## Linux/UNIX IPC Programming

# POSIX Shared Memory

### Michael Kerrisk, man7.org © 2025

August 2025

#### mtk@man7.org

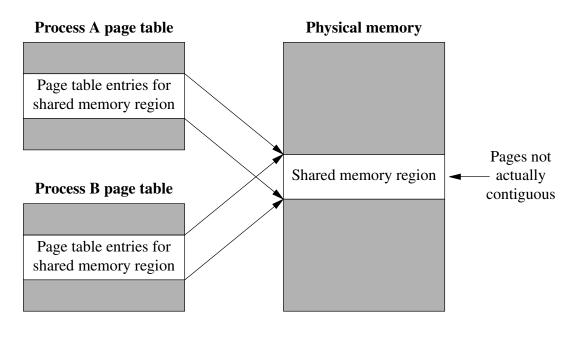
Outline	Rev: # caf166f4161b
16 POSIX Shared Memory	16-1
16.1 Overview	16-3
16.2 Creating and opening shared memory objects	16-8
16.3 Using shared memory objects	16-22
16.4 Synchronizing access to shared memory	16-31
16.5 API summary	16-41
16.6 Exercises	16-43

#### Outline

16	POSIX Shared Memory	16-1
16.1	Overview	16-3
16.2	Creating and opening shared memory objects	16-8
16.3	Using shared memory objects	16-22
16.4	Synchronizing access to shared memory	16-31
16.5	API summary	16-41
16.6	Exercises	16-43

### Shared memory

- Data is exchanged by placing it in memory pages shared by multiple processes
  - Pages are in user virtual address space of each process



#### Shared memory

- Data transfer is not mediated by kernel
  - User-space copy makes data visible to other processes
    - → Very fast IPC
  - Compare with (e.g.) pipes and sockets:
    - Send requires copy from user to kernel memory
    - Receive requires copy from kernel to user memory
- But, **need to synchronize access** to shared memory
  - E.g., to prevent simultaneous updates
  - Commonly, semaphores are used



Linux/UNIX IPC Programming

©2025 M Kerrisk

**POSIX Shared Memory** 

16-5 §16.1

#### POSIX shared memory objects

- Implemented (on Linux) as files in a dedicated tmpfs filesystem
  - tmpfs == memory-based filesystem that employs swap space when needed
- Objects have kernel persistence
  - Objects exist until explicitly deleted, or system reboots
  - Can map an object, change its contents, and unmap
  - Changes will be visible to next process that maps object
- Accessibility: user/group owner + permission mask



©2025 M. Kerrisk

POSIX Shared Memory

### POSIX shared memory APIs

- shm\_open(): open existing shared memory (SHM) object/create and open new SHM object
  - Returns file descriptor that refers to open object
- ftruncate(): set size of SHM object
- mmap(): map SHM object into caller's address space
- close(): close file descriptor returned by shm\_open()
- shm\_unlink(): remove SHM object name, mark for deletion once all processes have closed
- munmap(): unmap SHM object (or part thereof) from caller's address space
- Compile with cc -lrt
  - (No longer needed since glibc 2.34)
- shm\_overview(7) manual page

man7.org
Linux/UNIX IPC Programming

©2025 M. Kerrisk

**POSIX Shared Memory** 

16-7 §16.1

#### Outline

<u> </u>		
16 POSIX Sh	ared Memory	16-1
16.1 Overview		16-3
16.2 Creating	and opening shared memory objects	16-8
16.3 Using sha	ared memory objects	16-22
16.4 Synchron	nizing access to shared memory	16-31
16.5 API sum	mary	16-41
16.6 Exercises		16-43

### Creating/opening a shared memory object: shm\_open()

- Creates and opens a new object, or opens an existing object
- name: name of object (/somename)
- Returns file descriptor on success, or −1 on error
  - This FD is used in subsequent APIs to refer to SHM
  - (The close-on-exec flag is automatically set for the FD)



[TLPI §54.2]

Linux/UNIX IPC Programming

©2025 M Kerrisk

**POSIX Shared Memory** 

16-9 §16.2

### Creating/opening a shared memory object: shm\_open()

#### oflag specifies flags controlling operation of call

- O\_CREAT: create object if it does not already exist
- O\_EXCL: (with O\_CREAT) create object exclusively
  - Give error if object already exists
- O\_RDONLY: open object for read-only access
- O\_RDWR: open object for read-write access
  - NB: No O\_WRONLY flag...
- O\_TRUNC: truncate an existing object to zero length
  - Contents of existing object are destroyed



Linux/UNIX IPC Programming ©2025 M. Kerrisk

POSIX Shared Memory

### Creating/opening a shared memory object: shm\_open()

- mode: permission bits for new object
  - RWX for user / group / other
  - ANDed against complement of process umask
  - A Required argument; specify as 0 if opening existing object



Linux/UNIX IPC Programming

©2025 M. Kerrisk

POSIX Shared Memory

16-11 §16.2

#### Sizing a shared memory object

- New SHM objects have length 0
- We must set size using ftruncate(fd, size)
  - Bytes in newly extended object are initialized to 0
  - If existing object is shrunk, truncated data is lost
  - Typically, ftruncate() is called before mmap()
    - But the calls can also be in the reverse order
- Can obtain size of existing object using fstat(fd, &statbuf)
  - st\_size field of stat structure



#### Mapping a shared memory object: mmap()

- Complex, general-purpose API for creating memory mapping in caller's virtual address space
  - 15+ bits employed in *flags*
  - See TLPI Ch. 49 and mmap(2)
- We consider only use with POSIX SHM
  - In practice, only a few decisions to make
    - Usually just length, prot, and maybe offset



Linux/UNIX IPC Programming

©2025 M Kerrisk

**POSIX Shared Memory** 

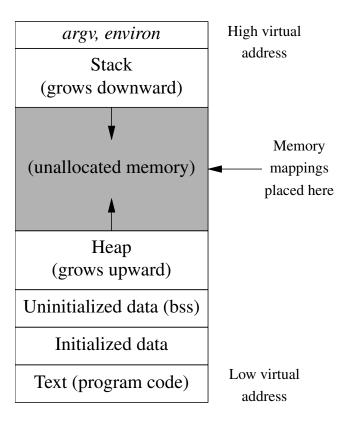
16-13 §16.2

### Mapping a shared memory object: mmap()

- fd: file descriptor specifying object to map
  - Use FD returned by shm\_open()
  - Note: once mmap() returns, fd can already be closed without affecting the mapping
- addr: address at which to place mapping in caller's virtual address space
  - Let's look at a picture...



#### Process memory layout (simplified)



man7.org

Linux/UNIX IPC Programming

©2025 M. Kerrisk

**POSIX Shared Memory** 

16-15 §16.2

### Mapping a shared memory object: mmap()

- addr: address at which to place mapping in caller's virtual address space
  - But, this address may already be occupied
    - Therefore, kernel takes addr as only a hint
    - Ignored if address is already occupied
  - addr == NULL ⇒ let system choose address
    - Normally use NULL for POSIX SHM objects
- mmap() returns address actually used for mapping
  - Treat this like a normal C pointer
- On error, mmap() returns MAP\_FAILED

man7.org

#### Mapping a shared memory object: mmap()

- length: size of mapping
  - Normally should be ≤ size of SHM object
  - System rounds up to multiple of system page size
    - sysconf(\_SC\_PAGESIZE)
- offset: starting point of mapping in underlying file or SHM object
  - Must be multiple of system page size
  - Commonly specified as 0 (map from start of object)



Linux/UNIX IPC Programming

©2025 M Kerrisk

**POSIX Shared Memory** 

16-17 §16.2

### Mapping a shared memory object: mmap()

- prot: memory protections
  - ullet  $\Rightarrow$  set protection bits in page-table entries for mapping
    - (Protections can later be changed using *mprotect(2)*)
  - PROT\_READ: for read-only mapping
  - PROT\_READ | PROT\_WRITE: for read-write mapping
  - Must be consistent with access mode of shm\_open()
    - E.g., can't specify O\_RDONLY to shm\_open() and then PROT\_READ | PROT\_WRITE for mmap()
  - Also PROT\_EXEC: contents of memory can be executed



### Mapping a shared memory object: mmap()

- flags: bit flags controlling behavior of call
  - POSIX SHM objects: need only MAP\_SHARED
  - MAP\_SHARED == make caller's modifications to mapped memory visible to other processes mapping same object



Linux/UNIX IPC Programming

©2025 M. Kerrisk

**POSIX Shared Memory** 

16-19 §16.2

#### Example: pshm/pshm\_create\_simple.c

```
./pshm_create_simple /shm-object-name size
```

Create a SHM object with given name and size



### Example: pshm/pshm\_create\_simple.c

```
size_t size = atoi(argv[2]);
int fd = shm_open(argv[1], O_CREAT | O_EXCL | O_RDWR, S_IRUSR|S_IWUSR);
ftruncate(fd, size);
void *addr = mmap(NULL, size, PROT_READ | PROT_WRITE, MAP_SHARED, fd, 0);
```

- SHM object created with RW permission for user, opened with read-write access mode
- fd returned by shm\_open() is used in ftruncate() + mmap()
- Same size is used in ftruncate() + mmap()
- mmap() not necessary, but demonstrates how it's done
- Mapping protections PROT\_READ | PROT\_WRITE consistent with O\_RDWR access mode



Linux/UNIX IPC Programming

©2025 M. Kerrisk

**POSIX Shared Memory** 

16-21 §16.2

#### Outline

Tatille	
16 POSIX Shared Memory	16-1
16.1 Overview	16-3
16.2 Creating and opening shared memory objects	16-8
16.3 Using shared memory objects	16-22
16.4 Synchronizing access to shared memory	16-31
16.5 API summary	16-41
16.6 Exercises	16-43

#### Using shared memory objects

- Address returned by mmap() can be used just like any C pointer
  - Usual approach: treat as pointer to some structured type
- Can read and modify memory via pointer



[TLPI §48.6]

Linux/UNIX IPC Programming

©2025 M. Kerrisk

**POSIX Shared Memory** 

16-23 §16.3

#### Example: pshm/pshm\_write.c

./pshm\_write /shm-name string

• Open existing SHM object *shm-name* and copy *string* to it



### Example: pshm/pshm\_write.c

- Open existing SHM object
- Resize object to match length of command-line argument
- Map object at address chosen by system
- **Output** Copy argv[2] to object (without '\0')
- SHM object is closed and unmapped on process termination

man7.org

Linux/UNIX IPC Programming

©2025 M. Kerrisk

**POSIX Shared Memory** 

16-25 §16.3

#### Example: pshm/pshm\_read.c

```
./pshm_read /shm-name
```

 Open existing SHM object shm-name and write the characters it contains to stdout



#### Example: pshm/pshm read.c

```
int fd = shm_open(argv[1], O_RDONLY, 0);
struct stat sb;
fstat(fd, &sb);
char *addr = mmap(NULL, sb.st_size, PROT_READ, MAP_SHARED, fd, 0);
                        /* 'fd' is no longer needed */
close(fd);
write(STDOUT_FILENO, addr, sb.st_size);
write(STDOUT_FILENO, "\n", 1);
```

- Open existing SHM object
- Use fstat() to discover size of object
- Map the object, using size from fstat() (in sb.st\_size)
- Write all bytes from object to *stdout*, followed by newline



Linux/UNIX IPC Programming

©2025 M Kerrisk

**POSIX Shared Memory** 

16-27 §16.3

#### Pointers in shared memory

A little care is required when storing pointers in SHM:

- Assuming we let system choose address at which to place SHM (as is recommended practice)
- ⇒ SHM may be placed at different address in each process.
- Suppose we want to build dynamic data structures, with pointers inside shared memory...
  - E.g., linked list
- $\Rightarrow$  Must **use relative offsets**, not absolute addresses
  - Absolute address has no meaning if mapping is at different location in another process



[TLPI §48.6]

### Pointers in shared memory

- Suppose we have situation at right
  - baseaddr is start of shared memory region
  - Want to store pointer to target in
     \*p
- **M** Wrong way:

```
*p = target
```

• Correct method (relative offset):

```
*p = target - baseaddr;
```

• To dereference "pointer":

```
target = baseaddr + *p;
```

man7.org

Linux/UNIX IPC Programming

©2025 M. Kerrisk

**POSIX Shared Memory** 

target -

baseaddr ·

16-29 §16.3

### The /dev/shm filesystem

#### On Linux:

- tmpfs filesystem used to implement POSIX SHM is mounted at /dev/shm
- Can list objects in directory with ls(1)
  - Is -I shows permissions, ownership, and size of each object

```
$ ls -l /dev/shm
-rw----. 1 mtk mtk 4096 Oct 27 13:58 myshm
-rw----. 1 mtk mtk 32 Oct 27 13:57 sem.mysem
```

- POSIX named semaphores are also visible in /dev/shm
  - As small SHM objects with names prefixed with "sem."
- Can delete objects with rm(1)



#### Outline

16	POSIX Shared Memory	16-1
16.1	Overview	16-3
16.2	Creating and opening shared memory objects	16-8
16.3	Using shared memory objects	16-22
16.4	Synchronizing access to shared memory	16-31
16.5	API summary	16-41
16.6	Exercises	16-43

# Synchronizing access to shared memory

- Accesses to SHM object by different processes must be synchronized
  - Prevent simultaneous updates
  - Prevent read of partially updated data
- Semaphores are a common technique
- POSIX unnamed semaphores are often convenient, since:
  - Semaphore can be placed inside shared memory region
    - (And thus, automatically shared)
  - We avoid task of creating name for semaphore



#### Synchronizing access to shared memory

- Other synchronization schemes are possible
  - E.g., if using SHM to transfer large data volumes:
    - Using semaphore pair to force alternating access is expensive (two context switches on each transfer!)
    - Divide SHM into (logically numbered) blocks
    - Use pair of pipes to exchange metadata about filled and emptied blocks (also integrates with poll()/epoll!)



Linux/UNIX IPC Programming

©2025 M. Kerrisk

**POSIX Shared Memory** 

16-33 §16.4

### Example: synchronizing with unnamed semaphores

- Example application maintains sequence number in SHM object
- Source files:
  - pshm/pshm\_seqnum.h: defines structure stored in SHM object
  - pshm/pshm\_seqnum\_init.c:
    - Create and open SHM object
    - Initialize semaphore and (optionally) sequence number inside SHM object
  - pshm/pshm\_seqnum\_get.c:
    - Open existing SHM object
    - Display current value of sequence number
    - (Optionally) increase sequence number value



©2025 M. Kerrisk

POSIX Shared Memory

### Example: pshm/pshm\_seqnum.h

```
#include <sys/mman.h>
#include <fcntl.h>
#include <semaphore.h>
#include <sys/stat.h>
#include "tlpi_hdr.h"

struct shmbuf {    /* Shared memory buffer */
    sem_t sem;    /* Semaphore to protect access */
    int seqnum;    /* Sequence number */
};
```

- Header file used by pshm/pshm\_seqnum\_init.c and pshm/pshm\_seqnum\_get.c
- Includes headers needed by both programs
- Defines structure used for SHM object, containing:
  - Unnamed semaphore that guards access to sequence number



Sequence number

Linux/UNIX IPC Programming

©2025 M. Kerrisk

POSIX Shared Memory

16-35 §16.4

#### Example: pshm/pshm\_seqnum\_init.c

```
./pshm_seqnum_init /shm-name [init-value]
```

- Create and open SHM object
- Reset semaphore inside object to 1 (i.e., semaphore available)
- Initialize sequence number



#### Example: pshm/pshm\_seqnum\_init.c

- Delete previous instance of SHM object, if it exists
- Create and open SHM object
- Use ftruncate() to adjust size of object to match structure
- Map object, using size of structure
- Initialize semaphore state to "available"
  - pshared specified as 1, for process sharing of semaphore
- If argv[2] supplied, initialize sequence # to that value
  - Newly extended bytes of SHM object are initialized to 0

Linux/UNIX IPC Programming

©2025 M. Kerrisk

POSIX Shared Memory

16-37 §16.4

#### Example: pshm/pshm seqnum get.c

```
./pshm_seqnum_get /shm-name [run-length]
```

- Open existing SHM object
- Fetch and display current value of sequence number in SHM object shm-name
- If run-length supplied, add to sequence number



#### Example: pshm/pshm\_seqnum\_get.c

- Open existing SHM object
- Map object, using size of shmbuf structure



Linux/UNIX IPC Programming

©2025 M. Kerrisk

**POSIX Shared Memory** 

16-39 §16.4

#### Example: pshm/pshm\_seqnum\_get.c

- Reserve semaphore before touching sequence number
- Display current value of semaphore
- If (nonnegative) <a href="mailto:argv[2]">argv[2]</a> provided, add to sequence number
  - Sleep during update, to see that other processes are blocked
- Release semaphore

man7.org

#### Outline

16	POSIX Shared Memory	16-1
16.1	Overview	16-3
16.2	Creating and opening shared memory objects	16-8
16.3	Using shared memory objects	16-22
16.4	Synchronizing access to shared memory	16-31
16.5	API summary	16-41
16.6	Exercises	16-43

### **API** summary

```
int shm_open(const char *name, int oflag, mode_t mode);
            // Open or create and open shared memory object
            // Returns file descriptor
int ftruncate(int fd, off_t length);
            // Set size of shared memory object referred to by 'fd'
void *mmap(void *addr, size_t length, int prot,
           int flags, int fd, off_t offset);
            // Map SHM object referred to by 'fd'
int fstat(int fd, struct stat *statbuf);
            // Retrieve 'stat' structure describing SHM object
            // (e.g., statbuf->st_size is object size)
int <u>sem_init(sem_t *sem, int pshared, unsigned int value);</u>
            // Initialize POSIX unnamed semaphore
// Operations on POSIX semaphores:
                          // Increment // Decrement
int sem_post(sem_t *sem);
int sem_wait(sem_t *sem);
```



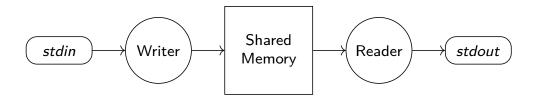
#### Outline

16	POSIX Shared Memory	16-1
16.1	Overview	16-3
16.2	Creating and opening shared memory objects	16-8
16.3	Using shared memory objects	16-22
16.4	Synchronizing access to shared memory	16-31
16.5	API summary	16-41
16.6	Exercises	16-43

#### Exercise

- Write two programs that exchange a stream of data of arbitrary length via a POSIX shared memory object [Shared header file: pshm/pshm\_xfr.h]:
  - The "writer" creates and initializes the shared memory object and semaphores used by both programs, and then reads blocks of data from *stdin* and copies them a block at a time to the shared memory region

    [Template: pshm/ex.pshm\_xfr\_writer.c].
  - The "reader" copies each block of data from the shared memory object to stdout [Template: pshm/ex.pshm\_xfr\_reader.c].



Note the following points:

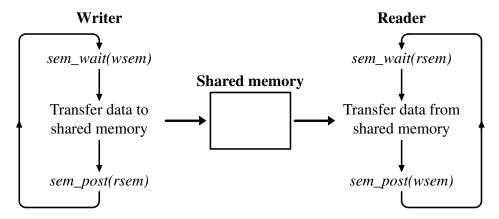
• Use the structure defined in pshm/pshm\_xfr.h for your shared memory.

[Exercise continues on next page]



#### Exercise

• You must ensure that the writer and reader have **exclusive**, **alternating access** to the shared memory region (so that, for example, the writer does not copy new data into the region before the reader has copied the current data to **stdout**). The following diagram shows how two semaphores can be used to achieve this. The semaphores should be initialized as **wsem=1** and **rsem=0**, so that the writer has first access to the shared memory.



(The simplest approach is to use two **unnamed** semaphores stored inside the shared memory object; see the structure definition in pshm/pshm\_xfr.h.)

[Exercise continues on next page]



man7.org

Linux/UNIX IPC Programming

©2025 M. Kerrisk

**POSIX Shared Memory** 

16-45 §16.6

#### Exercise

- When the "writer" reaches end of file, it should provide an indication to the "reader" that there is no more data. To do this, maintain a byte-count field in the shared memory region which the "writer" uses to inform the "reader" how many bytes are to be written. Setting this count to 0 can be used to signal end-of-file. Once it has sent the last data block, the "writer" should unlink the shared memory object.
- Test your programs using a large file that contains random data:

```
$ dd if=/dev/urandom of=infile count=100000
$ ./ex.pshm_xfr_writer < infile &
$ ./ex.pshm_xfr_reader > outfile
$ diff infile outfile
```

There is also a target in the Makefile for performing this test:

```
make pshm_xfr_test
```

[An optional exercise follows on the next page]



man7.org

#### Exercise



Create a file of a suitable size (e.g., 512 MB in the following):

\$ dd if=/dev/urandom of=/tmp/infile count=1000000

Then edit the BUF\_SIZE value in the pshm/pshm\_xfr.h header file to vary the value from 10'000 down to 10 in factors of 10, in each case measuring the time required for the reader to complete execution:

```
$ ./ex.pshm_xfr_writer < /tmp/infile &</pre>
$ time ./ex.pshm_xfr_reader > /dev/null
```

What is the reason for the variation in the time measurements?



Linux/UNIX IPC Programming

©2025 M Kerrisk

POSIX Shared Memory

16-47 §16.6

This page intentionally blank

But, here's a tech talk you might enjoy:

The Tragedy of systemd Benno Rice, linux.conf.au, 2019

(A very amusing talk that is *not* about knocking systemd)

https://www.youtube.com/watch?v=o AIw9bGogo