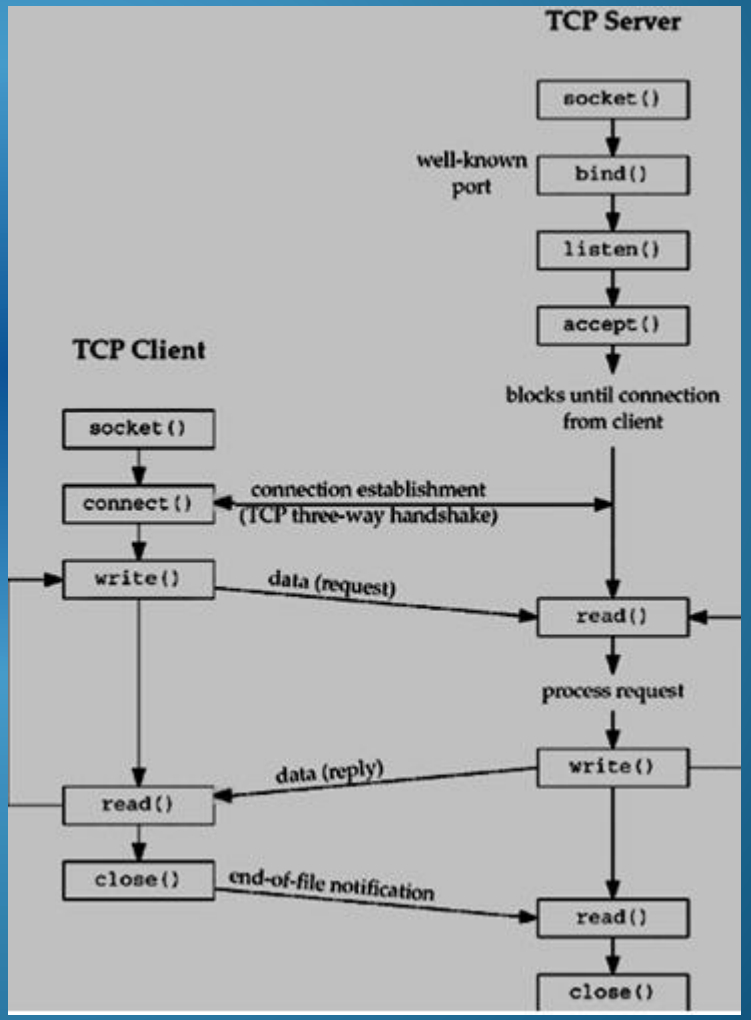
CS240 Operating Systems, Communications and Concurrency

Interprocess Communication Mechanisms

Socket Communication

Recall the socket mechanism which uses an underlying transport protocol such as TCP to create a connection between two end points on a network.

The paired sockets form a bi-directional streamed communication channel between client and server.

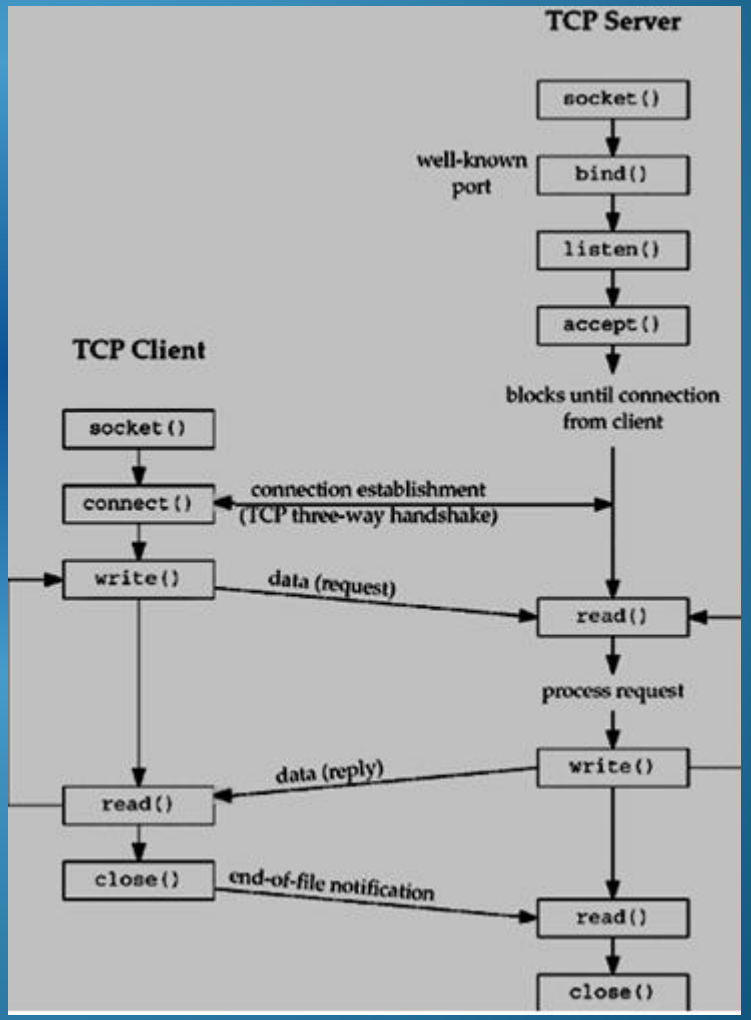


CS240 Operating Systems, Communications and Concurrency

Remote Procedure Call

When a client communicates with a server, it usually wants to execute a function at the server or retrieve data of some kind.

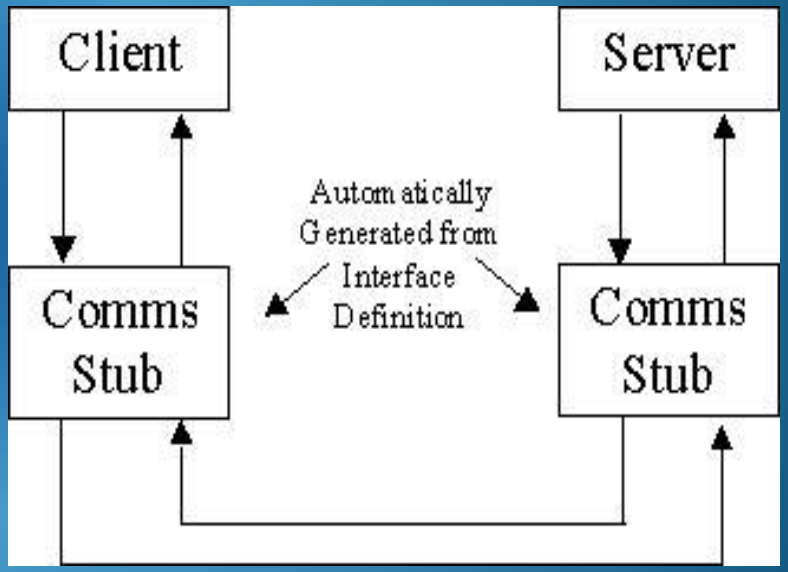
Using socket byte stream communication is really flexible for creating your own protocols but it places the burden on the programmer for writing all the code required to identify and invoke functions and exchange and parse parameters and results as a series of byte serialised message exchanges.



CS240 Operating Systems, Communications and Concurrency

Remote Procedure Call

RPC is a more convenient communication abstraction built on top of an underlying endpoint connection mechanism like sockets over TCP/IP which is intended to make client server programming easier. It enables you to call a remote function in a similar manner to a local one.

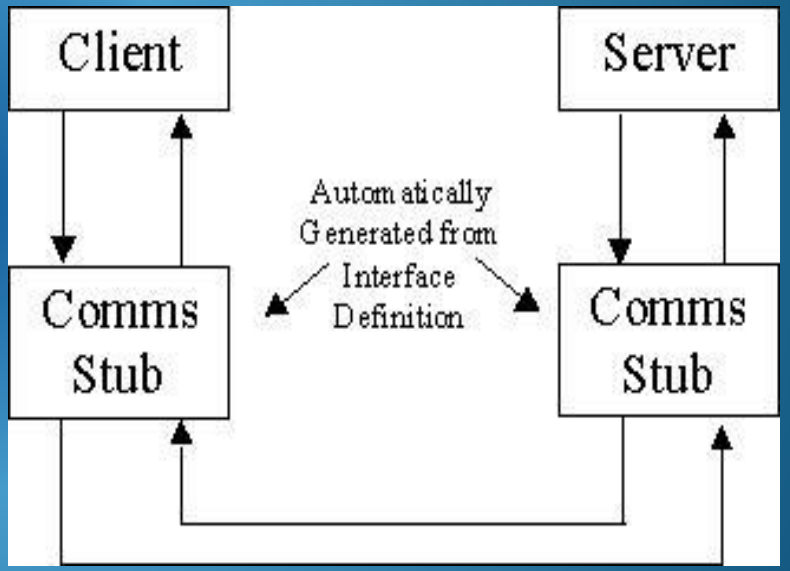


In procedural languages like C, this mechanism is called RPC, in object oriented environments like Java VM, where we are invoking methods on a remote object, it is known as RMI. (Remote Method Invocation)

CS240 Operating Systems, Communications and Concurrency

Remote Procedure Call

An RPC service will define an interface enabling communications stubs to be generated automatically from that interface.



The implementation of the RPC mechanism is hidden within the communication stub code. The stubs are then linked statically with the client and server code.

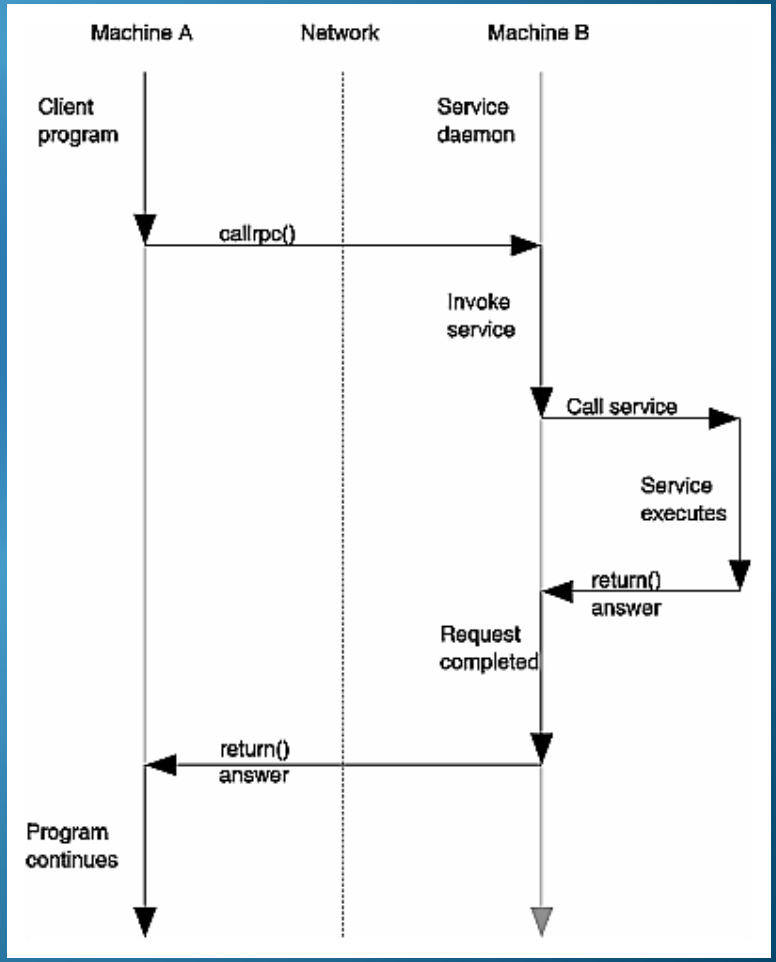
In the labs, please look out for the complexity difference in the code for socket based communication vs RPC implementation. This is the point to take away, that RPC is a simpler abstraction to use for a programmer but the socket based approach offers more low level control over the semantics of that interaction.

CS240 Operating Systems, Communications and Concurrency

Remote Procedure Call

How RPC works...

A client process calls the desired procedure and supplies arguments to it. The client RPC stub mechanism packs the arguments into a message and sends it to the appropriate server process. The receiving stub decodes it, identifies the procedure, extracts the arguments and calls the procedure locally. The results are then packed into a message and sent back in a reply which is passed back by the stub to the client as a procedure return.

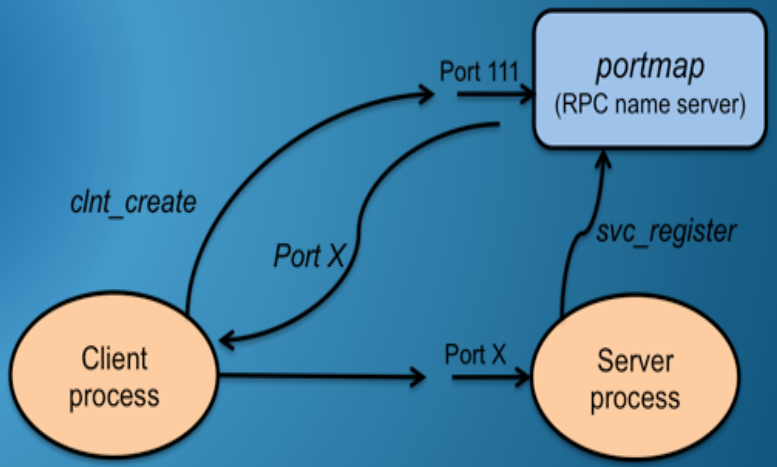


CS240 Operating Systems, Communications and Concurrency

Remote Procedure Call

How to find the server's port address...

Although RPC stub mechanisms understand the rules of how to communicate with a service interface, they may not know where the particular service exists on the network. The service could start up on any machine using any communication port.



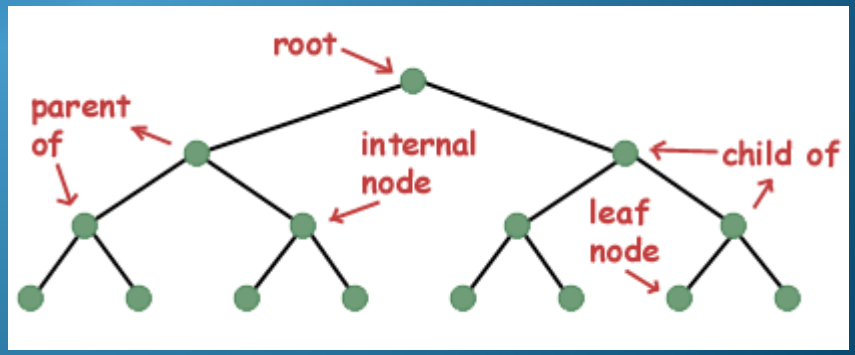
The server's socket is located by first communicating with a directory service with which the server has registered and then establishing a connection over which RPC style exchanges can take place.

CS240 Operating Systems, Communications and Concurrency

Resolving Address Space References

Some implementation problems make it difficult to make the remote procedure call mechanism completely transparent.

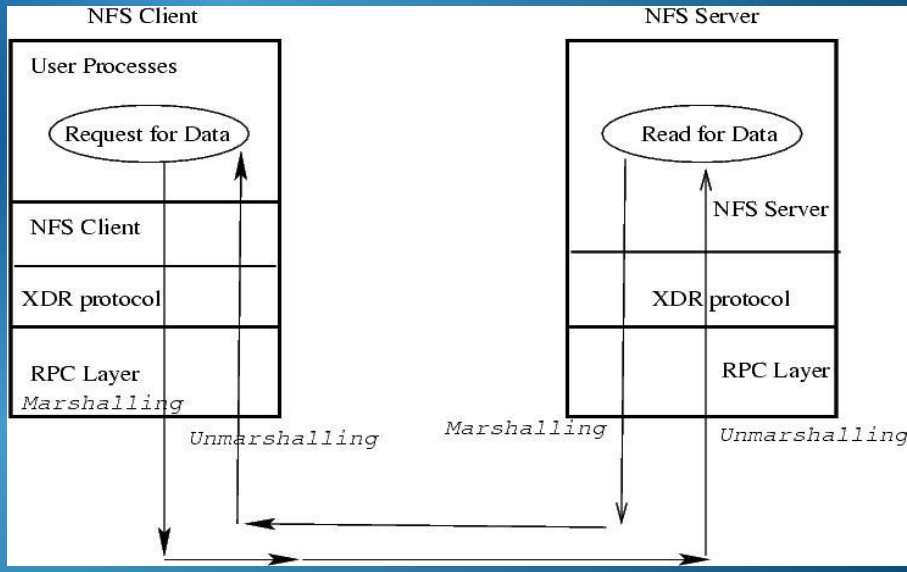
Arguments are passed by value, as in the remote address space, pointer references cannot be evaluated correctly. This makes it difficult to pass memory pointer based data structures or object references for example.



CS240 Operating Systems, Communications and Concurrency

Data Format Translation

Another difficulty may result if the server is running on a different machine architecture to the client or was written using a different programming language. It will be necessary for the RPC mechanism to find an agreeable external data representation format for representing the information exchanged so that it is interpreted correctly by both parties.

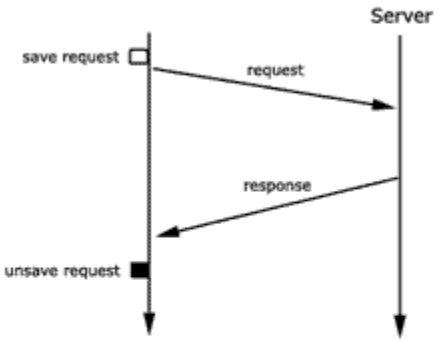


CS240 Operating Systems, Communications and Concurrency

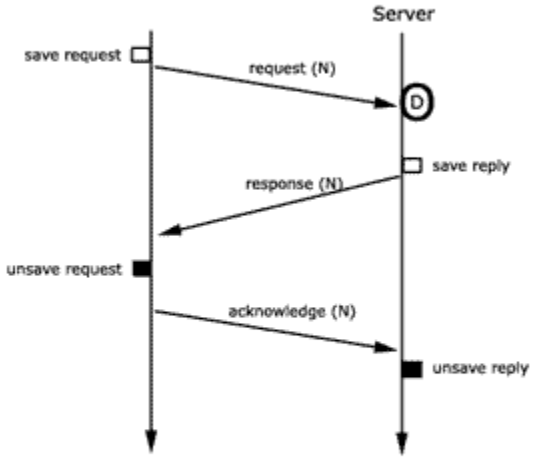
Failure Semantics

Different fault handling execution semantics may apply...

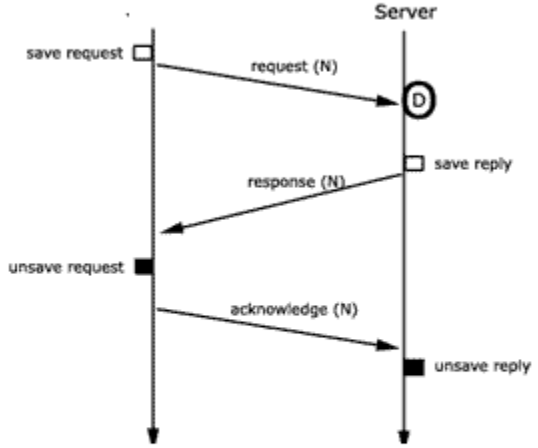
RPC Semantics



At least once requires client to store request for repeated transmission.



At Most Once requires requests and responses to be labelled with sequence numbers and saved at client and server to allow for retransmission.



Exactly Once - requires requests and responses to be written to stable storage to preserve effect if server crashes.

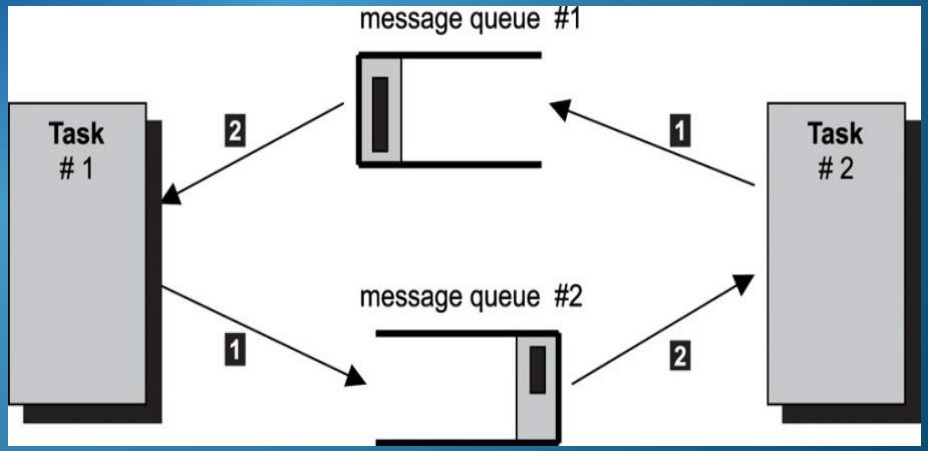
CS240 Operating Systems, Communications and Concurrency

Unix Interprocess Communication Mechanisms

Message Queues

Message queues allow processes to exchange data in the form of **whole messages** asynchronously. No rendezvous required.

Messages have an **associated priority** and are queued in priority order.



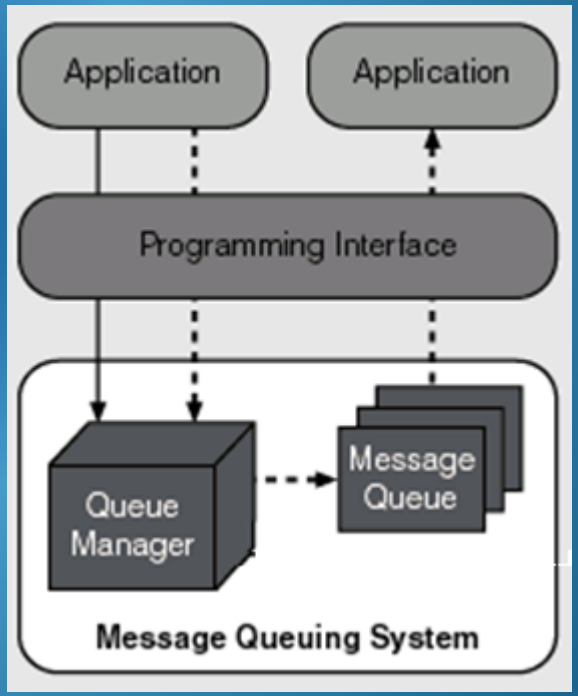
CS240 Operating Systems, Communications and Concurrency

Unix Interprocess Communication Mechanisms

Message Queue Manager

The operating system maintains the message queue persistently and independently of user processes until it is explicitly unlinked (destroyed) by a user process or the system is shutdown.

Communicating processes must have access permission to a particular message queue and the message queue API.



Each created message queue is identified by a name string of the form "/*somename*"

CS240 Operating Systems, Communications and Concurrency

Unix Interprocess Communication Mechanisms

Comparison of message queues vs pipes

Pipes

- No internal structure, just a byte stream
- Cannot distinguish between content of different writers
- No priority for writers
- Cannot determine state of pipe content

Message Queues

- Internal structure and geometry set by user
- Separate messages are distinguishable
- Queue sorted on message priority
- Process can determine status of queue, maximum msg size

CS240 Operating Systems, Communications and Concurrency

Unix Interprocess Communication Mechanisms

The main functions in the message queue API are the following:

`mq_open()` function creates a new message queue or opens an existing queue, returning a **message queue descriptor** for use in later calls.

`mq_send()` function writes a message to a queue.

`mq_receive()` function reads a message from a queue.

`mq_close()` function closes a message queue that the process previously opened.

`mq_unlink()` function removes a message queue name and marks the queue for deletion when all processes have closed it.

CS240 Operating Systems, Communications and Concurrency

Unix Interprocess Communication Mechanisms

Internal Structure - Each message queue has an associated set of attributes, which are set when it is created. Attributes can be queried using:-

```
mq_getattr()
```

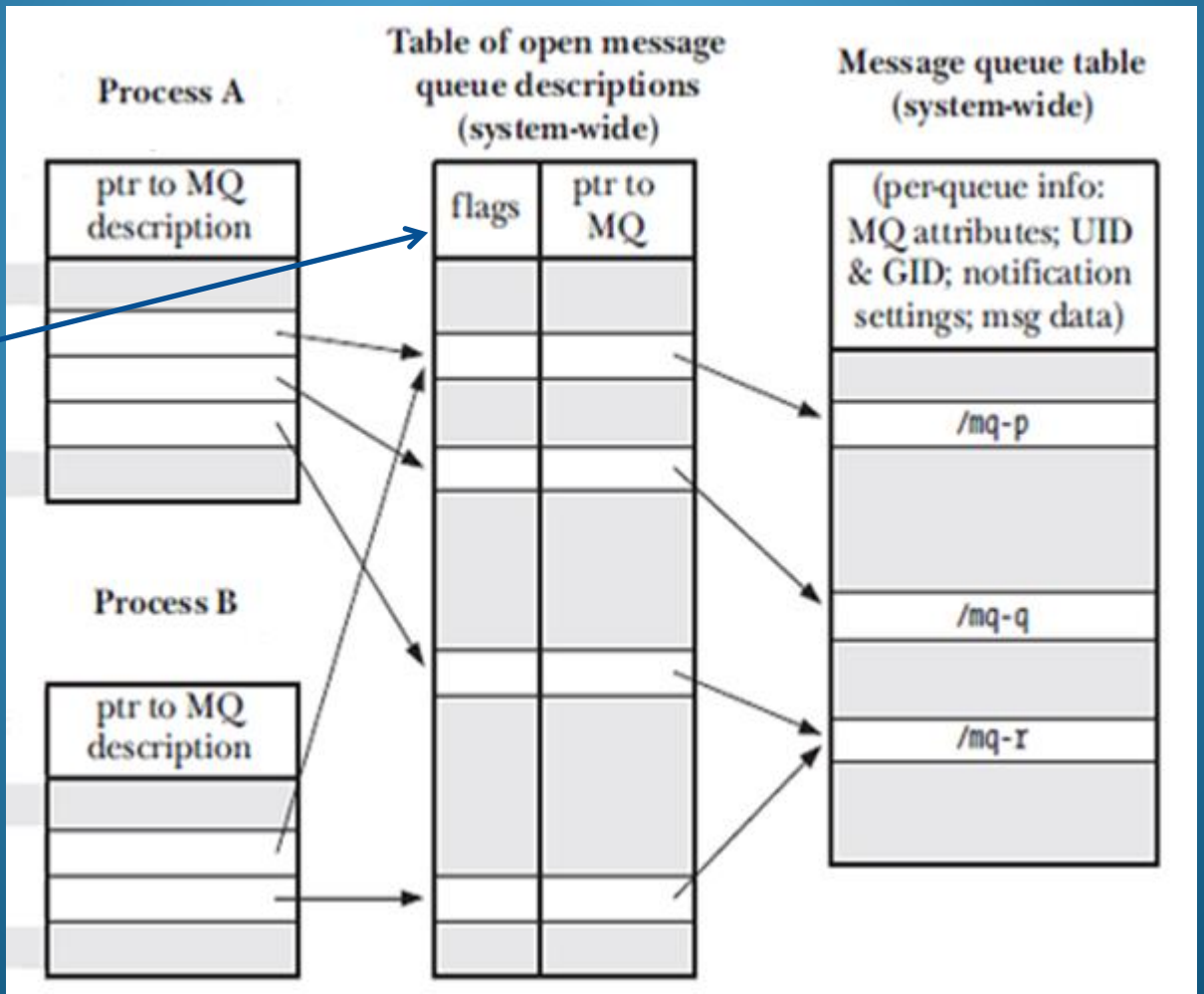
The message queue attribute structure fields are defined as:-

```
struct mq_attr {
   long mq_flags;  /*flags 0 or O_NONBLOCK semantics*/
   long mq_maxmsg; /*Max number of messages on queue*/
   long mq_msgsize; /*Max message size (in bytes)*/
   long mq_curmsgs; /*Num messages currently in queue*/
};
```

CS240 Operating Systems, Communications and Concurrency

Kernel Data Structures and Message Queues

Mode in which message queue is opened by a process

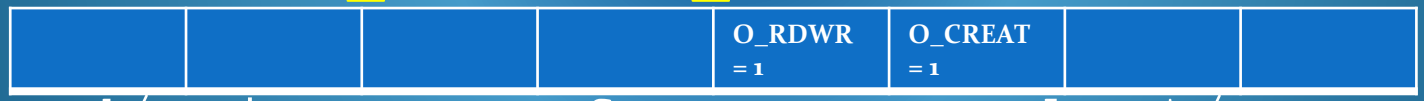


CS240 Operating Systems, Communications and Concurrency

Creating a new message queue

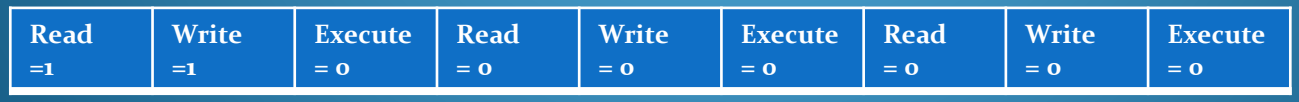
```
strcpy(qname, "/test_queue");

/* the flag O_CREAT is for creating a message
queue if it doesn't exist on opening */
modeflags = O_RDWR | O_CREAT;
```

e.g. modeflags bit mask




```
/* Read/Write perm for owner only */
permissions = 0600;
```

e.g. permissions bitmask




```
attr.mq_flags = 0; /*Blocking semantics mode*/
attr.mq_maxmsg = 10;
attr.mq_msgsize = 512;
```

CS240 Operating Systems, Communications and Concurrency

Creating a message queue

If the message queue doesn't exist, additional parameters are supplied to `mq_open`

```
mqd_t mq;
mq = mq_open(qname, modeflags, permissions,
                                    &attr);
```

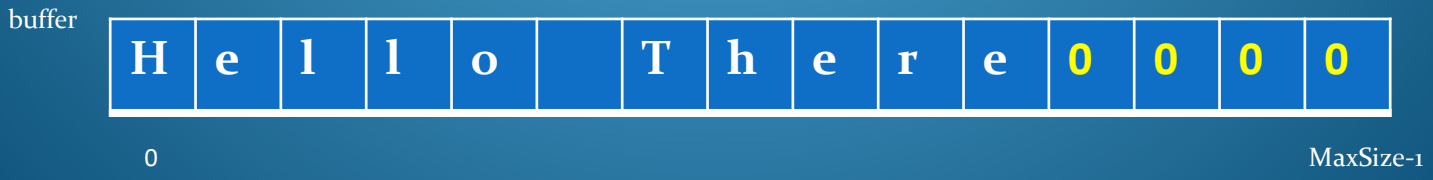
If the message queue already exists it can be opened with

```
modeflags = O_RDWR;
mq = mq_open(qname, modeflags);
```

CS240 Operating Systems, Communications and Concurrency

Sending a message to an opened message queue mq

```
char buffer[max_size];
int priority = 0;

memset(buffer,0, max_size);
strcpy(buffer,"Hello There");
mq_send(mq, buffer, max_size, priority);
```



CS240 Operating Systems, Communications and Concurrency

Reading a message from an opened message queue

```
int numRead;
char buffer[max_size];
int priority;

numRead = mq_receive(mq, buffer, max_size,
&priority);
```

`numRead` is the number of bytes in the received message.

CS240 Operating Systems, Communications and Concurrency

To close access to the message queue for this process

```
mq_close(mq)
```

To request destruction of the message queue entirely

```
mq_unlink(qname);
```

If the requesting process has permission to do this, then the queue is not destroyed until all open descriptors in other processes that are using the queue have been closed.

CS240 Operating Systems, Communications and Concurrency

The use of message queues in C programs requires the following include files:-

```
#include <fcntl.h>
#include <sys/stat.h>
#include <mqueue.h>
```

Programs using the POSIX message queue API must be compiled with `cc` using the `-lrt` option to link with the real-time library, `librt`.

FULL EXAMPLE PROGRAM GIVEN IN LAB HANDOUT

CS240 Operating Systems, Communications and Concurrency

Summary of Unix Interprocess Communication Mechanisms

Pipes –	Fast memory based, stream oriented, between related processes
Named Pipes –	Like pipes but uses shared files, useable by unrelated processes
Sockets -	Low level internet communication abstraction generally over TCP/IP between unrelated processes
Remote Procedure Call –	Simplifies client/server programming
Message Queues –	Asynchronous whole message communication with priority ordering