Northeastern University, Computer Systems CS 5600 Fall 2016
Final Project

Distributed Shared Memory for xv6

Abhay Kasturia, Nakul Camasundaram, Rashmi Dwaraka

December 1, 2016

Contents

Chapter 1

INTRODUCTION

In order to increase computing power, the use of mutliprocessors have become quite popular. Two kinds of parallel processors i,e tightly coupled shared-memory multiprocessors and distributed-memory multiprocessors (multicomputer). However, tightly coupled multiprocessors suffer with bottleneck accessing main memory, as there is a common bus which becomes a serialization point. Distributed-memory multiprocessors, however, do not suffer from this drawback.

Distributed-memory mutliprocessing system consist of a collection of independent computers connected by a high-speed interconnection network. Distributed Shared Memory (DSM) is a resource management component of a distributed operating system that implements the shared memory model in distributed systems, which have no physically shared memory. The shared memory model provides a virtual address space that is shared among all computers in a distributed system. The shared-memory abstraction gives these systems the illusion of physically shared memory and allows programmers to use the shared-memory paradigm.

In order to understand various design implications of building Shared memory paradigm, we propose to add the Shared memory support to xv6 Operating system.

Chapter 2

SOLUTION PROPOSAL

Each processor in a multicomputer has its own private memory which it alone can write and read. These processors can be connected using standard networking technology. Processors in multicomputer must communicate using shared memory and provide an abstraction layer to the programmer. The Distributed Shared Memory model simulates true physical shared memory on a loosely-coupled system. A number of processes share a single address space. This address space is divided into pages, which are distributed among the processes. The Processes either have no read or write access to a page. Read-pages can be replicated on multiple processors to reduce access times. The system provides a coherent address space: a read operation always returns the value of the most recent write to the same address. Mutual exclusion synchronization can be implemented by locking pages.

The solution to managed proposed in this project is to divide the shared address space into fixed-size pages, which are distributed among the processors. The DSM system coordinates the movement and validity of the pages. Pages with read-only access can be replicated at multiple processors to reduce access times. An write invalidation scheme is used to keep the address space coherent. For example, when a process tries to write on a page, all other copies of that page are first invalidated, then the write is permitted. To synchronize multiple requests for the same page, each page is owned by a specific process (which may change in time). This owner process has a valid copy of the page, and all requests for that page are sent to it. To be able to keep the address space coherent, the DSM system keeps track of the owner process and of all the processes containing a valid copy.

We choose to implement the DSM component in a distributed manner, where each processor will have its own DSM manager. In the event of a page fault, the DSM handler associated with the faulting process broadcasts a request for the page. All other DSM handlers receive this broadcast. The owner of the faulting page will send the page to the requester using an RPC. From this point on, the requesting process is the new owner, and it will respond to the next request for this page.

A write-request broadcast causes all DSM handlers to invalidate their copy of the page, so no copy-lists are needed. This scheme only works because each process gets the broadcasts in the same order, and no broadcasts are lost. All DSM handlers have the same view of the system and there can be no confusion over which process is the owner and which processes have a valid copy.

In this system, pages can be locked too. The DSM handlers try to handle a broadcast affecting a locked page as soon as possible. For instance: on receiving a write-request for a read-locked page, the DSM handler will send the page right away. The invalidation is done as soon as the page is unlocked again. A page fault (read and write) takes one broadcast and one RPC to complete. When a read-copy is upgraded to a write-copy, no RPC is used, Page requester will have the copy and be the owner of the page.

Chapter 3

EXPERIMENTS AND RESULTS

We propose to test the system by building a matrix multiplication test suite to show the performance impact of shred memory paradigm.

Chapter 4

REFERENCES

(1) http://www.cdf.toronto.edu/ csc469h/fall/handouts/nitzberg91.pdf

(2) http://css.csail.mit.edu/6.824/2014/papers/li-dsm.pdf

(3) https://github.com/mit-pdos/xv6-public