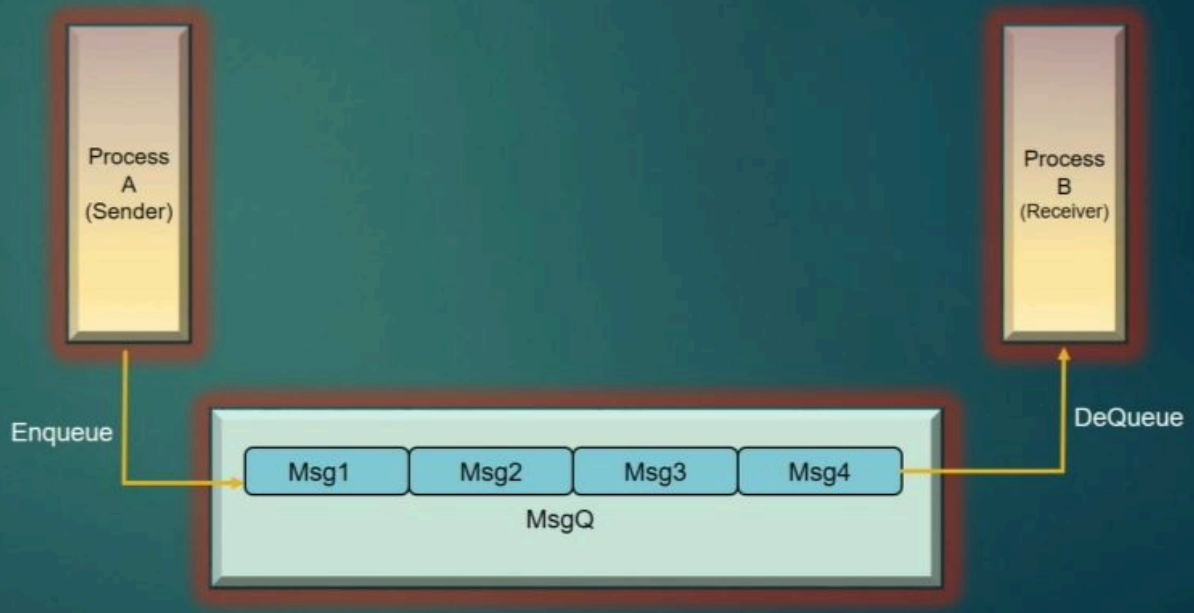
1.1

Message Queues

1. IPC using Message Queues
2. Msgq Concepts
3. MsgQ management APIs
4. Using a MsgQ
5. Code Walk – Step by Step



IPC – Techniques -> Message Queue

1.2

- Linux/Unix OS provides another mechanism called *Message Queue* for carrying out IPC
- Process(s) running on same machine can exchange any data using Message queues
- Process can create a new message Queue Or can use an existing msgQ which was created by another process
- A Message Queue is identified uniquely by the ID, No two msgQ can have same ID
- Message Queue resides and manage by the Kernel/OS
- Sending process A can post the data to the message Queue, Receiving process B reads the data from msg Q
- Process which creates a msgQ is termed as owner Or creator of msgQ

Udemy

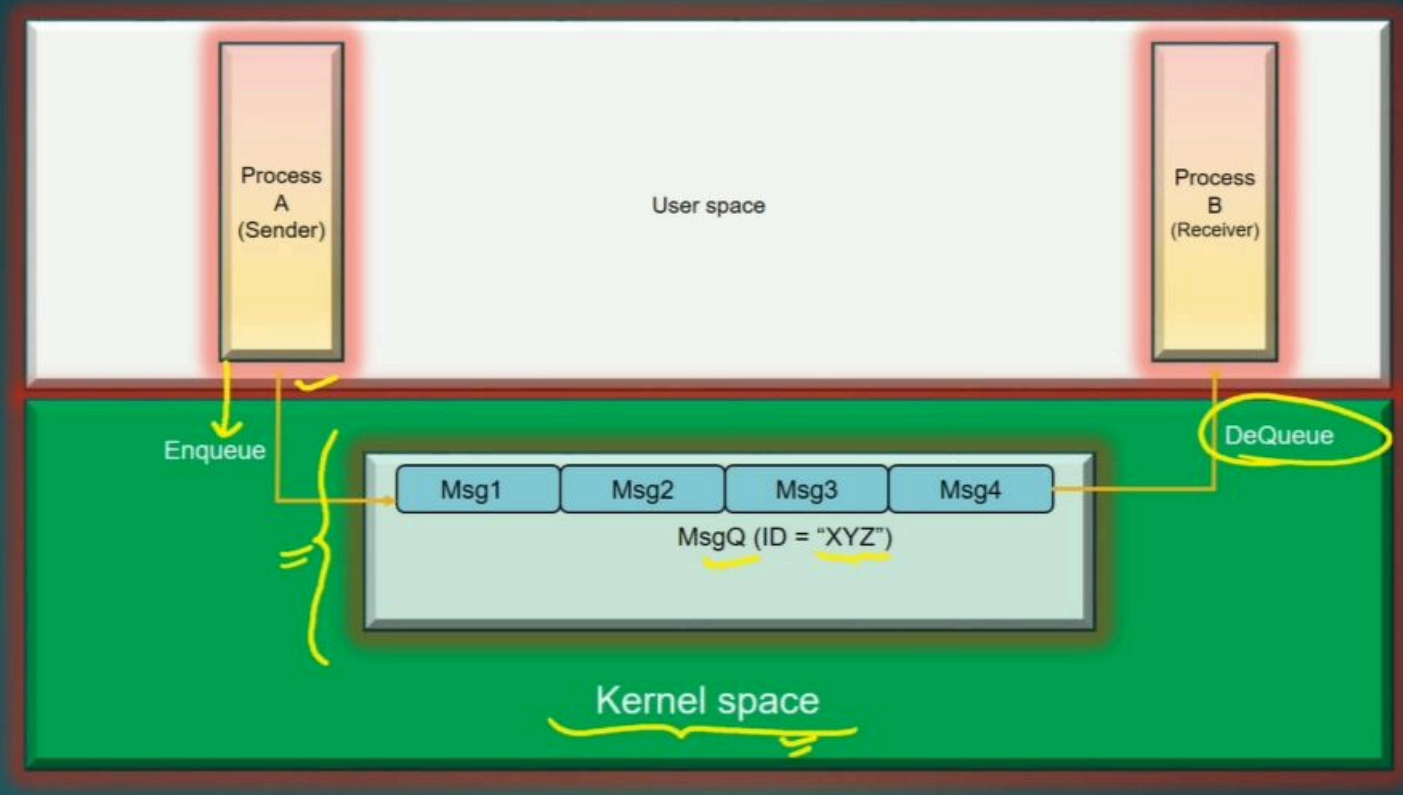
2.1

KERNEL RESOURCE

Udemy

IPC – Techniques -> Message Queue

2.2



- There can be multiple message Queues created by several processes in the system
- Each message Queue is identified by a unique name called msgQ ID which is a string

Udemy

3.1

MQ_OPEN( )

1. Message Queue Creation

3.2

A Process can create a new msgQ or use an existing msgQ using below API :

```
mqd_t
mq_open (const char *name,
         int oflag);
```

```
mqd_t
mq_open (const char *name,
         int oflag, mode_t mode,
         struct mq_attr *attr);
```

- Two flavors of the API

- *name* – Name of msg Q , eg "/server-msg-q"
- *oflag* – Operational flags
  - O_RDONLY : process can only read msgs from msgQ but cannot write into it
  - O_WRONLY : process can only write msgs into the msgQ but cannot read from it
  - O_RDWR : process can write and read msgs to and from msgQ
  - O_CREAT : The queue is created if not exist already
  - O_EXCL : mq_open() fails if process tries to open an existing queue. This flag has no meaning when used alone. Always used by OR-ing with O_CREAT flag
- *mode* - Permissions set by the owning process on the queue, usually 0660
- *attr* – Specify various attributes of the msgQ being created
  - Like maximum size the msgQ can grow, should be less than or equal to */proc/sys/fs/mqueue/msg_max*
  - Maximum size of the msg which msgQ can hold , should be less than or equal to */proc/sys/fs/mqueue/msgsize_max*

```
struct mq_attr {
    long mq_flags;       /* Flags: 0 */
    long mq_maxmsg;      /* Max. # of messages on queue */
    long mq_msgsize;     /* Max. message size (bytes) */
    long mq_curmsgs;     /* # of messages currently in queue */
};
```

If mq_open() succeeds, it returns a file descriptor (a handle) to msgQ. Using this handle, we perform All msgQ operations on msgQ throughout the program

3.3

1. Message Queue Creation

A Process can create a new msgQ or use an existing msgQ using below API :

Example 1: Without attributes

```
mqd_t msgq;

if ((msgq = mq_open ("/server-msg-q", O_RDONLY | O_CREAT | O_EXCL, 0660, 0)) == -1) {
        perror ("Server: mq_open (server)");
        exit (1);
}
```

Example 2: With attributes

```
mqd_t msgq;

struct mq_attr attr;

attr.mq_flags = 0;
attr.mq_maxmsg = 10;
attr.mq_msgsize = 4096;
attr.mq_currmsgs = 0;

if ((msgq = mq_open ("/server-msg-q", O_RDONLY | O_CREAT | O_EXCL, 0660, &attr)) == -1) {
        perror ("Server: mq_open (server)");
        exit (1);
}
```

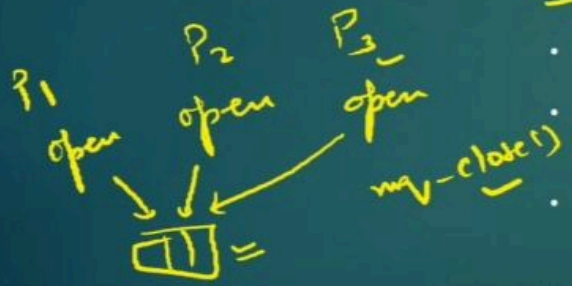
4.1

2. Message Queue Closing

A Process can close a msgQ using below API :

```
int
mq_close (mqd_t msgQ);
```

Return value :
0 – sucess
-1 - failure



- After closing the msgQ, the process cannot use the msgQ unless it open it again using mq_open()
- Operating system removes and destroy the msgQ if all processes using the msgQ have closed it
- OS maintains information regarding how many process are using same msgQ (invoked mq_open()). This concept is called *reference counting*
- msgQ is a kernel resource which is being used by application process. For kernel resources, kernel keeps track how many user space processes are using that particular resource
- When a kernel resource (msgQ in our example) is created for the first time by appln process, *reference_count = 1*
- if other process also invoke open() on existing kernel resource (mq_open() in our case) , kernel increments *reference_count by 1*
- When a process invoke close() in existing kernel resource (mq_close() in our case), kernel decrements *reference_count by 1*
- When *reference_count = 0*, kernel cleanups/destroys that kernel resource

Remember, Kernel resource could be anything, it could be socket FD, msgQ FD etc

Udemy

5.1

mq_send()
sending a msg

0x00091A

5.2

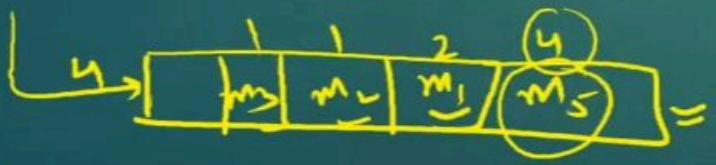
3. Enqueue a Message

A Sending Process can place a message in a message Queue using below API :

```
int
mq_send (mqd_t msgQ,
         char *msg_ptr,
         size_t msg_len,
         unsigned int msg_prio);
```

Return value :
0 – success
-1 - failure

- ***mq_send*** is for sending a message to the queue referred by the descriptor msgQ.
- The ***msg_ptr*** points to the message buffer. ***msg_len*** is the size of the message, which should be less than or equal to the message size for the queue.
- ***msg_prio*** is the message priority, which is a non-negative number specifying the priority of the message.
  - Messages are placed in the queue in the decreasing order of message priority, with the older messages for a priority coming before newer messages.
- If the queue is full, ***mq_send*** blocks till there is space on the queue, unless the O_NONBLOCK flag is enabled for the message queue, in which case *mq_send* returns immediately with errno set to EAGAIN.



Udemy


6.1
Dequeue a msg
mq_receive()

3. Dequeue a Message

6.2

A Receiving Process can dequeue a message from a message Queue using below API :

```
int
mq_receive (mqd_t msgQ,
            char *msg_ptr,
            size_t msg_len,
            unsigned int *msg_prio);
```

Return value
n_bytes of recvd msg – success
-1 - failure

- ***mq_receive*** is for retrieving a message from the queue referred by the descriptor msgQ.
- The ***msg_ptr*** points to the empty message buffer. ***msg_len*** is the size of the buffer in bytes.
- The oldest msg of the highest priority is deleted from the queue and passed to the process in the buffer pointed by ***msg_ptr.***
- If the pointer ***msg_prio*** is not null, the priority of the received message is stored in the integer pointed by it
- The default behavior of ***mq_receive*** is to block if there is no message in the queue. However, if the O_NONBLOCK flag is enabled for the queue, and the queue is empty, ***mq_receive*** returns immediately with errno set to EAGAIN.
- On success, ***mq_receive*** returns the number of bytes received in the buffer pointed by msg_ptr.

Udemy

7.1

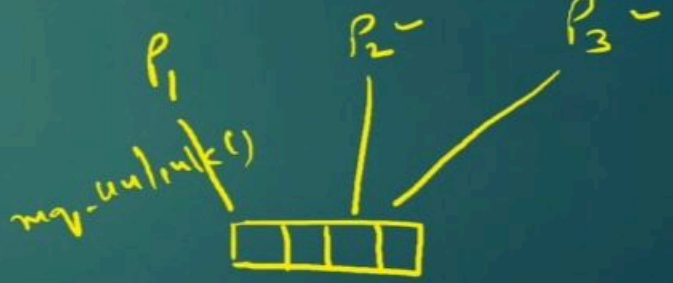
mq_unlink

4. Destroying a Message Queue

A creating Process can destroy a message queue using below API :

```
int
mq_unlink (const char *msgQ_name);
```

- ***mq_unlink*** destroys the msgQ (release the kernel resource)
- Should be called when the process has invoked mq_close() on msgQ
- return -1 if it fails, 0 on success
- Postpone if other processes are using msgQ



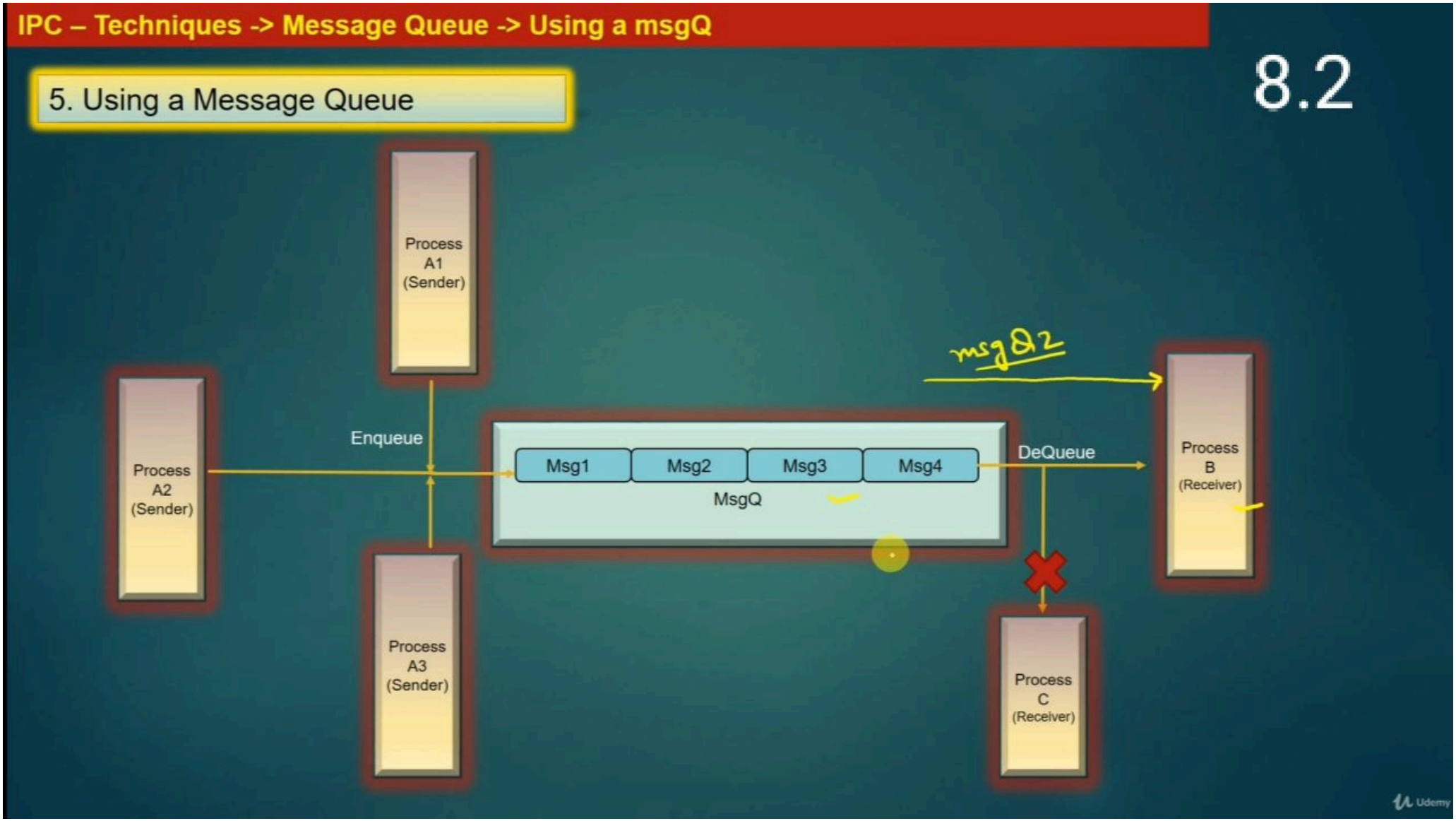
Udemy

5. Using a Message Queue

- A Message Queue IPC mechanism usually supports N:1 communication paradigm, meaning there can be N senders but 1 receiver per message Queue
- Multiple senders can open the same msgQ using msgQ name, and enque their msgs in the same Queue
- Receiver process can dequeue the messages from the message Queue that was placed by different sender processes
- However, Receiving process can dequeue messages from different message Queues at the same time (Multiplexing using select())
    - A msg Queue can support only one client process
    - Client can dequeue msgs from more than one msg Queues
    - No limitation on Server processes

Udemy

5. Using a Message Queue



9.1

CODE WALK

Walk and Implementation

Download code :

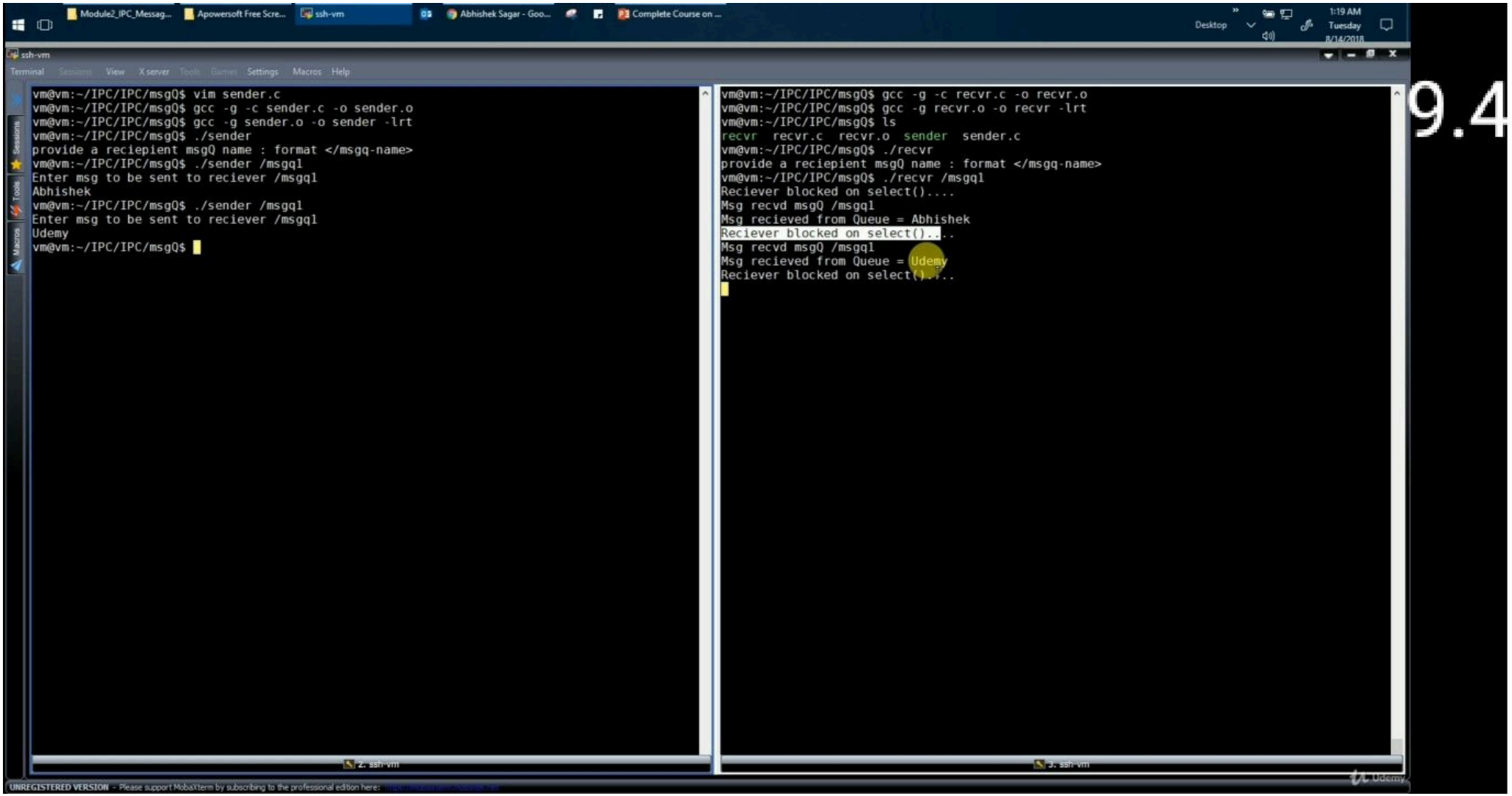
git clone https://github.com/sachinites/IPC

Dir : IPC/IPC/msgQ

Files : sender.c
recvr.c

9.3

DEMONSTATION


Module2_IPC_Messag...
Apowersoft Free Scre...
ssh-vm
Abhishek Sagar - Goo...
Complete Course on ...
Desktop
1:19 AM
Tuesday
8/14/2018
ssh-vm
Terminal Sessions View X server Tools Games Settings Macros Help
vm@vm:~/IPC/IPC/msgQ$ vim sender.c
vm@vm:~/IPC/IPC/msgQ$ gcc -g -c sender.c -o sender.o
vm@vm:~/IPC/IPC/msgQ$ gcc -g sender.o -o sender -lrt
vm@vm:~/IPC/IPC/msgQ$ ./sender
provide a reciepient msgQ name : format </msgq-name>
vm@vm:~/IPC/IPC/msgQ$ ./sender /msgq1
Enter msg to be sent to reciever /msgq1
Abhishek
vm@vm:~/IPC/IPC/msgQ$ ./sender /msgq1
Enter msg to be sent to reciever /msgq1
Udemy
vm@vm:~/IPC/IPC/msgQ$
2. ssh-vm
vm@vm:~/IPC/IPC/msgQ$ gcc -g -c recvr.c -o recvr.o
vm@vm:~/IPC/IPC/msgQ$ gcc -g recvr.o -o recvr -lrt
vm@vm:~/IPC/IPC/msgQ$ ls
recvr recvr.c recvr.o sender sender.c
vm@vm:~/IPC/IPC/msgQ$ ./recvr
provide a reciepient msgQ name : format </msgq-name>
vm@vm:~/IPC/IPC/msgQ$ ./recvr /msgq1
Reciever blocked on select()....
Msg recvd msgQ /msgq1
Msg recieved from Queue = Abhishek
Reciever blocked on select()...
Msg recvd msgQ /msgq1
Msg recieved from Queue = Udemy
Reciever blocked on select()...
3. ssh-vm
UNREGISTERED VERSION - Please support MobaXterm by subscribing to the professional edition here:
Udemy
9.4