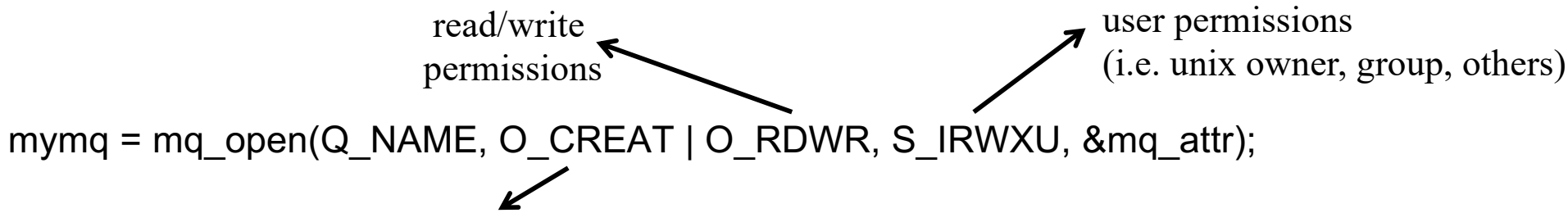
POSIX.4 Message Queue

- A POSIX.4 message queue is a **priority queue** of discrete messages
- It allows two or more processes to communicate and provides messages' prioritization.
- Within a process, shared variables with mutex is an efficient way to communicate
- What about communication among different processes? Different processes can communicate with message queues. Their use is easier than shared memory regions and avoids to use mutex across processes' boundary.

Creating/opening a message queue

```
#include  <mqueue.h>
#define  Q_NAME  ``/CS431queue''  // the message queue name should start with '/'
 struct mq_attr    mq_attr;
 mqd_t   mymq;
//set up the message queue attributes
memset(&mq_attr, 0, sizeof(mq_attr)); //clear up any default settings
mq_attr.mq_maxmsg = 10;     // maximum number of messages stored in the message queue
mq_attr.mq_msgsize = 128;    // size of messages expressed as number of bytes
```

Creating/opening a message queue



The system will create the message queue if it does not already exist.
Be aware that the system will not return any error if the queue existed already!
If **O_CREAT | O_EXCL** are used, the system will return an error if the queue already exists

- The queue_name must be constructed like a normal file pathname, **BUT:**
- To be portable across different platforms, the queue_name should start with ``/''
- To be portable across different platforms, the queue_name should contain no other ``/'' characters

Creating/opening a message queue

```
#include  <mqueue.h>

//create a message queue object
 mymq = mq_open(Q_NAME, O_CREAT | O_RDWR, S_IRWXU, &mq_attr);
```

- oflag ➔ **O_CREAT** causes the message queue object to be created if necessary (see man pages for additional details)
- oflag ➔ you can set flags for read/write permissions (see man pages for additional details)
- mode ➔ it allows to set user permissions (i.e. unix owner, group, others) (see man pages for additional details)

- Remember: **mq_open** operates on a name that may not exist in the file system. So, message queues may not show up in the output of a command like ***ls***.

Example with Message Queues

```
//program1: user interface, create a message queue and send waveform spec
void main(void)
{...
long buffer[4];
memset(buffer, 0, sizeof(buffer)); //clean up the buffer space

//buffer[0] for msg seq #, buffer[1] for min_volt, buffer[2] for max_volt, buffer[3] for freq

struct mq_attr    wave_q_attr
//set up the message queue attributes

memset(&wave_q_attr, 0, sizeof(wave_q_attr));//clear up any default settings
wave_q_attr.mq_maxmsg = 10;
wave_q_attr.mq_msgsize = sizeof(buffer);

...
(to be continued)
```

Example: Sending Messages

It tells the system not to suspend when you try to receive on an empty queue or send to a queue where there is no space!

```
(continue)
//WAVE_SPEC_Q  is our name for the message queue. It is GLOBAL across all processes.
mqd_t  wave_q_id;
wave_q_id = mq_open("/WAVE_SPEC_Q", O_WRONLY | O_CREAT | O_NONBLOCK, 0600,
      &wave_q_attr);
      // 600  =  queue permission, same as file permission:  owner(rw-), group(- - -), others(- - -)
while(command != 'q' ) {
 //ask user if (s)he wants to give new spec or quit
 // if 'q' is received, set the message sequence number to -1, otherwise, increment it by 1
  //get new waveform spec data from user and put the waveform spec  into the buffer
  buffer_length = sizeof(buffer);
  status = mq_send(wave_q_id, (char *) buffer, buffer_length, prio);
  //prio = the priority assigned to msg,  a rule of thumb is to use the sender thread priority.
  //Real-time messages are prioritized. Not FIFO.  Same priority msg are kept in FIFO order
}

 //clean up
  status = mq_close(wave_q_id);    //like closing a file, the message queue STILL lives in the
      system
  status = mq_unlink("/WAVE_SPEC_Q");   //destroy the message queue after everyone closes
      the queue
}
```

MQ_PRIO_MAX
➔ highest priority

0➔ lowest priority

Example: Receiving Messages

```
//program 2: create a receiving queue for waveform spec msg, and display the wave form
void main(void)
{       long buffer[4];
        struct mq_attr    wave_q_attr
        //set up the  same message queue attribute
        memset(&wave_q_attr, 0, sizeof(wave_q_attr));
        wave_q_attr.mq_maxmsg = 10;
        wave_q_attr.mq_msgsize = sizeof(buffer);
        ...
        //set  up the buffer to store the waveform spec,
        //buffer[0] = msg seq #, buffer[1] = min_volt, buffer[2] = max_volt, buffer[3] = freq
        memset(buffer, 0, sizeof(buffer)); //clean up the buffer space
        mqd_t  wave_q_id;
        wave_q_id = mq_open("/WAVE_SPEC_Q", O_RDONLY | O_CREAT | O_NONBLOCK,
                            0600, &wave_q_attr);
        while (buffer[0]  != -1) {     // execute the loop if the message seq # != -1
             buffer_length = sizeof(buffer);
             status = mq_receive(wave_q_id, (char *) buffer, buffer_length, &prior);
             //& prior: priority of the message that you just read.
             //check message sequence number, get the wave form spec from buffer
             //use posix timer to periodically send out the voltages to display the waveform
        }      }
```

It informs the system how large is the **receiver buffer!**

State of Message Queues

- Life span of a message queue
  - Once a message queue is created, like a file, its life is longer than the life of the program that creates it, unless it is unlinked.
  - This feature permits debugging after a program fails and allows for different programs to read and write to shared message queues.
  - Unlike a file, a message queue will disappear if the system reboots.
  - Message queues are inherited when a process **forks**. In addition, when a process calls **exit** or **exec**, message queues are closed implicitly.

- Quiz: in our previous example, what would happen if we use cntr_c, instead of using the “q”, to terminate the program, and then restart the program again?

POSIX Example: mq_master.c

```
#include <signal.h>
#include <time.h>
#include <fcntl.h>    /* For O_* constants */
#include <sys/stat.h> /* For mode constants */
#include <mqueue.h>
#include <sched.h>
#include <stdio.h>
#include <stdlib.h>

#define MY_MQ "/my_mq"

sigset_t signal_set;
```

```
void init_timer(long period)
{
   timer_t timer;
   struct itimerspec timer_spec;

   // Create a timer.
   timer_create(CLOCK_REALTIME, NULL, &timer);

   // Specify the timer's period.
   timer_spec.it_value.tv_sec = 0;
   timer_spec.it_value.tv_nsec = 1; /* Start immediately */
   timer_spec.it_interval.tv_sec = 0;
   timer_spec.it_interval.tv_nsec = period;

   // Setup a signal set for sigwait() to wait for SIGALRM
   sigemptyset(&signal_set);
   sigaddset(&signal_set, SIGALRM);
   sigprocmask(SIG_BLOCK, &signal_set, NULL);

   // Setup the timer's period.
   timer_settime(timer, 0, &timer_spec, NULL);
}
```

POSIX Example: mq_master.c

```
/* ****** MASTER TASK *******/

int main(void)
{
    mqd_t queue;
    struct mq_attr queue_attr;

    // Open the message queue read-only and non-
       blocking.
    queue_attr.mq_maxmsg = 10;
    queue_attr.mq_msgsize = sizeof(unsigned long);
    queue = mq_open(MY_MQ, O_RDONLY |
       O_CREAT | O_NONBLOCK, 0600, &queue_attr);
    // Setup scheduler
    struct sched_param spar;
    spar.sched_priority =
       sched_get_priority_max(SCHED_FIFO);
    sched_setscheduler(0, SCHED_FIFO, &spar);
    // Setup timer
    init_timer(20 * 1000000);
```

```
while (1)
    {
       unsigned long mesg;
       sigwaitinfo(&signal_set, NULL);
       printf("--- New period ---\n");
       while (mq_receive(queue, (char *)&mesg,
       sizeof(mesg), NULL) >= 0)
       {
           printf("Mesg: [%ld]\n", mesg);
       }
       printf("-----------------\n");
    }

    return EXIT_SUCCESS;
}
```

POSIX Example: mq_slave.c

```
#include <signal.h>
#include <time.h>
#include <fcntl.h>    /* For O_* constants */
#include <sys/stat.h> /* For mode constants */
#include <mqueue.h>
#include <sched.h>
#include <stdio.h>
#include <errno.h>
#include <stdlib.h>

#define MY_MQ "/my_mq"
#define SLAVE_PRIORITY 1
#define SLAVE_PERIOD 10

sigset_t signal_set;
```

```
void init_timer(long period)
{
   timer_t timer;
   struct itimerspec timer_spec;

   // Create a timer.
   timer_create(CLOCK_REALTIME, NULL, &timer);

   // Specify the timer's period.
   timer_spec.it_value.tv_sec = 0;
   timer_spec.it_value.tv_nsec = 1; /* Start immediately */
   timer_spec.it_interval.tv_sec = 0;
   timer_spec.it_interval.tv_nsec = period;

   // Setup a signal set for sigwait() to wait for SIGALRM
   sigemptyset(&signal_set);
   sigaddset(&signal_set, SIGALRM);
   sigprocmask(SIG_BLOCK, &signal_set, NULL);

   // Setup the timer's period.
   timer_settime(timer, 0, &timer_spec, NULL);
}
```

POSIX Example: mq_slave.c

```
/* ****** SLAVE TASK *******/
int main(void)
{
   mqd_t queue;
   queue = mq_open(MY_MQ, O_WRONLY);
   // Setup scheduler
   struct sched_param spar;
   spar.sched_priority =
     sched_get_priority_min(SCHED_FIFO) +
     SLAVE_PRIORITY;
   if(sched_setscheduler(0, SCHED_FIFO, &spar) < 0)
   {
      printf("Unable to set SCHED_FIFO. %d\n", errno);
      exit(EXIT_FAILURE);
   }
   // Setup timer
   init_timer(SLAVE_PERIOD * 1000000);
   unsigned long mesg = SLAVE_PRIORITY;
   while (1)
      {
        sigwaitinfo(&signal_set, NULL);
        if (mq_send(queue, (char *)&mesg,
        sizeof(mesg), SLAVE_PRIORITY) < 0)
        {
            printf("Unable to send. %d\n", errno);
        }
      }

   return EXIT_SUCCESS;
}
/**** END - SLAVE TASK *** */
```

POSIX Example: mq_slave2.c

```
#include <signal.h>
#include <time.h>
#include <fcntl.h>    /* For O_* constants */
#include <sys/stat.h> /* For mode constants */
#include <mqueue.h>
#include <sched.h>
#include <stdio.h>
#include <errno.h>
#include <stdlib.h>

#define MY_MQ "/my_mq"
#define SLAVE_PRIORITY 2
#define SLAVE_PERIOD 10

sigset_t signal_set;
```

```
void init_timer(long period)
{
   timer_t timer;
   struct itimerspec timer_spec;

   // Create a timer.
   timer_create(CLOCK_REALTIME, NULL, &timer);

   // Specify the timer's period.
   timer_spec.it_value.tv_sec = 0;
   timer_spec.it_value.tv_nsec = 1; /* Start immediately */
   timer_spec.it_interval.tv_sec = 0;
   timer_spec.it_interval.tv_nsec = period;

   // Setup a signal set for sigwait() to wait for SIGALRM
   sigemptyset(&signal_set);
   sigaddset(&signal_set, SIGALRM);
   sigprocmask(SIG_BLOCK, &signal_set, NULL);

   // Setup the timer's period.
   timer_settime(timer, 0, &timer_spec, NULL);
}
```

POSIX Example: mq_slave2.c

```
/* ****** SLAVE TASK *******/
int main(void)
{
    mqd_t queue;
    queue = mq_open(MY_MQ, O_WRONLY);
    // Setup scheduler
    struct sched_param spar;
    spar.sched_priority =
      sched_get_priority_min(SCHED_FIFO) +
      SLAVE_PRIORITY;
    sched_setscheduler(0, SCHED_FIFO, &spar);
    // Setup timer
    init_timer(SLAVE_PERIOD * 1000000);
    unsigned long mesg = SLAVE_PRIORITY;
    while (1)
      {
        sigwaitinfo(&signal_set, NULL);
        if (mq_send(queue, (char *)&mesg,
        sizeof(mesg), SLAVE_PRIORITY) < 0)
        {
            printf("Unable to send. %d\n", errno);
        }
      }

    return EXIT_SUCCESS;
}
/**** END - SLAVE TASK *** */
```

POSIX Example: run