ECE 3574: Applied Software Design

InterProcess Communication using Shared Memory

Chris Wyatt

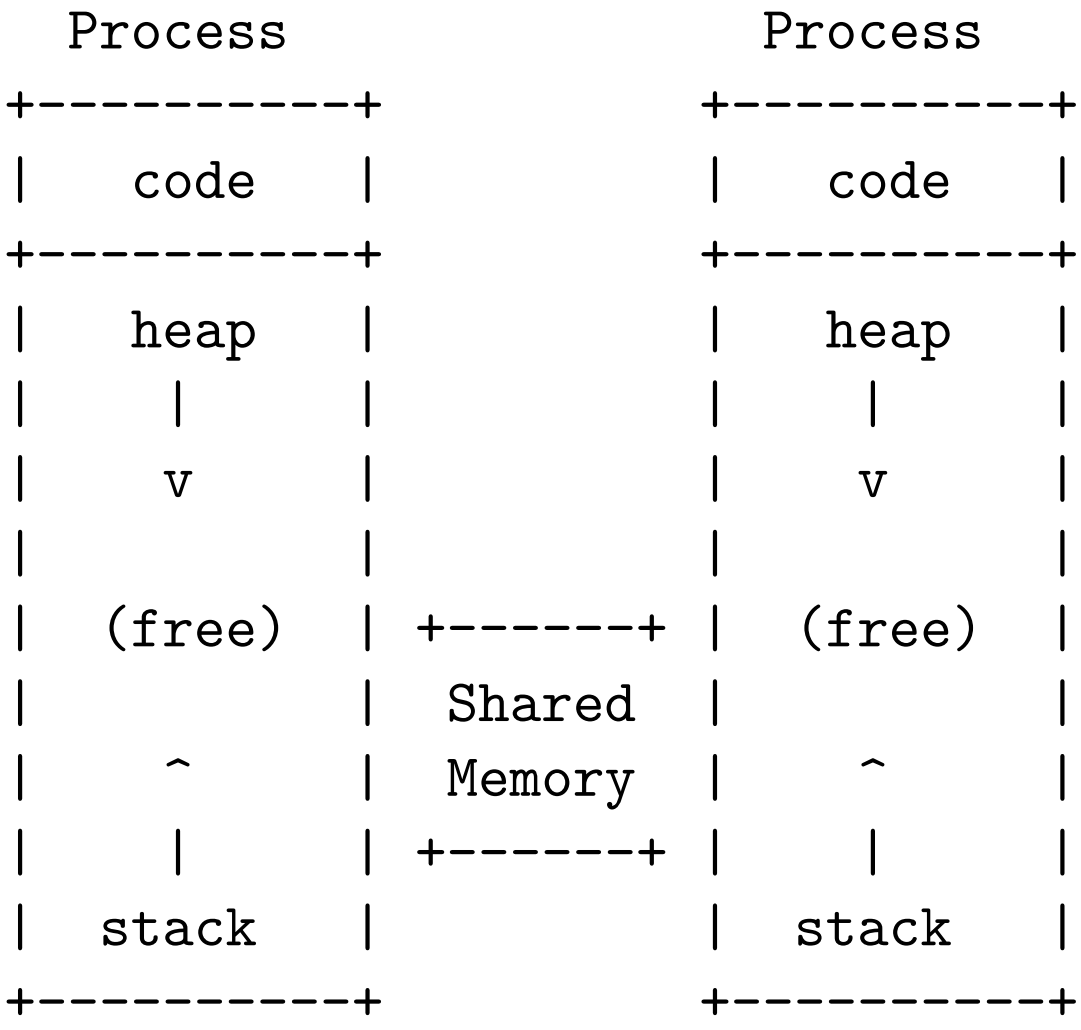
Today we are going to see how processes can communicate using shared memory

- Stack, Heap, and mapped memory segment
- POSIX Shared Memory API
- Windows Shared Memory API
- Cross-platform shared memory using QSharedMemory
- Boost interprocess library
- Exercise

An alternative to IPC with messaging is to share memory

- Rather than send messages over pipes/sockets, we explicitly share memory.
- Fast, but requires explicit *synchronization* !
- This is the model that maps onto threads (which share a heap).
- Can still implement message passing on top of the shared memory

A revision of our process memory model



There are a few shared memory APIs on Unix

We will look briefly at the POSIX shared memory API since it is the most portable. It is a generalization of memory mapped files.

- shm_open(): attach to an existing or create a new shared memory segment
- ftruncate(): size a shared segment
- mmap(): map a mapped object from caller's address space
- munmap(): unmap a mapped object from caller's address space
- close(): close file descriptor returned by shm_open()
- shm_unlink(): remove SHM object name, mark for deletion
- fstat(): retrieve stat structure describing objects

One process creates, the others attach. The kernel guarantees this operation is atomic.

shm_open – open a shared memory object

```
#include <sys/mman.h>
#include <fcntl.h>

int shm_open(const char *name, int oflag, ...);
```

- Each shared segment has a name (called the key)
- The `oflag` argument is an or'd combintation of

```
O_RDONLY open for reading only
O_RDWR   open for reading and writing
O_CREAT  create object if it does not exist
O_EXCL   error if create and object exists
```

ftruncate – truncate or extend a file to a specified length

Here the file is a shared memory segment

```
#include <unistd.h>

int ftruncate(int fildes, off_t length);
```

The first argument of the file descriptor returned from `shm_create()`.
The second argument is the new length in bytes.

mmap – allocate memory, or map files or devices into memory

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

void *
mmap(void *addr, size_t len, int prot,
     int flags, int fd, off_t offset);
```

This call is used next to map the memory into the address space of the process.
Returns a pointer to the beginning of the raw memory segment, the *base pointer*.

Once the mapping is made you can read (and if setup) write to the shared memory.

This required manual placement of objects in memory, offset from the base pointer.
This can be tricky as it requires manually computing pointer offsets based on object size.
We will see how to do this with "placement" new.
Note, you are generally limited to plain-old-data (POD) types unless the objects are allocation aware. For STL containers you can write a custom allocator.

When you are done you unmap the segment

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

int
munmap(void *addr, size_t len);
```

After this access to the shared memory is an access violation and will generate a seg fault.

finally close it, using the file descriptor

```
#include <unistd.h>

int
close(int fildes);
```

Lets look at an example.

See `posix_example/count.cpp`.

The Windows shared memory API is similar

- CreateFileMapping/OpenFileMapping replace shm_create
- MapViewOfFile replaces mmap
- UnmapViewOfFile replaces munmap
- CloseHandle replaces close

Qt provides a cross-platform abstraction QSharedMemory

Shared memory with a *locking mechanism*.
This makes synchronization easier. We will see how to do that ourselves next week.
See example `qt_shared_deque`

Another popular cross-platform IPC library is boost::interprocess

- It provides wrappers for the platform-specific shared memory APIs similar to QSharedMemory.
- It provides shared memory aware containers like vector.
- It also provides a key based object store in the shared memory segment.

The key-based store provides the ability to easily create objects in a shared memory segment, giving a string name to them so that any other process can find, use and delete them from the segment when the objects are not needed anymore.

Exercise 20

See website

Next Actions and Reminders

- Read about Message Serialization