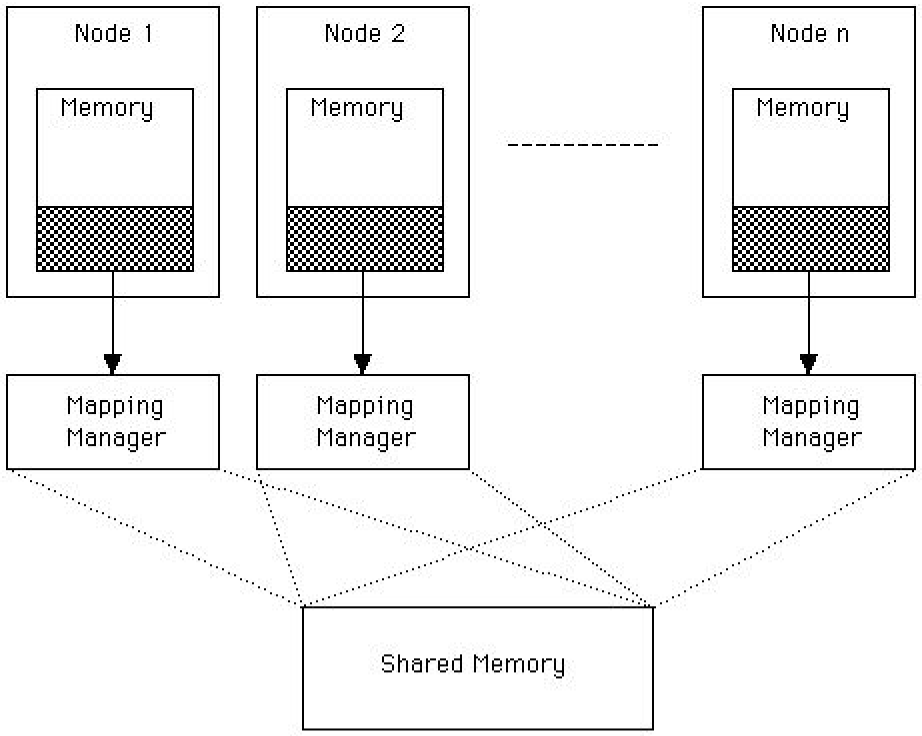
Distributed Systems (CS632)
Project Report: Distributed shared memory

Shivansh Rai (14658)

Objective

This project aims to develop a distributed shared memory architecture where physically separated memories can be addressed as one logically shared address space.
The initial setup will involve two processes running on different machines (nodes). If time permits, an attempt will be made to extend the algorithm to work with an arbitrary number of nodes.



Ref. Advanced Concepts in Operating Systems by Singhal & Shivaratri

Implementation details

The aim is to create an illusion that two processes on different machines share the same address space. This project models a two process system as a proof of concept for a n process system.
The two processes will be referred to as master and slave. Initially, the start address of the shared memory segment is fixed at 1 GB (2^{32}). The ending address is decided based on the number of pages (provided by the user) which are to be allocated in the shared memory system. Initially, the master process owns the first half of the shared memory and the slave process owns the second half. This is achieved via the mprotect(2) system call with appropriate read and write protection. If a process tries to access a page which doesn't belong to its address space, a page fault is generated. This in turn triggers a page fault handler which is responsible for

locating and migrating the faulting page from the relevant machine. The details of migration algorithm have been discussed below.

Migration algorithm

In the migration algorithm, the data is always migrated to the site where it is accessed. This is a “single reader/single writer” protocol, since only the threads executing on one host can read or write a given data item at any one time.
Instead of migrating individual data items, data is typically migrated between servers in multiples of pages to facilitate the management of data. The advantage of this algorithm is that no communication costs are incurred when a process accesses data currently held locally.
A second advantage of the migration algorithm is that it can be integrated with the virtual memory system of the host operating system if the size of the block is chosen equal to the size of a virtual memory page (or a multiple thereof). If a shared memory page is held locally, it can be mapped into the application's virtual address space and accessed using the normal machine instructions for accessing memory. An access to a data item located in data blocks not held locally triggers a page fault so that the fault handler can communicate with the remote hosts to obtain the data block before mapping it into the application's address space. When a data block is migrated away, it is removed from any local address space it has been mapped into.

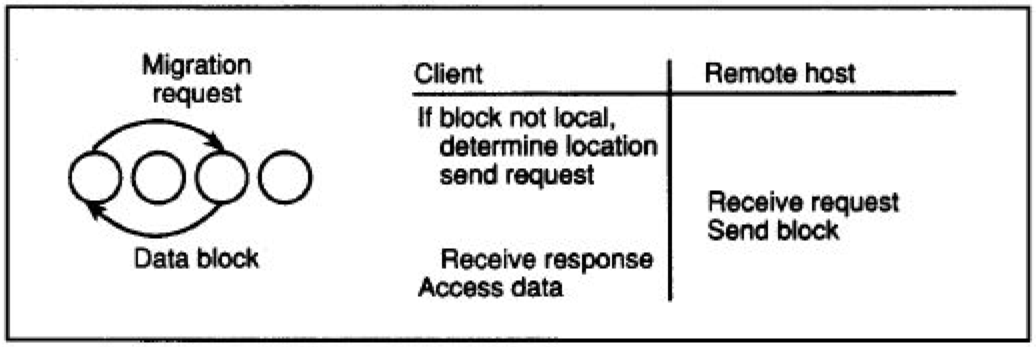

Migration request
Data block
Client
If block not local, determine location send request
Receive response Access data
Remote host
Receive request Send block

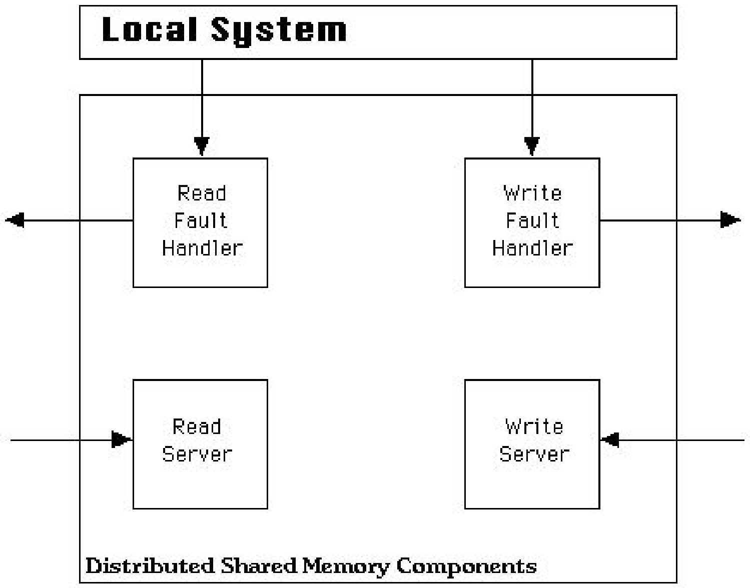
Figure 3. The migration algorithm.

When a process on a local machine attempts to write to a memory location (in a page) that resides on a remote machine, the following steps will be performed -

- the process on local machine will request the remote machine a copy of the page
- the remote machine protects (PROT_READ) its page (mprotect(2)), and sends a copy over socket. The read-only protection is done for the duration in which the page is being sent. Once the page has been transferred, the remote machine relinquishes its control over the page by marking it as PROT_NONE.
- the local machine maps (mmap(2)) the page's data into its own address space
- this page now belongs to the local machine which marks it as PROT_WRITE.
- any reads/writes to the corresponding page on the remote machine will incur a page fault, and the handler routine will follow the above steps
- to prevent race conditions, a per page mutex is maintained

The above model has been implemented for a two-process system. For a n-process system, each process will have control of exactly $\frac{1}{n}$ of the total shared memory segment initially.

Components of distributed shared memory



Page fault handler

A fault handler is a process which handles page faults. When there is an invalid memory access in the local machine, the fault handler will be invoked. The responsibility of the fault handler is to make a request to an appropriate server (on a different node) which contains the desired memory page(s).

Server

A server services a fault handler request. They contain the information regarding which machines owns the relevant memory pages and can fetch and deliver them to the requesting fault handler.

References

- https://en.wikipedia.org/wiki/Distributed_shared_memory
- http://courses.cs.vt.edu/~cs5204/fall00/distributedSys/amento/dsm.html
- https://ieeexplore.ieee.org/document/7322749/?part=1