. When the pipe is full, i.e. when there are N numbers are stored in /dev/numpipe, then any producer trying to write will block.

Solution: This scenario will be handled inside the write code of /dev/numpipe.

This scenario will be handled while performing down operation on empty.

4. When the pipe is empty, i.e. when there are no numbers in /dev/numpipe, then any consumer trying to read will block.

Solution: This scenario will be handled inside the read code of /dev/numpipe.

This scenario will be handled while performing down operation on full.

SKELETON:

Skeleton of read() and write() of device driver will be as follows:

//define a semaphore named empty and full

//define a mutex named mutex

myread() //This code will be executed by consumer

{

down\_interruptible(&full); //Decrement full count

//If down returns anything other than zero than down operation failed

//proceed if not interrupted by user else exit with return code

//Take mutex\_lock\_interruptible i.e. enter critical section

//Perform read operation

//If write successfull then perform mutex\_unlock(&mutex)

//Exit critical section

//Increase empty count

}

mywrite() //This code will be executed by producer

{

down\_interruptible(&empty); //Decrement empty count

//If down returns anything other than zero than down operation failed

//proceed if not interrupted by user else exit with return code

//Take mutex\_lock\_interruptible i.e. enter critical section

//Perform write operation

//If write successfull then perform mutex\_unlock(&mutex)

//Exit critical section

//Increase full count

}