# Lecture # 5.3
# Message Queues

**Course:** **Advance Operating System**

**Instructor: Arif Butt**

**Punjab University College of Information Technology (PUCIT)**
**University of the Punjab**

Source Code files available at: **https://bitbucket.org/arifpucit/spvl-repo/src**
Lecture Slides available at: **http://arifbutt.me**

# Today's Agenda

- Introduction to Message Queues
- Difference between FIOFs and Message Queues
- Two Implementations of Message Queues
  - System-V API
  - POSIX API
- Creating/Opening a System-V Message Queues
- Sending messages in a message queue
- Receiving messages from a message queue
- Message Queue control operations

# Introduction to Message Queues

- **Message Queues** can be used to pass messages between related or unrelated processes executing on same machine. Message queues are somewhat like pipes and fifos, but differ in two important aspects:
  - ➔ **First**, message boundaries are preserved, so that readers and writers communicate in units of messages
  - ➔ **Second**, message queues are kernel persistent
- A Message Queue can be thought of as a linked list of messages in the kernel space. Processes with adequate permissions can put messages onto the queue & processes with adequate permissions can remove messages from the queue

# Difference between FIFOs & Message Queues

- Message Queues have Kernel persistence while FIFOS have process persistence
- In pipes, a process read or write stream of bytes, while in message queues a process read or write a complete delimited message; it is not possible to read a partial message leaving rest behind in IPC object
- In pipes a write makes no sense unless a reader also exists. In message queues there is no requirement that some reader must be waiting before a process writes a message to the queue
- Message queues are priority driven. Queue always remains sorted with the oldest message of the highest priority at front
- A process can determine the status of a message queue

# SYSTEM-V vs POSIX Message Queues

The three IPC mechanisms (message queues, semaphores and shared memory) are collectively referred to as either System-V IPC or the POSIX IPC. Both System-V as well as the POSIX standard provides API for the implementation of these IPC mechanism. The two major differences between these two implementations for message queues are:

- System-V message queues can return a message of any desired priority, while POSIX message queue always return the oldest message of highest priority
- POSIX message queues allow the generation of signal when a message is placed onto an empty queue, where as nothing similar is provided by System-V message queue

# APIs Related to Message Queues

| Interface | System-V API | POSIX API |
|---|---|---|
| Header file | `<sys/msg.h>` | `<mqueue.h>` |
| Data Structure | `msqid_ds` | `mqd_t` |
| Create/open | `msgget()` | `mq_open()` |
| Close | `none` | `mq_close()` |
| Perform IPC | `msgsnd(), msgrcv()` | `mq_send(),mq_receive()` |
| Control operations | `msgctl()` | `mq_getattr(),mq_setattr(), mq_notify()` |

# Creating Or Opening A Message Queue

```
int msgget(key_t key, int msgflag);
```

- To create a brand new message queue or to get the identifier of an existing queue we use the `msgget()` system call, which on success returns a unique message queue identifier
- If a message queue associated with the first argument, `key` already exist, the call returns the identifier of the existing message queue, otherwise it creates a new message queue
- We can use `IPC_PRIVATE` constant as first argument. A parent process creates message queue prior to performing a `fork()`, and the child inherits the returned message queue identifier. For unrelated processes we can use this constant, but in that case the creator process has to write the returned message queue identifier in a file that can be read by the other process
- The second argument `msgflag` is normally `IPC_CREAT|0666`

# Creating Or Opening A Message Queue (cont...)

```
int msgget(key_t key, int msgflag);
```

- Instead of using `IPC_PRIVATE` constant as first argument, processes can use the `ftok()` library call with the same arguments to generate a unique key. The key is then used as first argument to `msgget()` to either generate a new message queue identifier or get an existing one

```
key_t ftok(char *pathname, int proj);
```

- The key returned by `ftok()` is a 32 bit value, created by taking:
  - Least significant 8 bits from `proj` argument
  - Least significant 8 bits of minor device number of the device containing the filesystem on which the file in the first argument reside
  - Least significant 16 bits of the inode number of the file referred by first argument `pathname`

# Sending Messages

```
int msgsnd(int msqid, const void* msgp, size_t msgsz,
                  int msgflg);
```

- The `msgsnd()` system call is used to send a message to the message queue identified by its first argument, which is the message queue identifier
- The second argument is a pointer to a structure of type `msgbuf` having following two fields:

```
struct msgbuf{
  long mtype;  //used to retrieve a message by type
  char mtext[512];
  }
```

- The third argument `msgsz` is the size of `mtext` field in the structure `msgbuf`
- The fourth argument `msgflag` can be 0 or `IPC_NOWAIT`

# Receiving Messages

```
int msgrcv(int msqid, void* msgp, size_t maxmsgsz,
           long msgtype, int msgflag);
```

- The `msgrcv()` system call reads and removes a message from message queue identified by its first argument `msqid`, and copies its contents into the buffer pointed to by its second argument `msgp`
- The third argument `maxmsgsz`, specifies the maximum space available in the `mtext` field
- The fifth argument `msgflg` is a bit mask, normally kept as `IPC_NOWAIT`
- The fourth argument `msgtype` is used to specify as to which message is to be removed and returned to the caller based. This is achieved by specifying the `msgtype` field of the `struct msgbuf`

| msgtype | Description |
|---|---|
| `msgtype == 0` | First message from queue is removed and returned |
| `msgtype > 0` | First message from queue whose `mtype` field equals to `msgtype` is removed and returned to calling process |
| `msgtype < 0` | First message of the lowest `mtype` field less than or equal to absolute value of `msgtype` is removed & returned |

# Example: Receiving Messages

| Queue posn | Msgtype | Msg Body |
|---|---|---|
| 1 | 300 | …. |
| 2 | 100 | …. |
| 3 | 200 | …. |
| 4 | 400 | …. |
| 5 | 100 | …. |

**Suppose that we have a message queue containing msgs as shown and we perform `msgrcv()` calls of the following form**

```
msgrcv(id,&msg,maxmsgsz,0,0);
```

**Would retrieve msgs in following order:**

1(mtypr=300)
2(mtypr=100)
3(mtypr=200)
4(mtypr=400)
5(mtypr=100)

```
msgrcv(id,&msg,maxmsgsz,100,0);
```

**Would retrive msgs in following order**

2(mtypr=100)
5(mtypr=100)

Any further calls would block

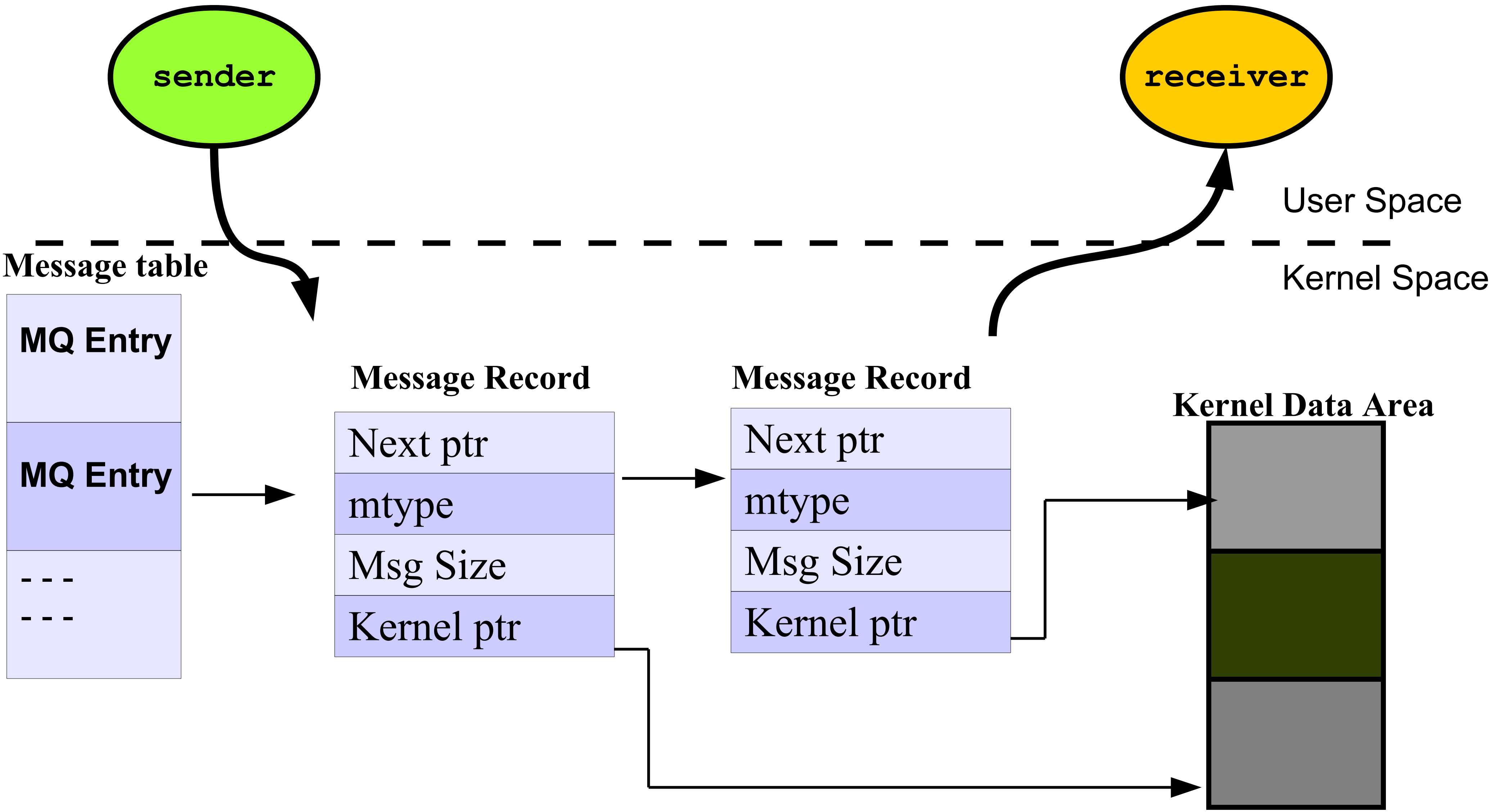
```
msgrcv(id,&msg,maxmsgsz,-300,0)
```

**Would retrive msgs in following order**

2(mtypr=100)
5(mtypr=100)
3(mtypr=200)
1(mtypr=300)

Any further call would block, since type of the remaining message(400) exceeds 300

# Message Queue As Linked List In Kernel



# Sys-V Message Queues
# Proof of Concept

**`sender.c, receiver.c`**

# Things To Do


If you have problems visit me in counseling hours. . . .