

# Unix/Linux IPC

These slides are created by Dr. Huang of George Mason University. Students registered in Dr. Huang's courses at GMU can make a single machine readable copy and print a single copy of each slide for their own reference as long as the slide contains the copyright statement, and the GMU facilities are not used to produce the paper copies. Permission for any other use, either in machine-readable or printed form, must be obtained from the author in writing.

CS471

1

## Facilities

- ☐ **Shared memory segments**
- ☐ **Semaphores**
- ☐ **Message queues**
- ☐ Signals (not covered)
- ☐ Pipes (not covered)
- ☐ Sockets (not covered)
- ☐ Remote procedure calls (not covered)

CS471

2

## Shared Memory Segments

- ❑ Mmap()-ed areas survives `fork()` but not `exec()`.
- ❑ Shared memory segments in the contrary can be accessed by any process
  - not limited to parent-child processes
  - Permissions can be set to determine who has what access to it
- ❑ Following programming conventions, we will call a shared memory segment an **shm**.
- ❑ Each shm has a system-wide unique ID.

CS471

3

## Shm Operations

- ❑ **shmget()**
  - Creating new shm
  - Obtaining the ID of an existing shm
- ❑ **shmat()**: attach an shm as an VM area
- ❑ **shmdt()**: detach an shm
- ❑ **shmctl()**:
  - Read/modify permissions, owners etc.
  - Destroy shm

CS471

4

## shmget()

```
int shmget (int shm_key, int size,  
            int shm_flags);
```

- ❑ **shm\_key**: a system-wide unique key for the shm
- ❑ **size**: # of bytes of the segments
- ❑ **shm\_flags**: the bit-OR of the following
  - SHM\_CREAT
  - mode (0666=all can read and write, ...)
- ❑ Returns the ID of the shm

CS471

5

## shmat()

```
void* shmat (int shm_id,  
             void* shmaddr,  
             int flags);
```

- ❑ **shmaddr**: recommended address of the shm in the virtual memory space of the calling process
  - NULL means no recommendation
- ❑ **flags**:
  - 0, read/write
  - SHM\_RDONLY
- ❑ Returns the starting address of the new area

CS471

6

## shmdt()

```
int shmdt (void* shmaddr);
```

- ❑ **shmaddr**: the address of the shm (in the VM of the calling process) to be detached.
- ❑ Returns 0 when successful
- ❑ Detachment does NOT destroy an shm.

CS471

7

## Destroy Shm

```
shmctl (shm_id, IPC_RMID, NULL);
```

- ❑ This only marks the shm for destroying.
- ❑ The shm is destroyed when the last process detaches it.

CS471

8

## Creating A New Shm

```
shm_id = shmget (4567, 4096,  
                IPC_CREAT | 0600);  
shm_ptr = shmat (shm_id, 0, 0);
```

- ❑ It is your responsibility to make sure 4567 is a unique key on the system.
- ❑ The new area will survive `fork()` but not `exec()`.

CS471

9

## Using An Existing Shm by A Different Process

```
shm_id = shmget (4567, 4096, 0600);  
shm_ptr = shmat (shm_id, 0, 0);
```

- ❑ Notice the use of the same key, 4567.
- ❑ Notice the absence of `IPC_Creat` in `shmget()`. This causes the system to look for an already-present shm, rather than creating a new one.

CS471

10

## Finding Shm Keys

- ❑ Ensuring a key is system-wide unique can be a problem in multi-user environments.
- ❑ The `ftok()` function helps find such a key.
  - This is a library function, not a system call.
- ❑ `int ftok (char* pathname, int id);`
  - `pathname`: the name of an arbitrary file.
  - `id`: for further distinction.
  - Returns a key that is “very likely” unique.

CS471

11

## Semaphores

- ❑ A semaphore in Unix/Linux is a special purpose shm.
- ❑ Share the same key space with shm.
- ❑ A semaphore ID is associated with a set of atomic semaphores.

CS471

12

## Semget()

```
int semget (int sem_key, int nsem,  
            int semflg)
```

- ❑ `sem_key`: system-wide semaphore key
- ❑ `nsem`: number of semaphores in the set
- ❑ `semflg`: the bit-OR of the following
  - `IPC_CREAT`
  - mode (0666=all can read and write, ...)
- ❑ Returns the ID of the semaphore set.

CS471

13

## Example

- ❑ Producer creates new semaphore with key 1234 by

```
sem_id = semget (1234, 1,  
                IPC_CREAT | 0666);
```
- ❑ Consumer obtains that semaphore by

```
sem_id = semget (1234, 1, 0);
```

CS471

14

## Semctl()

- ❑ `semctl (sem_id, i, GETVAL);`
  - returns the value of the i-th semaphore in the set `sem_id`.
- ❑ `semctl (sem_id, 0, IPC_RMID);`
  - destroys the semaphore set `sem_id`.
- ❑ See its man page for other functions.

CS471

15

## Semop()

```
int semop (sem_id,  
           struct sembuf *sops,  
           int nops)
```

- ❑ `sem_id`: semaphore set ID
- ❑ `sops`: an array of semaphore operations, performed *atomically*
- ❑ `nops`: # of operations in `sops`
- ❑ Returns 0 when successful, or else -1.

CS471

16



## Struct sembuf

- ❑ Each operation is described by a `sembuf` structure, including the following members
  - `sem_flg`: 0 in most situations
  - `sem_num`: give the semaphore to which this operation is applied. The first semaphore in a set is numbered 0.
  - `sem_op`: see next page

CS471

17

## Sem\_op

- ❑ Greater than 0
  - Semaphore value  $+=$  `sem_op`
  - Calling process proceeds immediately
- ❑ Equal to 0
  - Calling process waits until the value of the semaphore becomes 0
- ❑ Less than 0
  - Calling process waits until  
     $\text{semaphore value} + \text{sem\_op} \geq 0$
  - Semaphore value  $+=$  `sem_op`

CS471

18

## Message Queues

```
int msgget (int msg_key, int flags)
```

- ❑ `msg_key`: system-wide semaphore key; the same key space with `shm`.
- ❑ `flags`: the bit-OR of the following
  - `IPC_CREAT`
  - mode (0666=all can read and write, ...)
- ❑ Returns the ID of the message queue.

CS471

19

## Send Messages

```
int msgsnd (msg_id, void* msg_ptr,  
            int msize, int flags)
```

- ❑ `msg_id`: message queue ID
- ❑ `msg_ptr`: a pointer to the message, which must follow the format

```
struct msg_type {  
    long mtype; /* message type */  
    Other members determined by the app.  
}
```
- ❑ `msize`: message size in bytes, excluding `mtype`
- ❑ `flags`: 0 in most situations
- ❑ Returns 0 when successful, or else -1.

CS471

20

## Receive Messages

```
int msgrcv (msg_id, void* msg_ptr,  
            int msize,  
            int mtype, int flags)
```

- ❑ msg\_id: message queue ID
- ❑ msg\_ptr: a pointer to the message.
- ❑ msize: message size in bytes, excluding mtype
- ❑ mtype: retrieve from the queue the first message with the given type
- ❑ flags: 0 in most situations
- ❑ Returns 0 when successful, or else -1.

CS471

21

## Message Formats

<pre>struct command_msg {     long mtype;     char cmd;     int n; };</pre>	<pre>struct result_msg {     long mtype;     int result; };</pre>
---	---

CS471

22

## Foreground

```
int main ()
{
    /* obvious variable declarations omitted */
    int cmd_q, result_q; /* message queue Ids */
    struct command_msg c_msg;
    struct result_msg r_msg;

    c_msg.mtype = 1;
    cmd_q = msgget (47103, IPC_CREAT | 0600);
    result_q = msgget (47104, IPC_CREAT | 0600);
```

CS471

23

```
    if (!(pid=fork()))
        execl ("backg", "backg", NULL);

    while (1) {
        scanf ("%c", &c_msg.cmd);
        if (c_msg.cmd == 'q') {
            msgsnd (cmd_q, &c_msg,
                sizeof c_msg - sizeof c_msg.mtype , 0);
            wait(0);
            msgrcv (cmd_q, IPC_RMID, NULL);
            msgrcv (result_q, IPC_RMID, NULL);
            exit(0);
        } /* end of command 'q' */
```

CS471

24

```

    /* Handle 's' and 'f' commands */
    scanf ("%d", &c_msg.n);
    msgsnd (cmd_q, &c_msg,
            sizeof c_msg - sizeof c_msg.mtype , 0);
    msgrcv (result_q, &r_msg,
            sizeof r_msg.result, 1, 0);
    printf ("Result = %d\n", r_msg.result);

} /* end of while (1) */
} /* end of main() */

```

CS471

25

## Background

```

int main ()
{
    /* obvious variable declarations omitted */
    int cmd_q, result_q; /* message queue Ids */
    struct command_msg c_msg;
    struct result_msg r_msg;

    r_msg.mtype = 1;
    cmd_q  = msgget (47103, 0);
    result_q = msgget (47104, 0);

```

CS471

26

```

while (1) {
    msgrcv (cmd_q, &c_msg,
           sizeof c_msg - sizeof c_msg.mtype, 1, 0);

    switch (c_msg.cmd) {
        case 's':
            for (sum=i=0; i<=c_msg.n; i++) sum+=i;
            r_msg.result = sum;
            break;
        case 'f': /* details omitted */
        case 'q': exit(0);
    } /* end of switch() */

    msgsnd (result_q, &r_msg, sizeof r_msg.result , 0);
} /* end of while (1) */

```

CS471

27

## Project #3

- ❑ Redo project #2 using semaphores and shm.
- ❑ Due midnight of July 23rd

CS471

28

## Foreground Process

- ❑ Accept the same commands as project #2.
- ❑ Communicate with background through shm.
  - Use `ftok()` to generate `shm_key`
  - Use your home directory as the pathname and character 'm' as the ID.
- ❑ Use semaphores for synchronization.
  - Use `ftok()` to generate `sem_key`
  - Use your home directory as the pathname.
  - Use IDs '1', '2', '3' ...
- ❑ Destroy shm and semaphores when terminates

CS471

29

## Foreground Logic

- ❑ Print own process ID
- ❑ Creates shm and semaphores
- ❑ Attach shm
- ❑ Execute the following in a loop
  - Wait for user command
  - Semaphore-signal background process for the readiness of the command and operand
  - Semaphore-wait for result
  - Print result
- ❑ Destroy shm and semaphores

CS471

30

## Background Process

- ❑ Perform the computations of summation and factorial.
- ❑ Use shm and semaphores for communications and synchronizations.
- ❑ Semaphore and shm keys are obtained through command line arguments:
  - **backg** *shm\_key sem\_key1 sem\_key2*
- ❑ Terminate itself when seeing the Q/q command.
- ❑ Detach the shm when terminates.

CS471

31

## Background Logic

- ❑ Print own process ID
- ❑ Obtain shm and semaphores
- ❑ Attach shm
- ❑ Execute the following in a loop
  - Semaphore-wait for user command
  - Perform computations
  - Semaphore-signal the foreground for the readiness of the result.
- ❑ Detach shm

CS471

32