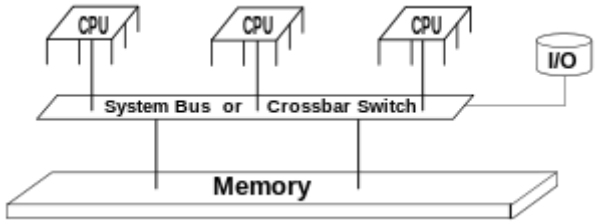
Shared memory

In computer science, **shared memory** is memory that may be simultaneously accessed by multiple programs with an intent to provide communication among them or avoid redundant copies. Shared memory is an efficient means of passing data between programs. Depending on context, programs may run on a single processor or on multiple separate processors.



An illustration of a shared memory system of three processors.

Using memory for communication inside a single program, e.g. among its multiple threads, is also referred to as shared memory.

Contents

In hardware

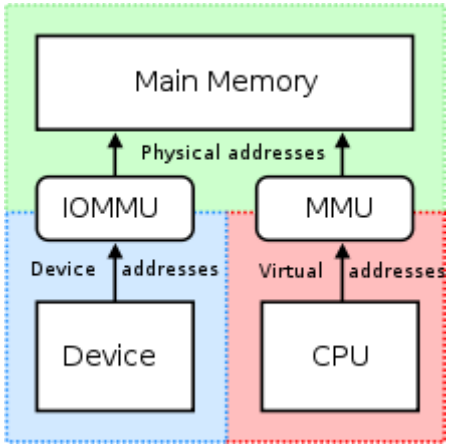
In computer hardware, *shared memory* refers to a (typically large) block of random access memory (RAM) that can be accessed by several different central processing units (CPUs) in a multiprocessor computer system.

Shared memory systems may use:[1]

- uniform memory access (UMA): all the processors share the physical memory uniformly;
- non-uniform memory access (NUMA): memory access time depends on the memory location relative to a processor;
- cache-only memory architecture (COMA): the local memories for the processors at each node is used as cache instead of as actual main memory.

A shared memory system is relatively easy to program since all processors share a single view of data and the communication between processors can be as fast as memory accesses to a same location. The issue with shared memory systems is that many CPUs need fast access to memory and will likely cache memory, which has two complications:

- access time degradation: when several processors try to access the same memory location it causes contention. Trying to access nearby memory locations may cause false sharing. Shared memory computers cannot scale very well. Most of them have ten or fewer processors;
- lack of data coherence: whenever one cache is updated with information that may be used by other processors, the change needs to be reflected to the other processors, otherwise the different processors will be working with incoherent data. Such cache coherence protocols can, when they work well, provide extremely high-performance access to shared information between multiple processors. On the other hand, they can sometimes become overloaded and become a bottleneck to performance.



HSA defines a special case of memory sharing, where the MMU of the CPU and the IOMMU of the GPU have an identical pageable virtual address space.

Technologies like crossbar switches, Omega networks, HyperTransport or front-side bus can be used to dampen the bottleneck-effects.

In case of a Heterogeneous System Architecture (processor architecture that integrates different types of processors, such as CPUs and GPUs, with shared memory), the memory management unit (MMU) of the CPU and the input–output memory management unit (IOMMU) of the GPU have to share certain characteristics, like a common address space.

The alternatives to shared memory are distributed memory and distributed shared memory, each having a similar set of issues.

In software

In computer software, *shared memory* is either

- a method of inter-process communication (IPC), i.e. a way of exchanging data between programs running at the same time. One process will create an area in RAM which other processes can access;
- a method of conserving memory space by directing accesses to what would ordinarily be copies of a piece of data to a single instance instead, by using virtual memory mappings or with explicit support of the program in question. This is most often used for shared libraries and for Execute in place (XIP).

Since both processes can access the shared memory area like regular working memory, this is a very fast way of communication (as opposed to other mechanisms of IPC such as named pipes, Unix domain sockets or CORBA). On the other hand, it is less scalable, as for example the communicating processes must be running on the same machine (of other IPC methods, only Internet domain sockets—not Unix domain sockets—can use a computer network), and care must be taken to avoid issues if processes sharing memory are running on separate CPUs and the underlying architecture is not cache coherent.

IPC by shared memory is used for example to transfer images between the application and the X server on Unix systems, or inside the IStream object returned by CoMarshalInterThreadInterfaceInStream in the COM libraries under Windows.

Dynamic libraries are generally held in memory once and mapped to multiple processes, and only pages that had to be customized for the individual process (because a symbol resolved differently there) are duplicated, usually with a mechanism known as copy-on-write that transparently copies the page when a write is

attempted, and then lets the write succeed on the private copy.

Support on Unix-like systems

POSIX provides a standardized API for using shared memory, *POSIX Shared Memory*. This uses the function `shm_open` from sys/mman.h.[2] POSIX interprocess communication (part of the POSIX:XSI Extension) includes the shared-memory functions `shmat`, `shmctl`, `shmdt` and `shmget`.[3][4] Unix System V provides an API for shared memory as well. This uses shmget from sys/shm.h. BSD systems provide "anonymous mapped memory" which can be used by several processes.

The shared memory created by `shm_open` is persistent. It stays in the system until explicitly removed by a process. This has a drawback that if the process crashes and fails to clean up shared memory it will stay until system shutdown.

POSIX also provides the `mmap` API for mapping files into memory; a mapping can be shared, allowing the file's contents to be used as shared memory.

Linux distributions based on the 2.6 kernel and later offer /dev/shm as shared memory in the form of a RAM disk, more specifically as a world-writable directory (a directory in which every user of the system can create files) that is stored in memory. Both the RedHat and Debian based distributions include it by default. Support for this type of RAM disk is completely optional within the kernel configuration file.[5]

Support on Windows

On Windows, one can use `CreateFileMapping` and `MapViewOfFile` functions to map a region of a file into memory in multiple processes.[6]

Cross-platform support

Some C++ libraries provide a portable and object-oriented access to shared memory functionality. For example, Boost contains the Boost.Interprocess C++ Library[7] and Qt provides the QSharedMemory class.[8]

Programming language support

For programming languages with POSIX bindings (say, C/C++), shared memory regions can be created and accessed by calling the functions provided by the operating system. Other programming languages may have their own ways of using these operating facilities for similar effect. For example, PHP provides an API to create shared memory, similar to POSIX functions.[9]

See also

- Distributed memory
- Distributed shared memory
- Shared graphics memory
- Heterogeneous System Architecture
- Global variable
- Nano-threads

- Execute in place
- Shared register
- Shared snapshot objects
- Von Neumann Architecture Bottleneck

References

1. El-Rewini, Hesham; Abd-El-Barr, Mostafa (2005). *Advanced Computer Architecture and Parallel Processing*. Wiley-Interscience. pp. 77–80. ISBN 978-0-471-46740-3.
2. Documentation of shm_open (http://www.opengroup.org/onlinepubs/007908799/xsh/shm_open.html) from the Single Unix Specification
3. Robbins, Kay A.; Robbins, Steven (2003). *Unix systems programming: communication, concurrency, and threads* (https://archive.org/details/unixsystemsprogr0000robb) (2 ed.). Prentice Hall PTR. p. 512 (https://archive.org/details/unixsystemsprogr0000robb/page/512). ISBN 978-0-13-042411-2. Retrieved 2011-05-13. "The POSIX interprocess communication (IPC) is part of the POSIX:XSI Extension and has its origin in Unix System V interprocess communication."
4. Shared memory facility (http://www.opengroup.org/onlinepubs/007908799/xsh/sysshm.h.html) from the Single Unix Specification.
5. Christoph Rohland; Hugh Dickins; KOSAKI Motohiro. "tmpfs.txt" (https://www.kernel.org/doc/Documentation/filesystems/tmpfs.txt). kernel.org. Retrieved 2010-03-16.
6. Creating Named Shared Memory (http://msdn.microsoft.com/en-us/library/windows/desktop/aa366551%28v=vs.85%29.aspx) from MSDN.
7. Boost.Interprocess C++ Library (http://www.boost.org/doc/libs/1_48_0/doc/html/interprocess.html)
8. "QSharedMemory Class Reference" (http://doc.qt.io/archives/qt-4.8/qsharedmemory.html).
9. Shared Memory Functions in PHP-API (http://www.php.net/manual/en/ref.shmop.php)

External links

- IPC:Shared Memory (http://www.cs.cf.ac.uk/Dave/C/node27.html) by Dave Marshall
- Shared Memory Introduction (http://www.kohala.com/start/unpv22e/unpv22e.chap12.pdf), Ch. 12 from book by Richard Stevens "UNIX Network Programming, Volume 2, Second Edition: Interprocess Communications".
- SharedHashFile (https://github.com/simonhf/sharedhashfile), An open source, shared memory hash table.