Distributed Shared Memory

Abhijeet Srivastava (2020202011)
Rishi Tripathi (2020202003)
Shubham Singh (2020202002)

Shared Memory –

Shared memory is used for inter-process communication on a single system. A memory is shared across multiple processes that can read and write on the memory.

Distributed Shared Memory –

In a distributed system, the memory shared between processes cannot exist on a single system. Each process has some part of the shared memory.

Advantages –

- It helps in allocating a huge amount of memory that is not feasible to have on a single system.
- Data is locally available rather than fetching data from another entity for each access.
- Data is only moved when needed by a process.

Implementations of Distributed Shared Memory –

Distributed Shared Memory can be implemented in the following ways –

- **Central Server Algorithm** – All the requests to read or write data is made to a central server which relays it to the required process. The actual copy of data remains with the process that created it. Other processes can have a read copy and write requests are fulfilled by updating and sending back the acknowledgement of data.
- **Migration Algorithm** – There is no single entity that relays the data. Each process when requests the data (both read and write requests), the data is passed on to the requesting process. It involves a lot of movement of data even if the data is only read.
- **Read Replication Algorithm** – It is similar to Migration Algorithm, with the distinction that data is moved only if there is a write request. For read request, a read copy is shared with the requesting process. For write request, the data is moved to requesting process and updated there. All other copies of data should be invalidated to maintain consistency.
- **Full Replication Algorithm** – In this algorithm, data resides on multiple processes at once. data needs to be Each process can update data. To maintain consistency, the updates to same sequenced.

Our Implementation –

We have implemented Read Replication Algorithm, as it has advantages over Migration and Central Server algorithm in terms of time to read data.

We have not chosen full replication algorithm, as maintaining many writable copies of data can have inconsistency and would require a lot of memory.

We have created a python library that can be imported into python scripts. The library has functions to read and write data. We are using read and write locks to maintain consistency and while writing (updating) a variable all other copies are invalidated.

- Read locks are shared locks, i.e, more than 1 processes can acquire read lock simultaneously.
- Write locks are exclusive locks. Only 1 process can acquire write lock at a time.

Read Mechanism –

The process first acquires read lock. If the process has a read copy in its cache, it is read directly. Otherwise, it asks all the processes for the variable/object to find owner of the variable/object and fetches a read copy from the owner. It then saves read copy in its cache. The owner's write copy is changed to read copy. At the end, it releases the read lock.

Write Mechanism –

The process that has updated the variable most recently is the owner of the variable. The process that has to write on a variable/object, first acquires write and read lock. If the process has the write copy, it can directly update the variable. If the process has read copy or no copy at all, it first invalidates all other processes' copies, and writes the variable which creates a write copy. The process then releases the write and read lock.

Implementation code details and Steps to run are available in README.md.