# Section Outline

- What is IPC
- IPC standards
- Posix IPC Methods
  - Pipes
  - Fifos
  - Message queues
  - Shared memory

# What is IPC

- **Inter-process communication (IPC)** is a set of methods for the exchange of data among multiple threads in one or more processes.
  - Processes may be running on one or more computers connected by a network.
  - IPC methods are divided into methods for **message passing**, **synchronization**, **shared memory**, and **remote procedure calls** (RPC).

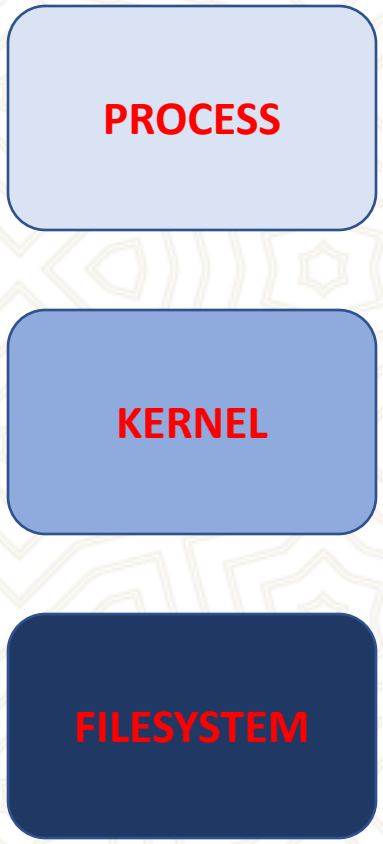
# IPC Methods

- Unix
- System V
- POSIX
- Others
  - Sockets
  - Dbus
  - So on…

# POSIX IPC

- Pipe
- FIFO
- **Signals**
- **Semaphores**
- Message Queues
- Shared Memory

# Persistence of IPC Objects

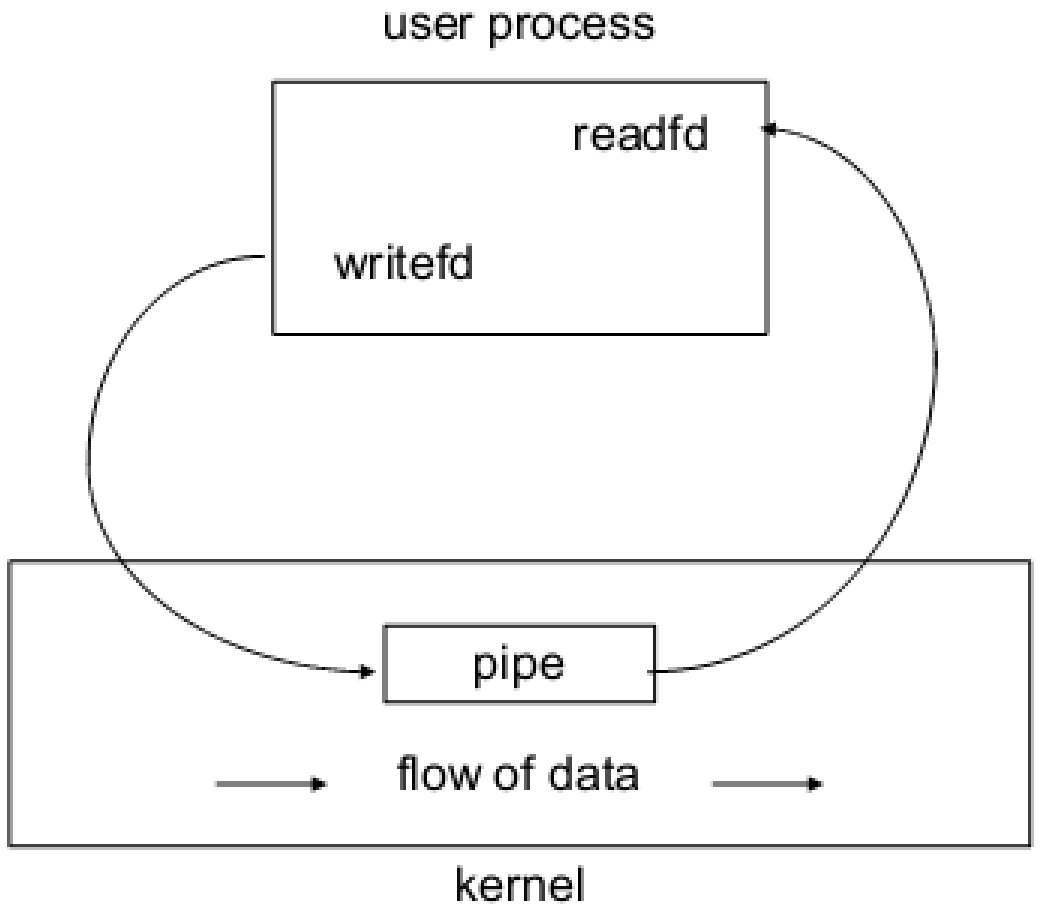



- process-persistent IPC:
  - exists until last process with
  - IPC object closes the object
- kernel-persistent IPC
  - exists until kernel reboots or
  - IPC object is explicitly deleted
- file-system-persistent IPC
  - exists until IPC object is
  - explicitly deleted

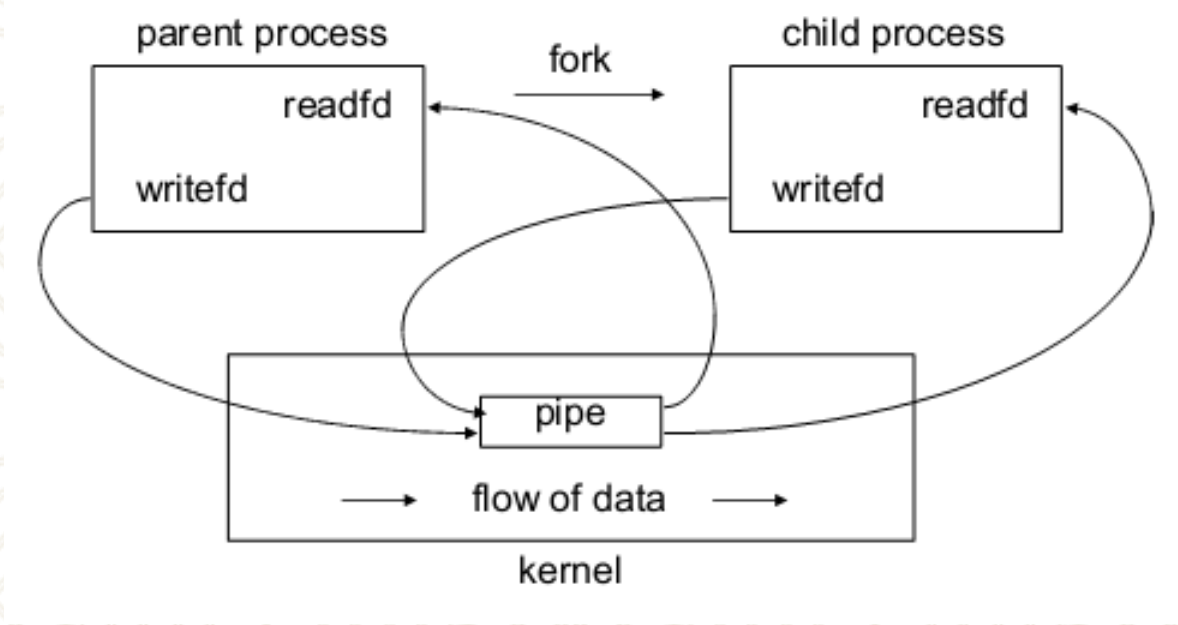
# Pipes

- A pipe provides a one-way flow of data
  - example: who | sort| lpr
- The difference between a file and a pipe:
  - pipe is a data structure in the kernel.
- A pipe is created by using the pipe system call
  - int **pipe** (int* filedes);
  - Two file descriptors are returned
- **filedes[0]** is open for reading
- **filedes[1]** is open for writing
- Typical size is **512 bytes** (Minimum limit defined by POSIX)

# Pipe
# (Single Process)

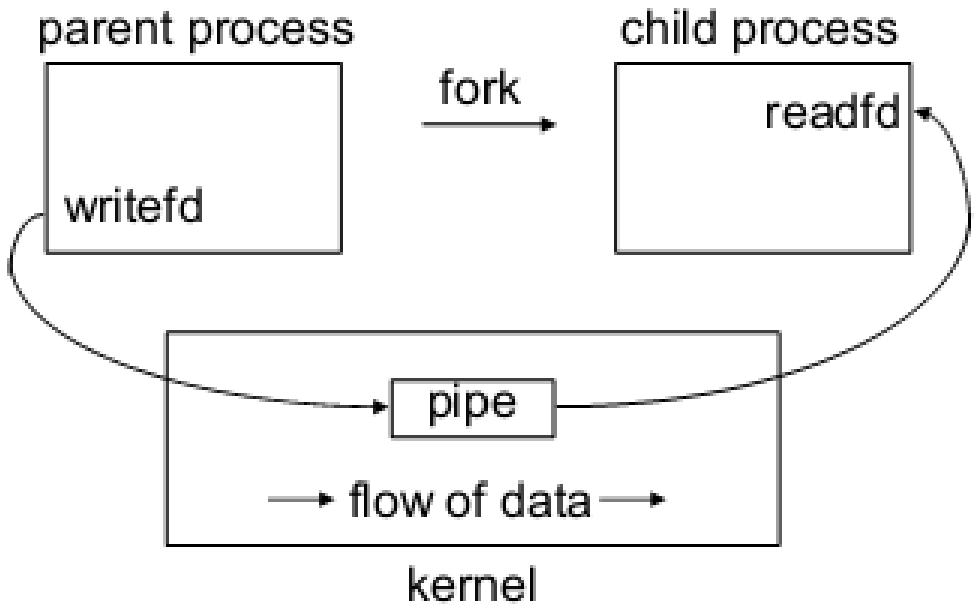


# Pipe
# (Two Process)




**Just after fork**

# Pipe
# (Two Process)




- Parent opens file, child reads file
  - parent closes **read** end of pipe
  - child closes **write** end of pipe

# A simple pipe example

```c
int main (int argc, char *argv[])  {

        int pipe1[2];
        pid_t childpid;

        pipe(pipe1);

        if((childpid=fork())==0)  {// Child
                close(pipe1[1]);
                server(pipe1[0]);
                exit(0);
        }

        close(pipe1[0]);
        client(pipe1[1]);

        waitpid(childpid, NULL,0); // wait for child to terminate
        exit(0);
}
```

```c
void client (int writefd ) {

        size_t len;
        char buff[MAX_LINE];

        fgets(buff, MAX_LINE, stdin);
        len = strlen(buff);
        if(buff[len-1]=='\n')
                len--;
        write(writefd, buff, len);
}
```

```
lucid@ubuntu:~/Downloads$ ./Pipe
hello.txt
This
is
the
content
of
Hello.txt
file
lucid@ubuntu:~/Downloads$
```

- **Ex_1_pipe.c**

# More on Pipes

- FILE ***popen** (const char * command, const char *type)
  - Type is r, the calling process reads the standart output of the command,
  - Type is w, the calling process writes to the standart input of the command
  - Return file * if OK, NULL on error
- int **pclose** (FILE *stream);
  - Closes a standard I/O stream that was created by popen

# FIFOs

- Pipes have no names, they can only be used between processes that have a parent process in common.
- FIFO stands for first-in, first-out
- Similar to a pipe, it is a one-way (half duplex) flow of data
- A FIFO has a pathname associated with it, allowing unrelated processes to access a single FIFO
- FIFOs are also called ***named pipes***

# FIFOs

```
#include <sys/types.h>
#include <sys/stat.h>
```

int **mkfifo** (const char *pathname, mode_t mode)

returns 0 if OK, -1 on error

# FIFO example

```
int main (int argc, char *argv[])  {

        int readfd, writefd;

        if((mkfifo(FIFO1, FIFO_MODE )<0)&&(errno != EEXIST))  {
                printf("can not open %s\n",FIFO1);
                exit(-1);
        }

        if((mkfifo(FIFO2, FIFO_MODE )<0)&&(errno!= EEXIST))  {
                printf("can not open %s\n",FIFO2);
                exit(-1);
        }

        readfd  = open(FIFO1, O_RDONLY);
        writefd = open(FIFO2, O_WRONLY);

        server(readfd, writefd);

        exit(0);
}
```

```
lucid@ubuntu:~/Downloads$ ./Client
enter a file name
hello.txt

sending file name to server

This
is
the
content
of
Hello.txt
file
lucid@ubuntu:~/Downloads$
```

```
lucid@ubuntu:~/Downloads$ ./Server
received file name (hello.txt)
sending contents of the file back to client...
lucid@ubuntu:~/Downloads$
```

- **Ex_2_client.c**
- **Ex_2_server.c**

Example: https://www.geeksforgeeks.org/named-pipe-fifo-example-c-program/

# Message Queues

- **Unlike pipes and FIFOs**, message queues support messages that have **structure**.
- Like FIFOs, message queues are persistent objects that must be initially created and eventually deleted when no longer required.
- Message queues are created with a specified maximum message size and maximum number of messages.
- Message queues are created and opened using a special version of the open system call, **mq_open**.

# POSIX Message Queue Functions

- mq_open()
- mq_close()
- mq_unlink()
- mq_send()
- mq_receive()
- mq_setattr()
- mq_getattr()
- mq_notify()

# mq_open(const char *name, int oflag,...)

- name
  - Must start with a slash and contain no other slashes
  - QNX puts these in the /dev/mqueue directory
- oflag
  - **O_CREAT** – to create a new message queue
  - **O_EXCL** – causes creation to fail if queue exists
  - **O_NONBLOCK** – usual interpretation
- mode – usual interpretation
- &mqattr – address of structure used during creation

# mq_attr structure

- This structure, pointed to by the last argument of **mq_open**, has at least the following members:
  - mq_maxmsg – maximum number of messages that may be stored in the message queue
  - mq_msgsize – the size of each message, in bytes
  - mq_flags – not used by mq_open, but accessed by mq_getattr and mq_setattr
  - mq_curmsgs – number of messages in the queue

# mq_close(mqd_t mqdes)

- This function is used to close a message queue after it has been used.
- As noted earlier, the message queue is not deleted by this call; it is persistent.
- The message queue's contents **are not altered** by **mq_close** unless a prior call(by this or another process) called **mq_unlink** (see next slide). In this respect, an open message queue is just like an open file: deletion is deferred until all open instances are closed.

# mq_unlink(const char *name)

- This call is used to remove a message queue.
- Recall (from the previous slide) that the deletion is deferred until all processes that have the message queue open have closed it (or terminated).
- It is usually a good practice to call mq_unlink immediately after all processes that wish to communicate using the message queue have opened it. In this way, as soon as the last process terminates (closing the message queue), the queue itself is deleted.

# Message Queue Persistence - I

- As noted, a message queue is persistent.
- Unlike a FIFO, however, the contents of a message queue are also persistent.
- It is not necessary for a reader and a writer to have the message queue open at the same time. A writer can open (or create) a queue and write messages to it, then close it and terminate.
- Later a reader can open the queue and read the messages.

# Message Queue Persistence - II

- mkdir /dev/mqueue
- mount -t mqueue none /dev/mqueue
- ls -la /dev/mqueue

```
cihan@sdf-1:~/Desktop> ls -la /dev/mqueue/
total 0
drwxrwxrwt  2 root  root    80 2009-03-18 20:59 .
drwxr-xr-x 14 root  root  4600 2009-03-18 20:52 ..
-rwxr-x---  1 cihan users   80 2009-03-18 19:40 myqueue123
-rwxr-x---  1 cihan users   80 2009-03-18 20:59 test
cihan@sdf-1:~/Desktop>
```

# mq_send(mqd_t mqdes, const char *msq_ptr, size_t msglen, unsigned msg_prio)

- **mqdes**
  - the descriptor required by mq_open
- **msg_ptr**
  - pointer to a char array containing the message
- **msglen**
  - number of bytes in the message; this must be no larger than the maximum message size for the queue
- **prio**
  - the message priority (0..MQ_PRIO_MAX); messages with larger (higher) priority leap ahead of messages with lower (smaller) priority

# mq_receive(mqd_t mqdes, char *msq_ptr, size_t msglen, unsigned *msg_prio)

- **mqdes**
  - the descriptor returned by mq_open
- **msg_ptr**
  - pointer to a char array to receive the message
- **msglen**
  - number of bytes in the msg buffer; this should normally be equal to the maximum message size specified when the message queue was created
- **msg_prio**
  - pointer to a variable that will receive the message's priority
- The call returns the **size of the message, or -1**

# A simple Message Queue Example

## Sender

```
/* forcing specification of "-i" argument */
if (msgprio == 0) {
    printf("Usage: %s [-q] -p msg_prio\n", argv[0]);
    exit(1);
}

/* opening the queue        --  mq_open() */
if (create_queue) {
    msgq_id = mq_open(MSGQOBJ_NAME, O_RDWR | O_CREAT | O_EXCL, S_IRWXU | S_IRWXG, NULL);
} else {
    msgq_id = mq_open(MSGQOBJ_NAME, O_RDWR);
}
if (msgq_id == (mqd_t)-1) {
    perror("In mq_open()");
    exit(1);
}

/* producing the message */
currtime = time(NULL);
snprintf(msgcontent, MAX_MSG_LEN, "Hello from process %u (at %s).", my_pid, ctime(&currtime));

/* sending the message      --  mq_send() */
mq_send(msgq_id, msgcontent, strlen(msgcontent)+1, msgprio);

/* closing the queue        -- mq_close() */
mq_close(msgq_id);
```

- **Ex_4_mq_dropone.c**

```
lucid@ubuntu:~/Downloads$ ./Drop
Usage: ./Drop [-q] -p msg_prio
lucid@ubuntu:~/Downloads$ ./Drop -q -p 11
I (5012) will use priority 11
lucid@ubuntu:~/Downloads$ ./Drop -p 110
I (5015) will use priority 110
lucid@ubuntu:~/Downloads$ ./Drop -p 17
I (5016) will use priority 17
lucid@ubuntu:~/Downloads$
```

# A simple Message Queue Example

## Receiver

```
/* opening the queue        --  mq_open() */
msgq_id = mq_open(MSGQOBJ_NAME, O_RDWR);
if (msgq_id == (mqd_t)-1) {
    perror("In mq_open()");
    exit(1);
}

/* getting the attributes from the queue        --  mq_getattr() */
mq_getattr(msgq_id, &msgq_attr);
printf("Queue \"%s\":\n\t- stores at most %ld messages\n\t\
       - large at most %ld bytes each\n\t- currently holds %ld messages\n",
       MSGQOBJ_NAME, msgq_attr.mq_maxmsg, msgq_attr.mq_msgsize, msgq_attr.mq_curmsgs);

/* getting a message */
msgsz = mq_receive(msgq_id, msgcontent, MAX_MSG_LEN, &sender);
if (msgsz == -1) {
    perror("In mq_receive()");
    exit(1);
}
printf("Received message (%d bytes) from %d: %s\n", msgsz, sender, msgcontent);

/* closing the queue    --  mq_close() */
mq_close(msgq_id);

mq_unlink(MSGQOBJ_NAME);
return 0;
```

```
lucid@ubuntu:~/Downloads$ ./Take
Queue "/test":
        - stores at most 10 messages
                - large at most 8192 bytes each
        - currently holds 3 messages
Received message (56 bytes) from 110: Hello from process 5015 (at Fri Aug  9 07:
34:05 2013
).
```

- **Ex_4_mq_takeone.c**

# The effect of fork on a message queue

- Message queue descriptors **are not (in general) treated as file descriptors**; the unique open, close, and unlink calls should already suggest this.
- Open message queue descriptors **are not inherited by child processes** created by fork.
- Instead, a child process must explicitly open (using mq_open) the message queue itself to obtain a message queue descriptor

# Detecting non-empty queues

- mq_receive on an empty queue **normally causes a process to block**, and this may not be desirable.
- Of course, **O_NONBLOCK** could be applied to the queue to prevent this behavior, but in that case the mq_receive call will return -1, and our only recourse is to try mq_receive again later.
- With the **mq_notify** call we can associate a single process with a message queue so that it (the process) will be notified when the message queue changes state from empty to non-empty

# mq_notify(mqd_t mqdes, const struct sigevent *notification)

- queuefd
  - as usual, to identify the message queue
- sigev
  - a struct sigevent object that identifies the signal to be sent to the process to notify it of the queue state change.
- Once notification has been sent, **the notification mechanism is removed**. That is, to be notified of the next state change (from empty to non-empty), the notification **must be reasserted**.

# Changing the process to be notified

- Only one process can be registered (at a time) to receive notification when a message is added to a previously-empty queue.
- If you wish to change the process that is to be notified, you must remove the notification from the process which is currently associated (call mq_notify with NULL for the sigev argument), and then associate the notification with a different process.

# Attributes

- **mq_getattr** (queuefd,&mqstat)
  - retrieves the set of attributes for a message queue to the struct mq_attr object named mqstat.
  - the mq_flags member of the attributes is not significant during mq_open, but it can be set later
- **mq_setattr** (queuefd,&mqstat,&old)
  - Set (or clear) to O_NONBLOCK flag in the mqattr structure for the identified message queue
  - Retrieve (if old is not NULL) the previously existing message queue attributes
  - Making changes to any other members of the mqattr structure is **ineffective**.

# Timed send and receive

- Two additional functions, mq_timedsend and mq_timedreceive, are like mq_send and mq_receive except they have an additional argument, a pointer to a struct timespec.
- This provides the absolute time at which the send or receive will be aborted if it cannot be completed (because the queue is full or empty, respectively).

# Shared Memory

- Sharing memory in POSIX (and many other systems) requires
  - creating a persistent “object” associated with the shared memory, and
  - allowing processes to connect to the object.
- creating or connecting to the persistent object is done in a manner similar to that for a file, but uses the shm_open system call.

# Shared Memory Functions

- shm_open()
- mmap()
- munmap()
- ftruncate()
- shm_unlink()

# shm_open (name, oflag, mode)

- name is a string identifying an existing shared memory object or a new one (to be created). It should begin with ‘/’, and contain only one slash. In QNX 6, these objects will appear in a special directory.
- mode is the protection mode (e.g. 0644).
- shm_open returns a file descriptor, or –1 in case of error

# shm_open (name, oflag, mode)

- oflag is similar to the flags for files:
  - O_RDONLY – read only
  - O_RDWR – read/write
  - O_CREAT – create a new object if necessary
  - O_EXCL – fail if O_CREAT and object exists
  - O_TRUNC – truncate to zero length if opened R/W

# ftruncate(int fd, off_t len)

- This function (inappropriately named) causes the file referenced by fd to have the size specified by len.
- If the file was previously longer than len bytes, the excess is discarded.
- If the file was previously shorter than len bytes, it is extended by bytes containing zero.

# mmap (void *addr, size_t len, int prot, int flags, int fd, off_t off);

- mmap is used to map a region of the shared memory object (fd) to the process' address space.
- The mapped region has the given len starting at the specified offset off.
- Normally addr is 0, and allows the OS to decide where to map the region. This can be explicitly specified, if necessary.
- mmap returns the mapped address, or –1 on error.(more on next slide)

# mmap, continued

- prot – selected from the available protection settings:
  - PROT_EXEC
  - PROT_NOCACHE
  - PROT_NONE
  - PROT_READ
  - PROT_WRITE
- flags – one or more of the following:
  - MAP_FIXED – interpret addr parameter exactly
  - MAX_PRIVATE – don't share changes to object
  - MAP_SHARED – share changes to object

# munmap (void *addr, size_t len)

- This function removes mappings from the specified address range.
- This is not a frequently-used function, as most processes will map a fixed-sized region and use shm_unlink at the end of execution to destroy the shared memory object (which effectively removes the mappings).

# shm_unlink (char *name);

- This function, much like a regular unlink system call, removes a reference to the shared memory object.
- If the are other outstanding links to the object, the object itself continues to exist.
- If the current link is the last link, then the object is deleted as a result of this call.

# A Simple Shared Memory Example

## Sender

```c
/* creating the shared memory object    --  shm_open()  */
shmfd = shm_open(SHMOBJ_PATH, O_CREAT | O_EXCL | O_RDWR, S_IRWXU | S_IRWXG);
if (shmfd < 0) {
    perror("In shm_open()");
    exit(1);
}
fprintf(stderr, "Created shared memory object %s\n", SHMOBJ_PATH);

/* adjusting mapped file size (make room for the whole segment to map)      --  ftruncate() */
ftruncate(shmfd, shared_seg_size);

/* requesting the shared segment    --  mmap() */
shared_msg = (struct msg_s *)mmap(NULL, shared_seg_size, PROT_READ | PROT_WRITE, MAP_SHARED, shmfd, 0);
if (shared_msg == NULL) {
    perror("In mmap()");
    exit(1);
}
fprintf(stderr, "Shared memory segment allocated correctly (%d bytes).\n", shared_seg_size);

srandom(time(NULL));
/* producing a message on the shared segment */
shared_msg->type = random() % TYPES;
snprintf(shared_msg->content, MAX_MSG_LENGTH, "My message, type %d, num %ld", shared_msg->type, random());
```

- **Ex_5_shm_server.c**

```
lucid@ubuntu:~/Downloads$ ./SHMServer
Created shared memory object /foo1423
Shared memory segment allocated correctly (56 bytes).
lucid@ubuntu:~/Downloads$
```

# A Simple Shared Memory Example

## Receiver

```
/* creating the shared memory object    --  shm_open()  */
shmfd = shm_open(SHMOBJ_PATH, O_RDWR, S_IRWXU | S_IRWXG);
if (shmfd < 0) {
    perror("In shm_open()");
    exit(1);
}
printf("Created shared memory object %s\n", SHMOBJ_PATH);

/* requesting the shared segment    --  mmap() */
shared_msg = (struct msg_s *)mmap(NULL, shared_seg_size, PROT_READ | PROT_WRITE, MAP_SHARED, shmfd, 0);
if (shared_msg == NULL) {
    perror("In mmap()");
    exit(1);
}
printf("Shared memory segment allocated correctly (%d bytes).\n", shared_seg_size);


printf("Message type is %d, content is: %s\n", shared_msg->type, shared_msg->content);
```

- **Ex_5_shm_client.c**

```
lucid@ubuntu:~/Downloads$ ./SHMClient
Created shared memory object /foo1423
Shared memory segment allocated correctly (56 bytes).
Message type is 6, content is: My message, type 6, num 1256344664
lucid@ubuntu:~/Downloads$
```

# References

- http://cs.unomaha.edu/~stanw/091/csci8530/
- http://mij.oltrelinux.com/devel/unixprg/
- Man pages
- man mq_overview
- man mq_open, mq_close etc. etc. etc.
- http://forum.soft32.com/linux2/Utilities-listing-removing-POSIX-IPC-objects-ftopict15659.html