

# **CS241 System Programming**

## **IPC – Shared Memory and Message Queues (VI)**

Klara Nahrstedt

Lecture 39

4/26/2006



# Contents

---

- Introduction to inter-process communication functions
- XSI Shared Memory
  - Access, attach, detach shared memory
- XSI Message Queues
  - Create message queue, send and receive to/from message queues

# Administrative

- Read R&R Chapter 15.1, 15.3, and 15.4
- MP5 is on – deadline May 1
- Homework 2 – posted April 24– deadline May 3, midnight
- Quiz 11 – April 28
  - Cover R&R Chapter 18.1-18.7
  - Cover R&R Chapter 20.1-20.8
- **Quiz 12 – May 3 → OPTIONAL !!!**
  - Cover Synchronization
  - Cover File Systems
  - Cover Memory Management

# POSIX: XSI Interprocess Communication (IPC)

- IPC is part of POSIX: XSI Extension
- Origin in UNIX System V IPC
- IPC includes
  - Message queues
  - Semaphore sets
  - Shared memory
- We will go into detail on shared memory IPC mechanism and message queues

# IPC Functions

- **Message Queues** functions and their meaning
  - msgctl /\*control\*/
  - msgget /\*create or access\*/
  - msgrcv /\* receive message\*/
  - msgsnd /\*send message\*/
- **Shared Memory** functions and their meaning
  - shmat /\*attach memory to process\*/
  - shmctl /\*control\*/
  - shmdt /\*detach memory from process \*/
  - shmget /\*create and initialize or access \*/

# Identifying and Accessing IPC Objects

- POSIX:XSI IPC object
  - unique integer  $\geq 0$
  - Returned from the get function for the object (similar to open file descriptors)
- Examples:
  - `msgget` returns an integer identifier for message queue objects
- Identifiers are associated with **additional data structures** in `sys/msg.h` or `sys/shm.h`
- Creation or accessing of IPC object requires a **key**
- Key can be created
  - By system (IPC\_PRIVATE)
  - By user – pick the key directly
  - By using `ftok` – the system generates a key from a specified path

# ftok Function

- `ftok` function allows independent processes to **derive the same key based on a known pathname**

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

```
key_t ftok(const char *path, int id);
```

- **Example:**

```
if ((thekey = ftok("tmp/trouble.c", 1) == (key_t)-1))  
perror("failed to derive key from /tmp/trouble.c");
```

# Accessing IPC Resources from Shell

- `ipcs` command displays information about POSIX:XCI **IPC resources**

```
ipcs [-qms] [-a | -bcopt]
```

- If no options are given, `ipcs` gives **all information** on message queues, share memory segments and semaphore sets
- If option `-q`, `-m` and `-s` are given, resources of message queues, shared memory, and semaphore sets are shown, respectively.

# POSIX:XSI Shared Memory

- Shared memory IPC allows processes to read/write from/to the **same memory segment**
- We need `sys/shm.h` header file
- Important data structure is the representation of the actual shared memory segment **`shmid_ds`**

```
struct ipc_perm shm_perm; /*operation permission
    structure */
size_t shm_segsz; /*size of segment in bytes*/
pid_t shm_lpid; /*process ID of last operation */
pid_t shm_cpid; /*process ID of creator*/
shmatt_t shm_nattch; /*number of current attaches*/
time_t shm_atime; /*time of last shmat*/
time_t shm_dtime; /*time of last shmdt*/
time_t shm_ctime; /*time of last shctl*/
```

# Accessing a shared memory segment

- Need function `shmget` to access shared memory segment
  - Function returns an **identifier for the shared memory** segment associated with the key parameter
  - Function **creates the shared segment** if either the key is `IPC_PRIVATE` or `shmflg&IPC_CREAT` is non-zero and no shared segment or identifier are associated with key.

```
#include <sys/shm.h>
int shmget(key_t key, size_t size, int shmflg);
```

# Attaching and detaching a shared memory segment

- Need function `shmat` to **attach shared memory segment**

- Use structure `shm_id` for needed information
- **Increment** the value of `shm_nattach` for `shm_id`

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

```
void *shmat(int shm_id, const void *shm_addr, int shm_flg);
```

- Need function `shmdt` to **detach shared memory segment**

- **Decrement** `shm_nattach` in the structure `shm_id`

```
int shmdt(const void *shm_addr);
```

- Need function `shmctl` to **deallocate the shared memory**

- The **last process to detach the segment should deallocate** the shared memory segment by calling `shmctl`

# Controlling shared memory

- Need function `shmctl` to provide a variety of control operations on the shared memory segment `shmid`
  - Control operations are specified via `cmd` parameter
  - Interpretation of `buf` parameter depends on value of `cmd`

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

```
int shmctl(int shmid, int cmd, struct shmids  
*buf);
```

- `cmd` control commands
  - `IPC_RMID` – **remove shared memory segment** `shmid` and destroy corresponding `shmids`
  - `IPC_SET` – **set values of fields** for shared memory segment `shmid` from values found in `buf`
  - `IPC_STAT` – **copy current values** for shared memory segment `shmid` into `buf`

## Example (detachandremove function)

```
#include <stdio.h>
#include <errno.h>
#include <sys/shm.h>
int detachandremove(int shmid, void *shmaddr){
int error =0;
if (shmdt(shmaddr) == -1) error = errno;
if (shmctl(shmid, IPC_RMID, NULL) == -1) && !error) error =
    errno;
if (!error) return 0;
errno=error;
return -1; }
```

# Shared Memory Examples

- Consider the following example (Program 15.5):
  - Parent and child process **share a small memory segment**
  - The **child stores its byte count** in the shared memory
  - The **parent waits for the child to finish** and then **outputs the number of bytes** received from each process along with the sum of these values
  - The parent creates the shared memory segment by using **IPC\_PRIVATE** – allows the memory to be shared with the child
  - The parent does not access the shared memory until it has detected the **termination of the child**

## Program 15.5 (Setup Shared Memory)

```
../* declare h files */
../* start program and declare variables */
if (((fd1 = open(argv[1],O_RDONLY)) == -1) ||
    fd2 = open(argv[2],O_RDONLY)) == -1)) {
    perror("Failed to open file"); return 1;
}
if ((id = shmget(IPC_PRIVATE, sizeof(int), PERM)) == -1) {
    perror("failed to create shared memory segment"); return 1;
}
if ((sharedtotal = (int *)shmat(id, NULL, 0)) == void *)-1) {
    perror("failed to attach shared memory segment"); return 1;
}
```

## Program 15.5 (Parent-Child Communication)

```
if ((childpid = fork()) == -1) {
    perror("failed to create child process");
    if (detachremove(id, sharedtotal) == -1)
        perror("failed to destroy shared memory segment"); return 1; }
if (childpid > 0) fd = fd1; /*parent code*/
else fd = fd2;
while ((bytesread = readwrite(fd, STDOUT_FILENO)) > 0)
    totalbytes += bytesread;
if (childpid == 0) *sharedtotal = totalbytes; return 0; /*child code*/

if (r_wait(NULL) == -1) perror("failed to wait for child");
else ..../* print information */
if (detachandremove(id, sharedtotal) == -1) {
    perror("failed to destroy shared memory segment"), return 1; }
return 0; }
```

# POSIX:XSI Message Queues

- Message queues
  - IPC mechanism
    - allows a process to send and receive messages from other processes.
- Important data structures for message queues are in `sys/msg.h`.
- Major data structure is `msqid_ds`

```
struct ipc_perm msg_perm; /*oper. permission structure */
msgqnum_t msg_qnum; /*number of msg currently in queue*/
msglen_t msg_qbytes; /*max. bytes allowed in queue*/
pid_t msg_lspid; /* process ID of msgsnd*/
pid_t msg_lrpid; /* process ID of msgrcv*/
time_t msg_stime; /* time of last msgsnd*/
time_t msg_rtime; /* time of last msgrcv */
time_t msg_ctime; /* time of last msgctl */
```

# Access a message queue

- Need function `msgget` to **access the message queue**
  - Returns message queue **identifier associated with key** parameter
  - Creates the identifier if either `key = IPC_PRIVATE` or `msgflg&IPC_CREAT` is nonzero and no message queue or identifier are already associated with key

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

- Example: create a new message queue

```
#define PERMS (S_IRUSR | S_IWUSR)

int msqid;
if ((msqid = msgget(IPC_PRIVATE, PERMS)) == -1)
    perror("failed to create new private message queue");
```

# Insert Message into queue

- Need function `msgsnd` to **insert messages into the queue**
  - Parameter `msqid` identifies the **message queue**
  - Parameter `msgp` points to a **user-defined buffer** that contains the message to be sent
  - Parameter `msgsz` specifies the **actual size of the message text**
  - Parameter `msgflg` specifies **actions** to be taken under various conditions

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

- **User-defined message format**

```
struct mymsg{
    long mtype; /* message type */
    char mtext[1]; /*message text*/
} mymsg_t;
```

## Protocol to send message (e.g., `mymessage`) to a message queue

- Step 1: **Allocate** buffer `mbuf` which is of type `mymsg_t` and size `sizeof(mymsg_t) + strlen(mymessage)`
- Step 2: **Copy** `mymessage` into the `mbuf->mtext` member
- Step 3: **Set** message type in the `mbuf->mtype` member
- Step 4: **Send** the message
- Step 5: **Free** `mbuf`

**Don't forget to check for errors and to free `mbuf` if any error occurs**

# Remove message from message queue

- Need function `msgrcv` to receive message
  - Parameter `msqid` identifies the queue
  - Parameter `msgp` points to a user-defined buffer holding the message
  - Parameter `msgsz` specifies the size of message text
  - Parameter `msgtyp` can be used by the receiver for message selection
  - Parameter `msgflg` specifies actions to be taken under various conditions

```
#include <sys/msg.h>
ssize_t msgrcv(int msqid, void *msgp,
size_t msgsz, long msgtyp, int msgflg);
```

# Control operations on message queue

- Need function `msgctl` to **de-allocate or change permissions** for the message queue
  - Parameter `cmd` specifies the action
  - Parameter `buf` is used to write or read state information, depending on `cmd`
- Possibilities of control (`cmd`)
  - `IPC_RMID` – **remove the message queue** `msqid` and destroy the corresponding `msqid_ds`
  - `IPC_SET` – **set members** of the `msqid_ds` data structure from `buf`
  - `IPC_STAT` – **copy members** of the `msqid_ds` data structure into `buf`

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

## Example (Program 15.10)

- Consider program that copies messages from a message queue to standard output
- First, introduce `initqueue` to initialize the message queue
- Second, specify program that copies messages from the message queue to `STDOUT`

## Initialize message queue (Program 15.9)

```
... /* define h files and variables */
```

```
static int queueid
```

```
int initqueue(int key) {
```

```
    queueid = msgget(key, PERM | IPC_CREAT);
```

```
    if (queueid == -1) return -1;
```

```
    return 0; }
```

# Copy messages from message queue to STDOUT

```
../* define h files and variables */
#define MAXSIZE 4096;
typedef struct {
    long mtype;
    char mtext[MAXSIZE];
} mymsg_t;
int main(int argc, char *argv[]) {
    int id; int key; mymsg_t mymsg; int size;
    .... /* check arg*/
    key = atoi(argv[1]);
    id = initqueue(key); /*removed error code*/
    for ( ;; ) {
        size = msgrcv(id, &mymsg, MAXSIZE, 0,0);
        r_write(STDOUT_FILENO, mymsg.mtext, size); }
    ../*end code */ }
```

# Summary

- Pipes, semaphore sets (did not cover here, covered earlier in semester), message queues and shared memory are the **basic IPC communication mechanisms** among processes that are in
  - **producer – consumer , readers – writer, client – server,...**
- Pipes can be implemented with semaphores, with shared memory and with message queues
- Example:
  - represent pipe by `pipe_t` structure **allocated in shared memory**

```
typedef struct pipe {
    int semid; /*ID of protecting semaphore set*/
    int shmid; /* ID of the shared memory segment */
    char data[_POSIX_PIPE_BUF]; /* buffer currently in the pipe*/
    int data_size; /* bytes currently in the pipe */
    void *current_start; /*pointer to current start of data*/
    int end_of_file; /* true after pipe closed for writing*/
} pipe_t;
```
  - Need functions such as `pipe_open`, `pipe_read`, `pipe_write`, and `pipe_close`