Responsible lecturer: Illés, Zoltán associate professor
Media & Educational Informatics Department

Message queues

Goal: We are going to learn about message queues and their possibilities. There are two different implementation (System_V, POSIX) of message queues but both of them are some type of FIFO-s. One can send a notification when a data arrives, the other can handle the queue as several parallel queues marked by a long integer.

We learn: *ftok – key generator (include sys/types.h, sys/ipc.h); IPC_PRIVATE – constant to create a new queue; msgget – create a message queue System_V (include sys/types.h, sys/ipc.h, sys/msg.h); msgsnd, msgrcv – operations with message queue; mq_open – create a message queue (include fcntl.h, sys/stat.h, mqueue.h); mg_send, mg_recieve – operations with message queue; notify – ask for notification at data arriving; mg_close, msgctl – to close, delete message queue; ipcs, ipcrm – to check and remove message queue*

Tasks

1. Write a C program which sends a Hello message text to child process! Use System_V message queue for implementation. (*System_V implementation is an older one so it works everywhere! POSIX implementation is a newer one it is quicker! You can send a message filling a message structure. The first field must be a long type one. It will identify the destination to whom the message is sent.)*

```
int msgget(key_t key, int msgflg);
//key – the result of ftok function or IPC_PRIVATE
//msgflg – S_IRUSR, etc, IPC_CREAT – creates a new messagequeue
//result – mg identifier or -1 at error

int msgsnd(int msqid, const void *msgp, size_t msgsz, int msgflg);
ssize_t msgrcv(int msqid, void *msgp, size_t msgsz, long msgtyp,
                    int msgflg);
//msgid – id of messagequeues, pointer to message, msgsz – length of message
//msgflg – IPC_NOMWAIT – does not wait for a message, returns immediately if
empty
int msgctl(int msqid, int cmd, struct msqid_ds *buf);
//cmd – IPC_RMID – delete, IPC_SET – set values, IPC_GET – get values
//mgsp – pointer to the message NULL at deletion

key_t ftok(const char *pathname, int proj_id);
//pathname – an existing file name, proj_id – integer with 8 bit
//result a key

struct mess {
     long mtype;// value for classifying messages
     char mtext [ 1024 ]; //message contains maximum 1024 characters
};

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 (nonstandard) */
               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) */
          };
```

Financed from the financial support ELTE won from the Higher Education Restructuring Fund of the Hungarian Government

2. Modify the program and „forget" about deletion! (In command line write command ipcs. It will list out each not deleted System_V ipc communication possibility – now message queue! You have to use ipcrm command to delete it!)

3. Modify the original program and child process should wait to mtype 0! *(What happens? 0 means, that each message is read by the reciever. It does not matter which value was given by the sender. There is only one queue!)*

4. Write a chat program in which parent and child send messages to each other with the help of a message queue (System_V). *(To avoid reading your own message back, use two different values for mtype – one for the parent and one for the child. These values may be the PID numbers!)*

5. Modify your chat program and use POSIX implementation of message queue! (Remember the name of the queue must start with character / ! (*To compile the POSIX mqueue you have to use –lrt option! Be careful, because in this implementation the queue is a single queue and not a parallel on. So you have to make synchronization using up e.g. signals or use two queues!)*

```
mqd_t mq_open(const char *name, int oflag);
mqd_t mq_open(const char *name, int oflag, mode_t mode,
                    struct mq_attr *attr);
//name – name of mqueue (start with /),oflag – O_RDWR, etc., mode – S_IRUSR etc.

int mq_send(mqd_t mqdes, const char *msg_ptr,
                    size_t msg_len, unsigned int msg_prio);
//mqdes – descriptor from mq_open, msg_ptr – pointer to the message, msg_len –
length of message, msg_prio – message priority
ssize_t mq_receive(mqd_t mqdes, char *msg_ptr,
                          size_t msg_len, unsigned int *msg_prio);
//the same as mq_send

int mq_close(mqd_t mqdes);
//mq_des – the descriptor of the queue
```

6. Write a program in which both parent and child generate random numbers! The child waits for a while and then sends the numbers through a POSIX queue. The parent asks for a notification if data is in the queue otherwise it is working on its own task and writes numbers to the screen! After the notification it stops to generate numbers, reads the numbers from the queue. (*Be careful, the notification is sent only in that case if the queue was entirely empty!)*

```
int mq_notify(mqd_t mqdes, const struct sigevent *sevp);
//mgdes – messagequeue descriptor, sevp – pointer to sigevent see below

union sigval {          /* Data passed with notification */
           int     sival_int;         /* Integer value */
           void   *sival_ptr;         /* Pointer value */
       };

struct sigevent {
        int          sigev_notify; /* Notification method */
        int          sigev_signo;  /* Notification signal */
        union sigval sigev_value;  /* Data passed with
                                      notification */
        void       (*sigev_notify_function) (union sigval);
                     /* Function used for thread
                        notification (SIGEV_THREAD) */
        void        *sigev_notify_attributes;
                     /* Attributes for notification thread
                        (SIGEV_THREAD) */
        pid_t        sigev_notify_thread_id;
                     /* ID of thread to signal (SIGEV_THREAD_ID) */
    };
```