Advanced Programming in the UNIX Environment

Week 08, Segment 2: System V IPC

Department of Computer Science
Stevens Institute of Technology

Jan Schaumann
jschauma@stevens.edu
https://stevens.netmeister.org/631/
System V IPC

There are three types of asynchronous IPC originating from System V:

• Semaphores
• Shared Memory
• Message Queues

All three use IPC structures, referred to by an identifier and a key; all three are (necessarily) limited to communication between processes on one and the same host.

Since these structures are not known by name, special system calls (msgget(2), semop(2), shmat(2), etc.) and special userland commands (ipcrm(1), ipcs(1), etc.) are necessary.
System V IPC: Semaphores

A semaphore is a counter used to provide access to a shared data object for multiple processes. To obtain a shared resource a process needs to do the following:

1. Test semaphore that controls the resource.
2. If value of `semaphore > 0`, decrement semaphore and use resource; increment semaphore when done.
3. If `value == 0` sleep until `value > 0`

Semaphores are obtained using `semget(2)`, properties controlled using `semctl(2)`, operations on a semaphore performed using `semop(2)`. 
System V IPC: Semaphores

Press return to unlock:
Unlocked.
jschauma@apue$ ipcs -s
IPC status from <running system> as of Mon Oct 19 01:38:29 2020

Semaphores:
  T    ID   KEY   MODE   OWNER   GROUP
s 720896 704662389  --rw--rw--   jschauma   users

jschauma@apue$ ./a.out
Press return to lock:
Trying to lock...
Locked.
Press return to unlock:
Unlocked.
jschauma@apue$ ipcrm -s 720896
jschauma@apue$ ipcs -s
IPC status from <running system> as of Mon Oct 19 01:39:47 2020

Semaphores:
  T    ID   KEY   MODE   OWNER   GROUP
s 720896 704662389  --rw--rw--   jschauma   users

Press return to unlock:
Unlocked.
jscp

Locked.
Press return to unlock:
Unlocked.
sudo su - fred
fred@apue$ whoami
fred
cd ~jschauma/08
fred@apue$ ./a.out
Press return to lock:
Trying to lock...
Locked.
Press return to unlock:
Unlocked.
fred@apue$ ipcs -s
IPC status from <running system> as of Mon Oct 19 01:39:20 2020
IPC Data Flow

$ cat input | cat > output
System V IPC: Shared Memory
System V IPC: Shared Memory

• fastest form of IPC
• access to shared region of memory often controlled using semaphores
• obtain a shared memory identifier using `shmget(2)`
• attach shared memory segment to a processes address space by calling `shmat(2)`
• detach it using `shmdt(2)`
• catch-all for shared memory operations: `shmctl(2)`
System V IPC: Shared Memory

```bash
jm  327681  704662386  --rw-rw-rw-  jschauma  users  jschauma  users  0  1024 1074 1074 2:23:10 2:23:10 2:23:10

jm@apue$ ./a.out
1359: segment contains: "The cow says 'moo'."

jm@apue$ ./a.out
1579: segment contains: "The cow says 'moo'."

jm@apue$ ipcs -ma
IPC status from <running system> as of Mon Oct 19 02:23:52 2020

Shared Memory:
```
```
```
jm@apue$ ipcrm -m 327681
jm@apue$ ./a.out
1751: segment contains: ""

jm@apue$ ipcs -ma
IPC status from <running system> as of Mon Oct 19 02:25:06 2020

Shared Memory:
```
High address (args and env):

- `envp[16]` at `0x7FFFD279D8`
- `environ[16]` at `0x7FFFD279D8`
- `envp[8]` at `0x7FFFD27958`
- `environ[0]` at `0x7FFFD27958`
- `last arg` at `0x7FFFD27950`
- `first arg` at `0x7FFFD27948`

Stack:

- `func_array[]` ends at `0x7FFFD278DC`
- `func_array[]` (like 'array[]', but on stack) begins at `0x7FFFD278C0`
- `argv` at `0x7FFFD277B8`
- `argc` at `0x7FFFD277B0`
- `envp` at `0x7FFFD278AB`
- `func2 (from main): frame at` `0x7FFFD27884`
- `func frame at` `0x7FFFD27884`
- `static int n within func at` `0x601EEC`
- `func (called 1 times): frame at` `0x7FFFD27884`
- `func2 (from func): frame at` `0x7FFFD2785C`

Shared Memory:

- `shared memory area ends at` `0x7FFFD27AB`
- `shared memory area begins at` `0x7FFFD278A3`

Heap:

- `malloced area ends at` `0x7926986E5020`
- `malloced area begins at` `0x7926986E5000`

Uninitialized Data (BSS):

- `array[]` ends at `0x601F70`
- `array[]` (uninitialized, fixed-size char * on BSS) from `0x601F60`
- `num2 (uninitialized global int)` at `0x601F58`
- `string2 (uninitialized global char *)` at `0x601F50`
- `extern **environ` at `0x601F40`

Initialized Data:

- `extern **environ` at `0x601F40`
System V IPC: Message Queues

- linked list of messages stored in kernel space
- create or open existing queue using msgget(2)
- add message at end of queue using msgsnd(2)
- receive messages from queue using msgrcv(2)
- control queue properties using msgctl(2)

The message itself is contained in a user-defined structure such as

```c
struct mymsg {
    long mtype;    /* message type */
    char mtext[512]; /* body of message */
};
```
System V IPC: Message Queues

```
jschemaux@apue$ ./rcv 1
Who will get this?
schemaux@apue$ ./send "Unblock thyself."
Usage: msgsnd key message
schemaux@apue$ ./send 1 "Unblock thyself."
schemaux@apue$ ipcs -qa
IPC status from <running system> as of Mon Oct 19 20:13:28 2020

Message Queues:
T  ID  KEY  MODE  OWNER  GROUP  CREATOR  CGROUP  CBYTES  QNUM  QBYTES  LSPID  LRPID  STIME  RTIME  CTIME
q  393216  1  --rw-r--r--  jschemaux  users  jschemaux  users  0  0  2048  1652  952  20:11:20  20:12:01  20:10:48
```

cchemaux@apue$ ./rcv 2
msgset: No such file or directory
schemaux@apue$ ipcrm -q 393216
schemaux@apue$ ipcs -qa
IPC status from <running system> as of Mon Oct 19 20:13:48 2020

Message Queues:
T  ID  KEY  MODE  OWNER  GROUP  CREATOR  CGROUP  CBYTES  QNUM  QBYTES  LSPID  LRPID  STIME  RTIME  CTIME
```
POSIX Message Queues

`mq(3)` provides a real-time IPC interface similar to System V message queues. Notably:

- message queues are identified by a named identifier (no `ftok(3)` needed)
- message queues may or may not be exposed in the filesystem (e.g., `/dev/mqueue`)
- `mq_send(3)` and `mq_receive(3)` allow both blocking and non-blocking calls
- `mq_send(3)` lets you specify a priority; equal priority messages are queued as a FIFO, but higher priority messages are inserted before those of a lower priority
- `mq(3)` provides an asynchronous notification mechanism: `mq_notify(3)`
POSIX Message Queues

Number of messages in queue: 1
Message of priority 1: TUNA
Message of priority 0: avocado
Message of priority 0: onion
Message of priority 0: tomato

Number of messages in queue: 1
Message of priority 0: avocado
Number of messages in queue: 1
Message of priority 0: onion
Number of messages in queue: 1
Message of priority 0: tomato
Number of messages in queue: 1
Message of priority 1: TUNA

Number of messages in queue: 1
Message of priority 0: avocado
Number of messages in queue: 1
Message of priority 0: onion
Number of messages in queue: 1
Message of priority 0: tomato
Number of messages in queue: 1
Message of priority 1: TUNA

jschauma@apue$ ./mqsend avocado onion tomato
jschauma@apue$ ./mqsend -w avocado onion tomato
jschauma@apue$ ./mqsend avocado onion tomato
jschauma@apue$ ./mqsend avocado onion tomato
jschauma@apue$
System V IPC

- *asynchronous* IPC between processes on the same system
- old, but not obsolete
- semaphores are useful to guard access to shared resources
- shared memory allows for fast IPC
- message queues as a service are a popular way to implement "pub sub" models
  - Amazon Simple Queue Service
  - Apache Kafka
  - Java Message Service
  - RabbitMQ
  - ...