Inter-Process Communication

IPC in unix systems

3] Message Queues (SysV methods)

Scope of Labs

- Get familiar with Unix environment
- Know the importance of IPC
- Know how to implement and use different IPC techniques

Today Objectives

- Review on Child-Parent communication
- Review on Signals
- Introduction to System V communication methods
- Learn about Message queues & how to handle them

Review

- Programs need to communicate with each other for variable reasons (i.e. requesting functionality , sending results, Controlling… etc) & they do so using the Operating system as a medium
- In Unix system , it adopts the concept of child parent processes to keep control of child process
- Signals are sort of SW interrupts that are used to handle special functionalities

Sys V communication methods

- SySV communication methods were first created in one of UNIX predecessors called system V and then was ported to almost all Unix/Linux flavors

- They can be summarized into
  - SYSV style message queues
  - SYSV style shared memory segments
  - SYSV style semaphore sets

- Other IPC exist but we will talk today for this lab & the next one about SYSV methods

Sys V communication methods ….

- There are some common points that are found in all these three types
  - ***IPC key*****:** each instance of an IPC type has a unique key and this is used to identify it among others.
  - ***IPC get*****:** every IPC type has a create function called “get” which can be used to create a new instance or retrieve an existing one to be used.
  - ***IPC control*** **:** used to control the settings, this control includes retrieving the status, changing parameters of, or even deleting the IPC instance.
  - ***IPC operations*****:** Each type of IPC has its own type of operations that applies to that type.
- We will see examples soon enough don’t worry now ☺

Message Queues

- Message queues can be best described as an internal linked list within the kernel's addressing space. Messages can be sent to the queue in order and retrieved from the queue in several different ways. Each message queue is uniquely identified by an IPC identifier.

Creating and getting Message Queues
IPC get operation

message queue identifier on success , -1 on failure

Function used to get existing msg queue or create a new one

```
int msgget ( key_t key, int msgflg );  /*IPC get operation*/
```

Key (**IPC Key**) *: holds the key for the msg queue if one already exist , if the key doesn't exist or Key = IPC_PRIVATE* it creates a new one

Msgflg : holds certain flags for doing different operations

i.e. 1.msgflg = IPC_CREAT , it either creates a new msg queue & return its identifier (if Key = IPC_Private or doesn't exist) or returns the identifier for an existing queue with the same key value

i.e.2. msgflg = IPC_CREAT|IPC_EXCL , then either a new queue is created, or if the queue exists, the call fails with -1

Sending and Receiving Messages
IPC operation

Return 0 on success

Function used to send msgs on queue

```
int msgsnd ( int msqid, struct msgbuf *msgp, int msgsz, int msgflg );
                /*IPC operations */
```

msgqid : msg queue identifier (the one we get from msgget())

msgp : a pointer to the message itself (instance of my_msgbuffer)

msgsz : message size in bytes excluding the length of the message type (4 byte long).

msgflg : a flag to do different functions (i.e. = 0 , ignore flag)
= IPC_NOWAIT , If set, it checks whether the message queue is full, then the message is not written to the queue, and control is returned to the calling process. If not specified, then the calling process will suspend (block) until the message can be written.

Sending and Receiving Messages …… *IPC operation*

Number of bytes copied into message buffer

Function used to receive msgs from the queue

```
int msgrcv ( int msqid, struct msgbuf *msgp, int msgsz, long mtype,int msgflg );
                    /*IPC operations */
```

msgqid : msg queue identifier (the one we get from msgget())

msgp : a ptr to variable to get the message in it

msgsz : same as before

mtype : specifiy the type of the message I want to receive .

msgflg : a flag to do different functions (i.e. *= IPC_NOWAIT , If set, it checks whether the message queue is empty, then the control is returned to the calling process. If not specified, then the calling process will suspend (block) until it receives the message.*

Message Structure

- Each message should be saved in a ***msgbuf*** structure. This particular data structure can be thought of as a *template* for message data

```
/* message buffer for msgsnd a
struct msgbuf {
        long mtype;       /* type of message */
       char mtext[100];     /* message text */
};
```

The message type, represented as a positive number. (i.e. error msg 1 , request 2 , page not found 404 , ….etc)

The message data itself

Message Structure …..

- This structure ***msgbuf*** can get redefined by the application programmer. Consider this redefinition:

```
struct my_msgbuf {
    long    mtype;        /* Message type */
    long    request_id;    /* Request identifier */
    struct  client info;   /* Client information structure */
};
```

Sending and Receiving Messages ……

- On receiving : The kernel will search the queue for the oldest message having a matching type for mtype parameter, and will return a copy of it in the address pointed to by the *msgp* argument.
- One special case exists. If the *mtype* argument is passed with a value of zero, then the oldest message on the queue is returned, regardless of type.

Controlling a message queue
IPC control operation

Return 0 on success

Function used to control msg queue

```
int msgctl ( int msqid, int cmd, struct msqid_ds *buf );
            /*IPC control operations */
```

msgqid : msg queue identifier (the one we get from msgget())

cmd : command sent to do a control operation on the queue (will see examples after 2 slides)

msqid_ds : a pointer to a structure which save some info about the msg queue

Controlling a message queue …
IPC control operation

The msqid_ds data structure is defined in <sys/msg.h> as follows:

"Every message queue created is associated with similar structure carrying its information"

```
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  */
  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) */
        };
```

Controlling a message queue …
IPC control operation

```
struct ipc_perm {
        key_t          __key;      /* Key supplied to msgget(2) */
        uid_t          uid;        /* Effective UID of owner */
        gid_t          gid;        /* Effective GID of owner */
        uid_t          cuid;       /* Effective UID of creator */
        gid_t          cgid;       /* Effective GID of creator */
        unsigned short mode;       /* Permissions */
        unsigned short __seq;      /* Sequence number */
};
```

Controlling a message queue … … ***IPC control operation***

- Possible values of *cmd* are:
  - *IPC_STAT* : Retrieves the message queue data structure, and stores it in the address pointed to by buf .
  - *IPC_SET*: Write the values of the msqid_ds structure pointed to by buff to the message queue data structure. (this command can set the permissions of the ms_queue)
  - *IPC_RMID*: Removes the queue from the kernel.

Time to try something

- Open ipc_msg.c
- Compile & run ipc_msg.c
- Can anyone tell what happened ?!
- What does ipcs and ipcrm do ?

Any Questions ?!

Message Structure

```
/* message buffer for msgsnd and msgrcv calls */
struct my_msgbuf {
        long mtype;       /* type of message */
        long    request_id;     /* Request identifier */
        struct  client info;    /* Client information structure */
};
```

Message size