Inter-Process Communications (IPCs): Message Queues, Shared Memory, Semaphores & File Locking

Spring 2025

IPCs (System V)

- Three types of IPCs:
  - Message Queues
  - Shared Memory
  - Semaphores
- Each IPC structure is referred to by a non-negative integer identifier.
  - When an IPC is created, the program responsible for this creation provides a key of type `key_t`.
  - The Operating System converts this key into an IPC identifier.

Keys in the IPC Client-Server Paradigm

⇒ Keys can be created in three ways:

1. The "server" program creates a new structure by specifying a private key that is `IPC_PRIVATE`.
   - Client has to become explicitly aware of this private key.
   - This is often accomplished with the help of a file generated by the server and then looked-up by the client.
2. Server and client do agree on a key value (often defined and hard-coded in the header).
3. Server and client can agree on a pathname to an existing file in the file system `AND` a project-ID (0..255) and then call `ftok()` to convert these two values into a unique key!

Keys

- Keys help identify resources and offer access to the internal structures of the 3 IPC mechanisms (through systems calls):

```
    struct msqid_ds   // for message queues
    struct shmid_ds   // for shared segments
    struct semid_ds   // for semaphores
```

- Wrongly accessing resources returns -1

- Access rights for IPC mechanisms: read/write stored in `struct ipc_perm`

- Included header files:

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

The `ftok()` system call

- converts a pathname and a project identifier to a (System V) IPC-key
-
  ```
      key_t ftok(const char *pathname, int proj_id)
  ```
- Calling the `ftok()`:

```
if ( (thekey=ftok("/tmp/ad.tempfile", 23)) == -1)
    perror("Cannot create key from /tmp/ad.tempfile");
```

- The file `/tmp/ad.tempfile` must be accessible by the invoking process.

Message Queues

- Message queues allow for the exchange of messages between processes.
- The dispatching process sends a specific type of message and the receiving process may request the specific type of message.
- Each message consists of its *“type”* and the *“payload”*.
- Messages are pointers to stuctures:

```
    struct message{
             long type;
             char messagetext[MESSAGESIZE];
         };
```

- Header needed:

```
    #include <sys/msg.h>
```

The system call msgget() - creating/using a queue

```
    int msgget(key_t key, int msgflg)
```

- returns (creates) a message queue identifier associated with the value of the key argument.

- A new message queue is created, if key has the value IPC_PRIVATE.

- If key isn't IPC_PRIVATE and no message queue with the given key exists, the msgflg must be specified to IPC_CREAT (to create the queue).

- If a queue with key key exists and both IPC_CREAT and IPC_EXCL are specified in msgflg, then msgget fails with errno set to EEXIST.
  - IPC_EXCL is used with IPC_CREAT to ensure failure if the segment already exists.

Use-cases of `msgflg`

- Upon creation, the least significant bits of `msgflg` define the permissions of the message queue.
- These permission bits have the same format and semantics as the permissions specified for the mode argument of `open()`.
- The various use-cases of `msgflg` are:

	`PERMS`	`PERMS	IPC_CREAT`	`PERMS	IPC_CREAT	IPC_EXCL`
resource exists	use resource	use resource	error
resource does not exist	error	create and use new resource	create and use new resource

msgsnd() - sending a message to a queue

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

- send msgp (pointer to a record – see below) to message queue with id msqid.

-
  ```
  struct msgbuf {
        long mtype;         /* msg type-must be>0 */
        char mtext[MSGSZ]; /* msg data           */
        };
  ```
- sender must have write-access permission on the message queue to send a message.

msgrcv() – fetching a message from a queue

```
ssize_t msgrcv(int msqid, void *msgp,  size_t msgsz,
               long msgtyp, int msgflg);
```

- receive a message msgp from a message queue with id msqid
- msgtyp is an integer value.
- if msgtyp is zero, the first message is retrieved regardless its type.
  - This value can be used by the receiving process for designating message selection (see below).
- mesgsz specifies the size of the field mtext.
- msgflg is mostly set to 0.

The role of `msgtyp` in `msgrcv()`

`msgtyp` specifies the type of message requested as follows:

- if `msgtyp=0` then the first message in the queue is read.
- if `msgtyp` > 0 then the first message in the queue of type `msgtyp` is read.
- if `msgtyp` < 0 then the first message in the queue with the lowest type value is read.
  - Assume a queue has 3 messages with `mtype` 1, 40, 554 and and `msgtyp` is set to -554; If `msgrcv` is called three times, the messages will be received in the following order: 1, 40, 554.

msgctl() - controling a queue

```
int msgctl(int msqid, int cmd, struct msqid_ds *buf)
```

- performs the control operation specified by cmd on the message queue with identifier msqid
- The msqid_ds structure is defined in <sys/msg.h> as:

```
struct msqid_ds {
    struct ipc_perm msg_perm; /* Ownership and permissions */
    time_t  msg_stime;           /* Time of last msgsnd(2) */
    time_t  msg_rtime;           /* Time of last msgrcv(2) */
    time_t  msg_ctime;           /* Time of last change    */
    unsigned long  __msg_cbytes; /* Current number of bytes
                                    in queue (non-standard)*/
    msgqnum_t msg_qnum;          /* Current number of
                                         messages in queue */
    msglen_t msg_qbytes;         /* Maximum number of bytes
                                          allowed in queue */
    pid_t  msg_lspid;            /* PID of last msgsnd(2)  */
    pid_t  msg_lrpid;            /* PID of last msgrcv(2)  */
};
```

Operating with msgctl() on message queues

Some values for cmd:

- IPC_STAT: Copy information from the kernel data structure associated with msqid into the msqid_ds structure pointed to by buf.
- IPC_SET: Write the values of some members of the msqid_ds structure pointed to by buf to the kernel data structure associated with this message queue, updating also its msg_ctime element.
- IPC_RMID: Immediately remove the message queue, awakening all waiting reader and writer processes (with an error return and errno set to EIDRM).

The server in a message-queue communication

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define MSGSIZE 128
#define PERMS 0666
#define SERVER_MTYPE 27L
#define CLIENT_MTYPE 42L

struct message{
    long mtype;
    char mtext[MSGSIZE];
    };

main(){
    int qid;
    struct message sbuf, rbuf;
    key_t the_key;

    the_key = ftok("/home/ad/SysProMaterial/Set008/src/fileA", 226);

    if ( (qid = msgget(the_key, PERMS | IPC_CREAT)) < 0 ){
        perror("megget"); exit(1);
        }
    printf("Creating message queue with identifier %d \n",qid);
```

The server in a message-queue communication

```
    sbuf.mtype = SERVER_MTYPE;
    strcpy(sbuf.mtext,"A message from server");
    if (msgsnd(qid, &sbuf, strlen(sbuf.mtext)+1, 0) < 0){
        perror("msgsnd"); exit(1);
        }
    printf("Sent message: %s\n",sbuf.mtext);

    if ( msgrcv(qid, &rbuf, MSGSIZE, CLIENT_MTYPE, 0) < 0){
        perror("msgrcv"); exit(1);}
    printf("Received message: %s\n",rbuf.mtext);

    if ( msgrcv(qid, &rbuf, MSGSIZE, CLIENT_MTYPE, 0) < 0){
        perror("msgrcv"); exit(1);}
    printf("Received message: %s\n",rbuf.mtext);

    if (msgctl(qid, IPC_RMID, (struct msqid_ds *)0) < 0){
        perror("msgctl"); exit(1);}
    printf("Removed message queue with identifier %d\n",qid);

}
```

Client (1) in the message-queue communication

```
....
#define MSGSIZE 128
#define PERMS 0666
#define SERVER_MTYPE 27L
#define CLIENT_MTYPE 42L

struct message{
    long mtype;
    char mtext[MSGSIZE]; };

main(){
    int qid; struct message sbuf, rbuf; key_t the_key;

        the_key = ftok("/home/ad/SysProMaterial/Set008/src/fileA", 226);
    if ( (qid = msgget(the_key, PERMS)) < 0 ){
        perror("megget"); exit(1); }
    printf("Accessing message queue with identifier %d \n",qid);
    if ( msgrcv(qid, &rbuf, MSGSIZE, SERVER_MTYPE, 0) < 0){
        perror("msgrcv"); exit(1);}
    printf("Received message: %s\n",rbuf.mtext);
    sbuf.mtype = CLIENT_MTYPE;
    strcpy(sbuf.mtext,"A message from client 1");
    if (msgsnd(qid, &sbuf, strlen(sbuf.mtext)+1, 0) < 0){
        perror("msgsnd"); exit(1);
        }
    printf("Sent message: %s\n",sbuf.mtext);
}
```

Client (2) in the message-queue communication

```
......
#define MSGSIZE 128
#define PERMS 0666
#define SERVER_MTYPE 27L
#define CLIENT_MTYPE 42L

struct message{
    long mtype;
    char mtext[MSGSIZE]; };

main(){
    int qid; struct message sbuf, rbuf; key_t the_key;

        the_key = ftok("/home/ad/SysProMaterial/Set008/src/fileA", 226);
    if ( (qid = msgget(the_key, PERMS)) < 0 ){
        perror("megget"); exit(1); }
    printf("Accessing message queue with identifier %d \n",qid);
    sbuf.mtype = CLIENT_MTYPE;
    strcpy(sbuf.mtext,"A message from client 2");
    if (msgsnd(qid, &sbuf, strlen(sbuf.mtext)+1, 0) < 0){
        perror("msgsnd"); exit(1);
        }
    printf("Sent message: %s\n",sbuf.mtext);
}
```

Running the application

The server:

```
antoulas@sazerac:~/src$ ./msg-server
Creating message queue with identifier 0
Sent message: A message from server
```

Client 1:

```
antoulas@sazerac:~/src$ ./msg-client1
Accessing message queue with identifier 0
Received message: A message from server
Sent message: A message from client 1
antoulas@sazerac:~/src$
```

Server status:

```
antoulas@sazerac:~/src$ ./msg-server
Creating message queue with identifier 0
Sent message: A message from server
Received message: A message from client 1
```

Running the application

Client 2:

```
antoulas@sazerac:~/src$ ./msg-client2
Accessing message queue with identifier 0
Sent message: A message from client 2
antoulas@sazerac:~/src$
```

Server:

```
antoulas@sazerac:~/src$ ./msg-server
Creating message queue with identifier 0
Sent message: A message from server
Received message: A message from client 1
Received message: A message from client 2
Removed message queue with identifier 0
antoulas@sazerac:~/src$
```

Developing a Priority Queue

- Implement a Queue in which Jobs have Priorities
- A server gets the items from the queue and and in some way (pick one) "processes" these items.

q.h

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#include <string.h>
#include <errno.h>

#define QKEY        (key_t) 108
#define QPERM       0660
#define MAXOBN      50
#define MAXPRIOR    10

struct q_entry{
    long mtype;
    char mtext[MAXOBN+1];
    };
```

init_queue.c

```
#include <stdio.h>
#include <stdlib.h>
#include "q.h"

int init_queue(void){
    int queue_id;

    if ( (queue_id = msgget(QKEY, IPC_CREAT | QPERM)) == -1 )
        perror("msgget failed");
    return(queue_id);
}
```

myqueue.c

```c
#include <stdio.h>
#include <stdlib.h>
#include "q.h"

int myenter(char *objname, int priority){
        int len, s_qid;
        struct q_entry s_entry;

        if ( (len=strlen(objname)) > MAXOBN){
                printf("name too long\n"); exit(1); }
        if ( priority > MAXPRIOR || priority < 0 ){
                printf("invalid priority level"); return(-1); }
        if ( (s_qid = init_queue()) == -1 ) return(-1);
        else    printf("Entering Queue with ID: %d \n",s_qid);

        s_entry.mtype= (long)priority;
        strncpy(s_entry.mtext, objname, MAXOBN);

        if (msgsnd(s_qid, &s_entry, len, 0) == -1 ){
                perror("msgsnd failed"); return(-1);}
        else    {
                printf("Object %s With Priority %ld has been Enqueued
                     Successfully \n",\
                        s_entry.mtext, s_entry.mtype);
                return(0);
                }
}
```

myqueue.c

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

        if ( argc!= 3){
                fprintf(stderr,"usage: %s objname priority\n",argv[0]);
                }
        if ((priority = atoi(argv[2])) <=0 || priority > MAXPRIOR){
                printf("invalid priority");
                exit(2);
                }

        if ( myenter(argv[1], priority) < 0 ){
                printf("enter failure");
                exit(3);
                }
        exit(0);
}
```

dequeue.c

```
#include <stdio.h>
#include <stdlib.h>
#include "q.h"

int proc_obj(struct q_entry *msg){
        printf("Retrieved Object with Priority: %ld and Text: %s\n", \
                                msg->mtype, msg->mtext);
        }

int myserve(void){
        int mlen, r_qid;
        struct q_entry r_entry;

        if ( (r_qid=init_queue()) == -1)
                return(-1);
        else    printf("Accessing Queue with ID: %d\n",r_qid);

        for(;;){
                if ( (mlen=msgrcv(r_qid, &r_entry, MAXOBN,
                        (-1 * MAXPRIOR) , MSG_NOERROR) ) == -1 ){
                        perror("mesgrcv failed"); return(-1);
                        }
                else {
                        r_entry.mtext[mlen]=’\0’;
                        proc_obj(&r_entry);
                        }
                }
}
```

dequeue.c

```
main(){
        pid_t pid;

        switch (pid=fork()){
        case 0: // child
                myserve();
                break;
        case -1:
                printf("fork to start the server failed");
                break;
        default:
                printf("server process pid is %d \n", pid);
        }
exit(pid != 1 ? 0 : 1);
}
```

Running the priority queue program(s)

```
antoulas@sazerac:~/PriorityQueue$ ./enqueue object123 2
Entering Queue with ID: 262144
Object object123 With Priority 2 has been Enqueued Successfully
antoulas@sazerac:~/PriorityQueue$ ./enqueue object111 5
Entering Queue with ID: 262144
Object object111 With Priority 5 has been Enqueued Successfully
antoulas@sazerac:~/PriorityQueue$ ./enqueue object133 4
Entering Queue with ID: 262144
Object object133 With Priority 4 has been Enqueued Successfully
antoulas@sazerac:~/PriorityQueue$ ./enqueue object321 9
Entering Queue with ID: 262144
Object object321 With Priority 9 has been Enqueued Successfully
antoulas@sazerac:~/PriorityQueue$ ./enqueue object311 7
Entering Queue with ID: 262144
Object object311 With Priority 7 has been Enqueued Successfully
antoulas@sazerac:~/PriorityQueue$ ./dequeue
server process pid is 4569
Accessing Queue with ID: 262144
Retrieved Object with Priority: 2 and Text: object123
Retrieved Object with Priority: 4 and Text: object133
Retrieved Object with Priority: 5 and Text: object111
Retrieved Object with Priority: 7 and Text: object311
Retrieved Object with Priority: 9 and Text: object321
antoulas@sazerac:~/PriorityQueue$ ./dequeue
server process pid is 4571
Accessing Queue with ID: 262144
antoulas@sazerac:~/PriorityQueue$
```