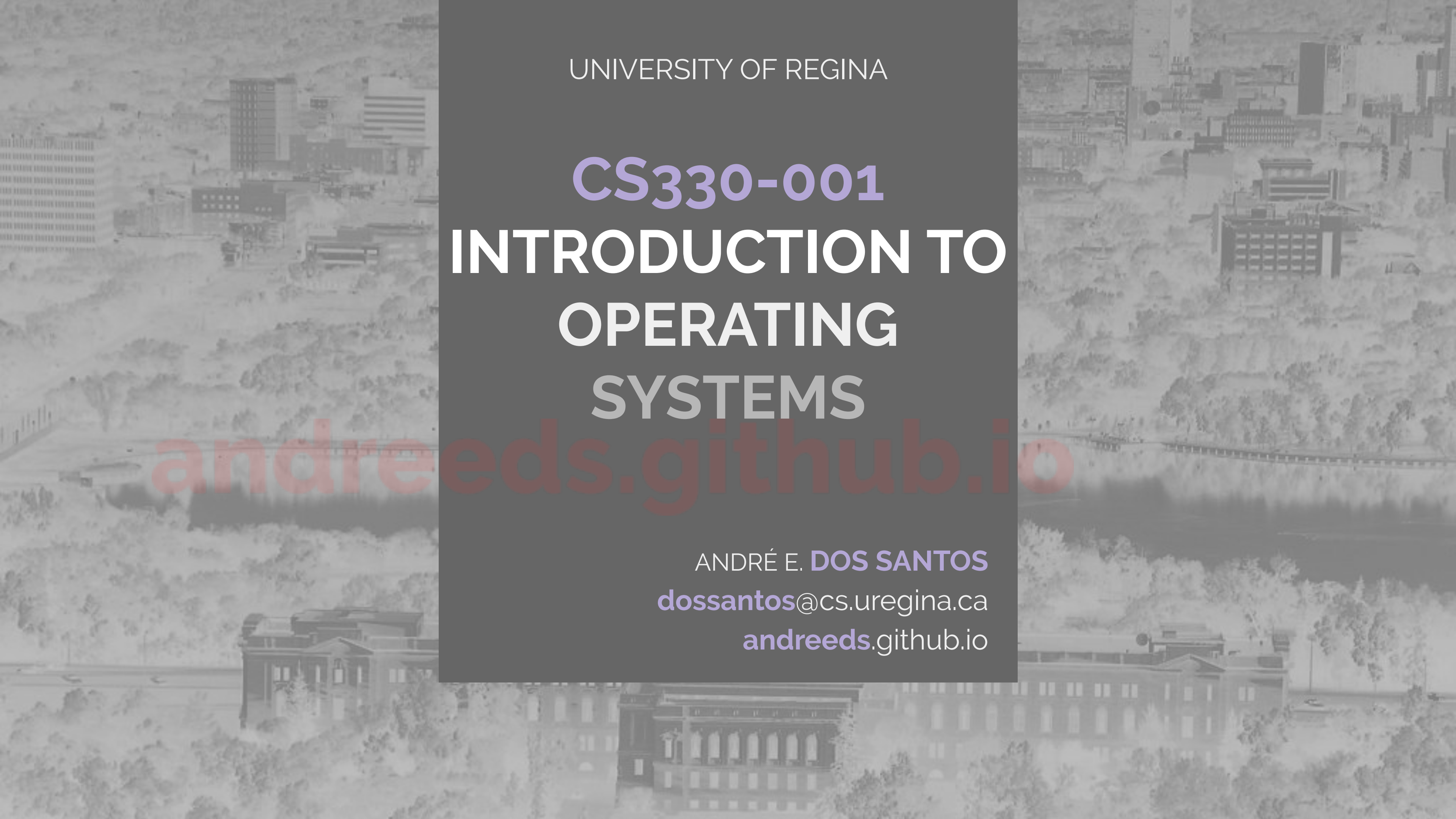


UNIVERSITY OF REGINA

CS330-001 INTRODUCTION TO OPERATING SYSTEMS

ANDRÉ E. DOS SANTOS
dossantos@cs.uregina.ca
andreeds.github.io

CS330-001
INTRODUCTION TO
OPERATING SYSTEMS

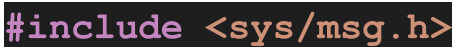
PROGRAMMING MESSAGE QUEUES

C

ANDRÉ E. **DOS SANTOS**
dossantos@cs.uregina.ca
andreeds.github.io

DATA STRUCTURES

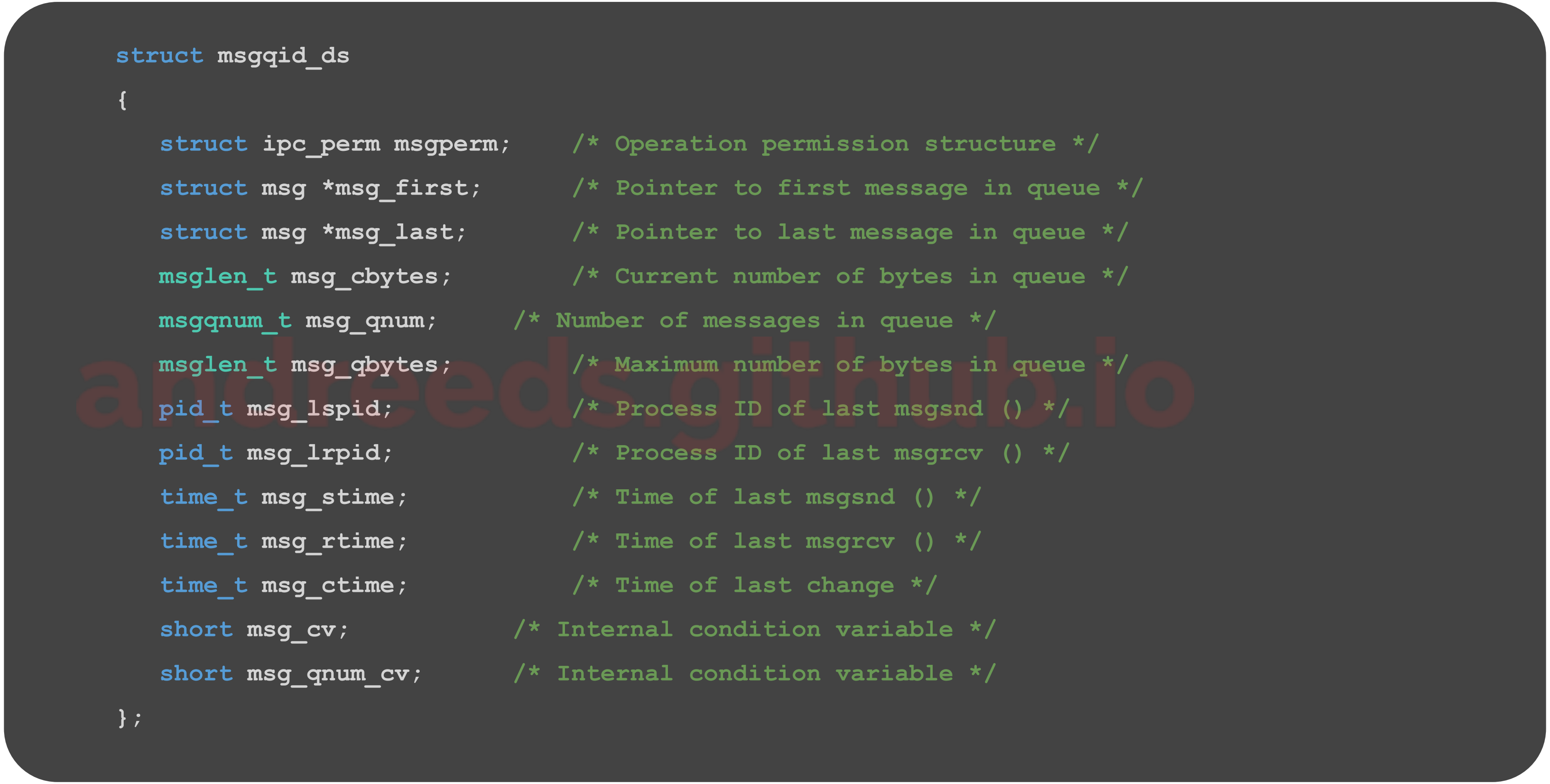
- The kernel maintains a data structure for every message queue in the system



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

DATA STRUCTURES

- Part of this data structure keeps track of permissions (like those used for files)



```
struct msgqid_ds
{
   struct ipc_perm msgperm;    /* Operation permission structure */
   struct msg *msg_first;      /* Pointer to first message in queue */
   struct msg *msg_last;       /* Pointer to last message in queue */
   msglen_t msg_cbytes;        /* Current number of bytes in queue */
   msgqnum_t msg_qnum;     /* Number of messages in queue */
   msglen_t msg_qbytes;        /* Maximum number of bytes in queue */
   pid_t msg_lspid;            /* Process ID of last msgsnd () */
   pid_t msg_lrpid;            /* Process ID of last msgrcv () */
   time_t msg_stime;           /* Time of last msgsnd () */
   time_t msg_rtime;           /* Time of last msgrcv () */
   time_t msg_ctime;           /* Time of last change */
   short msg_cv;           /* Internal condition variable */
   short msg_qnum_cv;      /* Internal condition variable */
};
```

ELEMENTARY SYSTEM CALLS

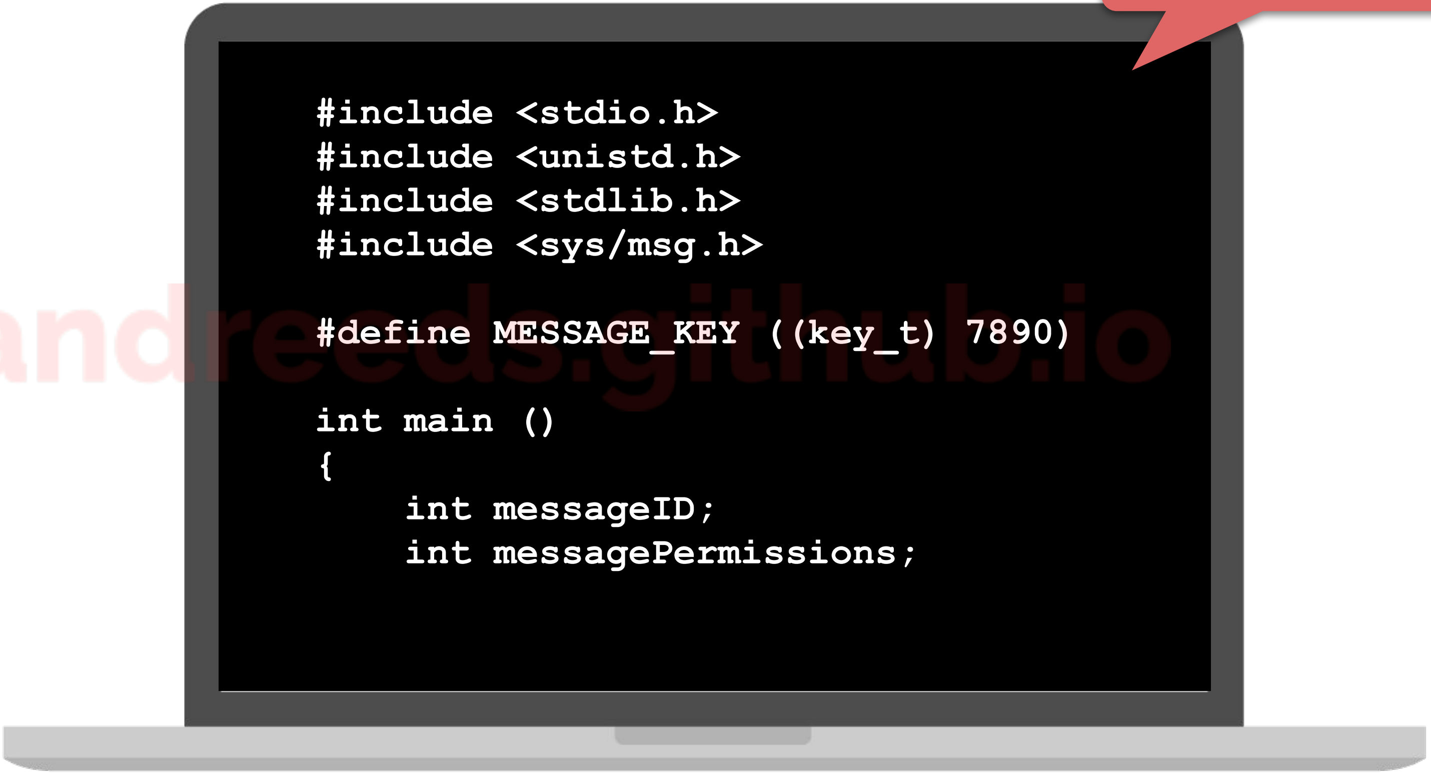
The `msgget` system call is used to create a **new**, or **access** an existing, **message queue**

```
#include <sys/msg.h>
int msgget (key_t key, int msgflag);
```

- The first parameter, key, designates the particular object to be created or accessed, and can be created by:
  - Letting the system pick the key (`IPC_PRIVATE`)
  - Picking the key directly by storing it in a header
- The third parameter, `msgflag`, specifies the access permissions for the message queue segment
- If successful, `msgget` returns a non-negative integer corresponding to the **message queue identifier**
  - Similar to a file descriptor
- If unsuccessful, msgget returns **−1** and sets `errno`

ELEMENTARY SYSTEM CALLS

aloc_m_q.c



```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <sys/msg.h>

#define MESSAGE_KEY ((key_t) 7890)

int main ()
{
    int messageID;
    int messagePermissions;
```

SHELL COMMANDS

There are two shell commands for working with message queue segments:

- `ipcs` : View the status of a shared memory segment
- `ipcrm` : Remove a message queue segment

- Example – Removing a shared memory segment
  - `ipcrm -q 0`
    - `0` is the `msqid` from `ipcs` and is equal to segmentID

ELEMENTARY SYSTEM CALLS

The `msgsnd` system call **appends a new message to the end of a message queue**

```
#include <sys/msg.h>
int msgsnd (int msgqid, const void *msg, size_t msgsz,
                    int msgflag);
```

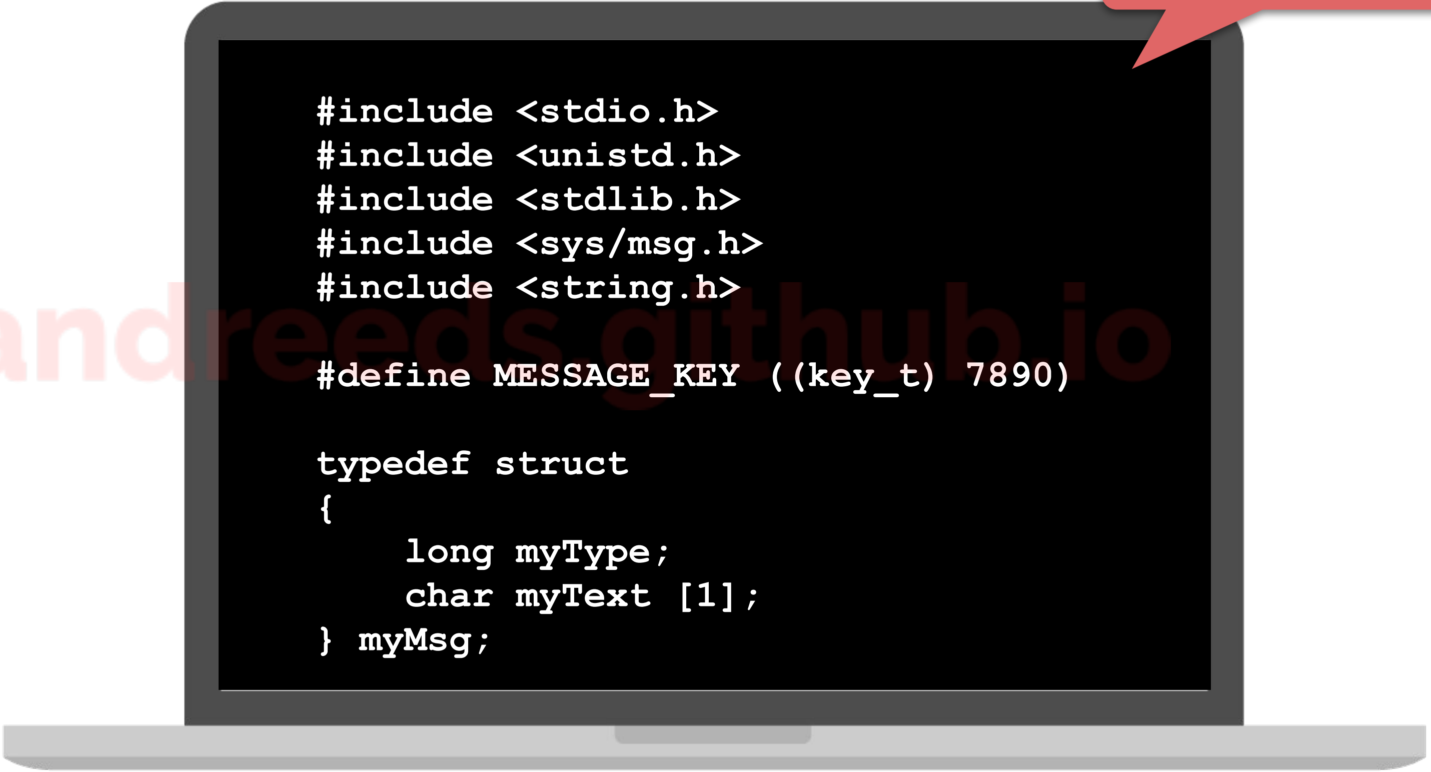
- The first parameter, `msgqid`, is the identifier for an existing message queue
- The second parameter, `msg`, is a pointer to a user-defined structure containing a message type and the actual text part of the message

```
struct myMsg
{
   long myType;
   char myText [1];
};
```

- The third parameter, `msgsz`, is the length of the actual text part of the message
- The fourth parameter, `msgflag`, determines what happens if the message queue is full
  - set it to **0** to indicate that the process should block until there is room in the message queue
- If **successful**, msgsnd returns **0**
- If unsuccessful, msgsnd returns **–1** and sets `errno`

ELEMENTARY SYSTEM CALLS

send_m_q.c



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

#define MESSAGE_KEY ((key_t) 7890)

typedef struct
{
    long myType;
    char myText [1];
} myMsg;
```

ELEMENTARY SYSTEM CALLS

The `msgrcv` system call reads a message from a message queue

```
#include <sys/msg.h>
ssize_t msgrcv (int msgqid, void *msg, size_t msgsz, long msgtype,
                int msgflag);
```

- `msgqid` is the identifier for an existing message queue
- `msg` is a pointer to a user-defined structure containing a message type and the actual text part of the message.

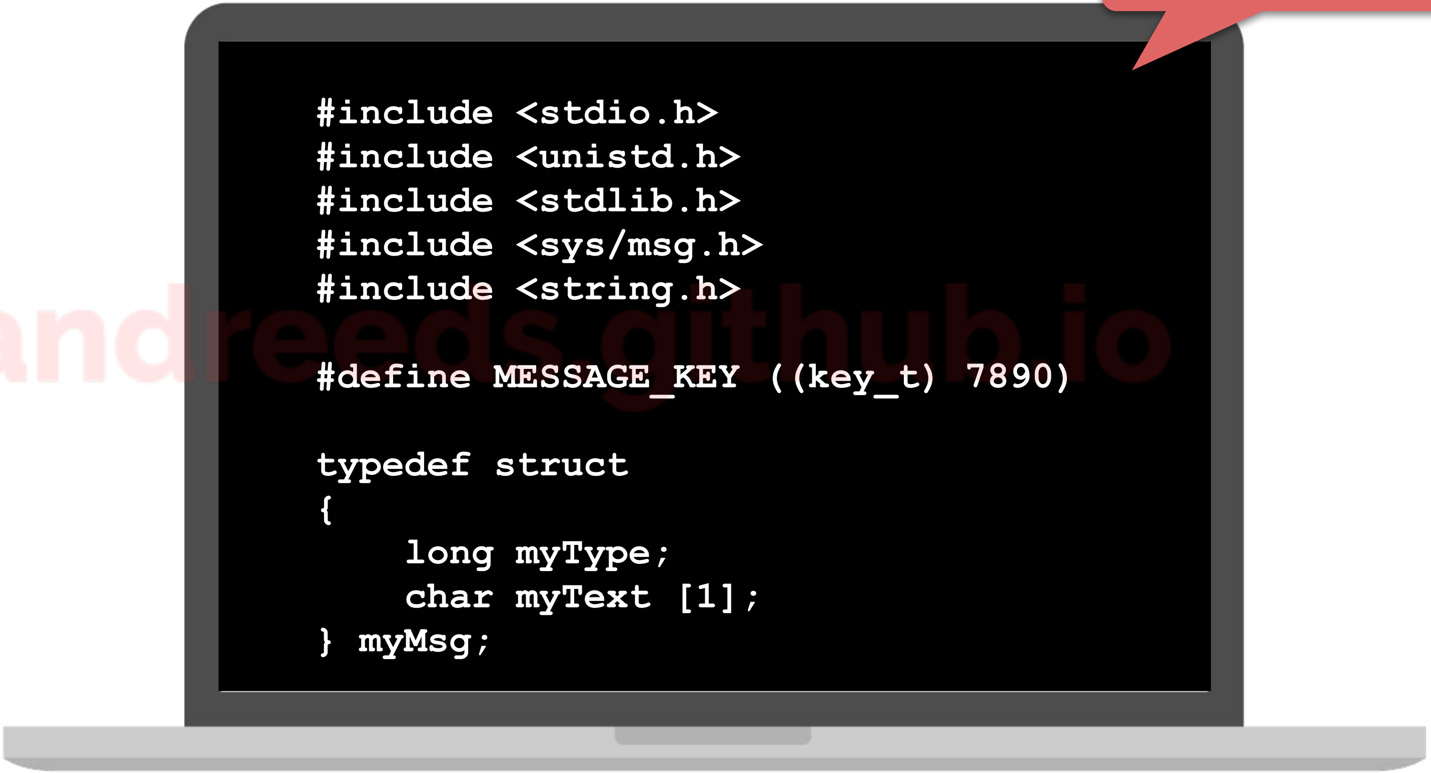
```
struct myMsg
{
   long myType;
   char myText [1];
};
```

- `msgsz` is the maximum length of the data that can be stored in msg
- `msgtype` determines which message is read from the message queue
- `msgflag` determines the action that should be taken if the requested message is not available
- If successful, `msgrcv` returns the **number of bytes in the text part of the message**
- If unsuccessful, msgrcv returns **–1** and sets `errno`

ELEMENTARY SYSTEM CALLS

send_m_q.c

URCourses



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

#define MESSAGE_KEY ((key_t) 7890)

typedef struct
{
    long myType;
    char myText [1];
} myMsg;
```

ELEMENTARY SYSTEM CALLS

The `msgctl` system call performs a **control operation** on a message queue.

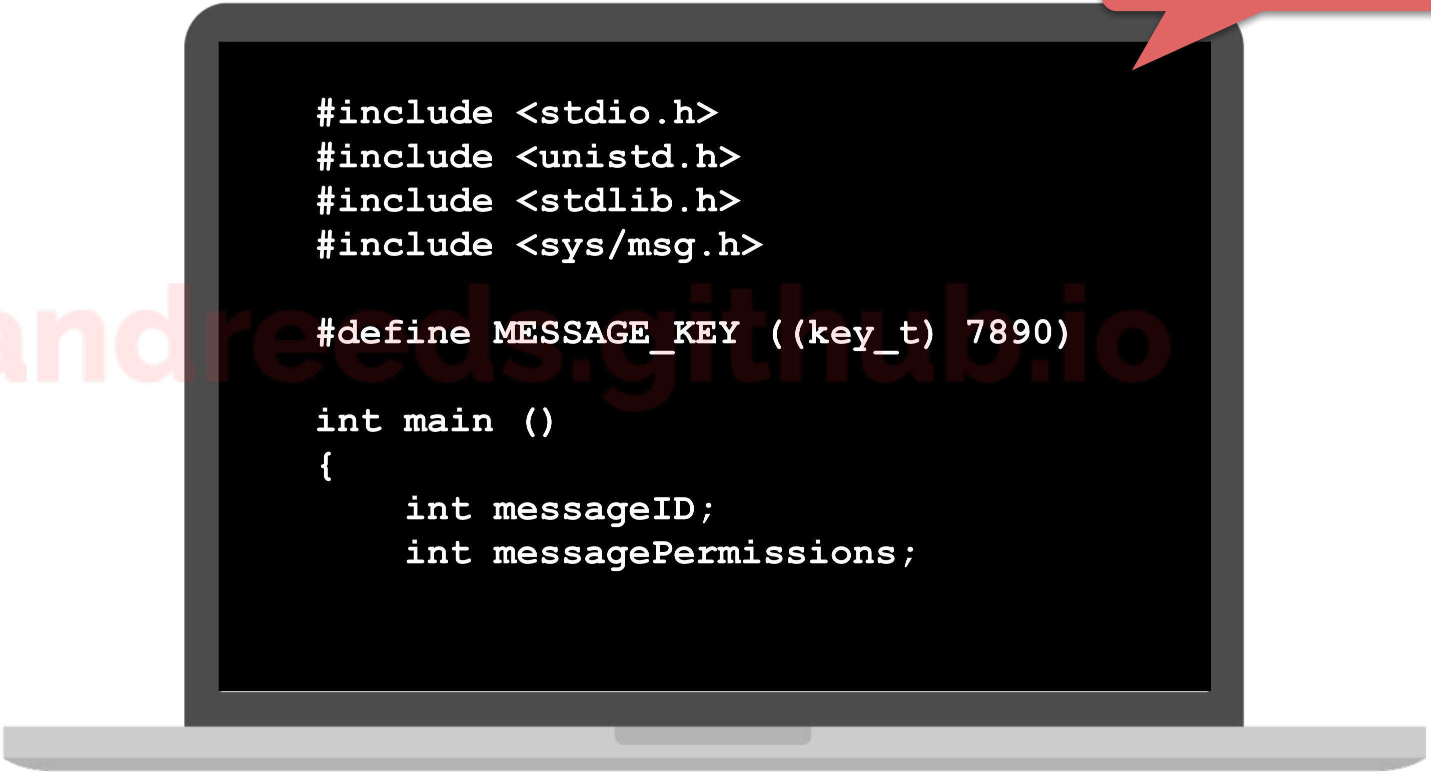
```
#include <sys/msg.h>
int msgctl (int msgqid, int operation, struct msgqid_ds *buffer);
```

- `msgqid` is the identifier for a message queue
- `operation` specifies one of five valid operations
  - e.g
    - `IPC_STAT`: Place a copy of the kernel-maintained `msgqid_ds` structure in buffer
    - `IPC_RMID`: Remove the message queue from the system
- If successful, msgctl returns **0**
- If unsuccessful, msgctl returns **–1** and sets `errno`

ELEMENTARY SYSTEM CALLS

send_m_q.c

URCourses



```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <sys/msg.h>

#define MESSAGE_KEY ((key_t) 7890)

int main ()
{
    int messageID;
    int messagePermissions;
```