Unix System Programming - Chapter 17

Neal Nelson

The Evergreen State College

Apr 2010

USP Chapter 17 - The Not Too Parallel Virtual Machine Project

- Section 17.1 - Workstation-level Virtual Machines
- Section 17.2 - Virtual Machine Programming Libraries: MPI, PVM, NTPVM
- Section 17.3 - NTPVM Dispatcher Project Overview
- Section 17.4 - IO and Testing Framework for the NTPVM Dispatcher
- Section 17.5 - Dispatcher handles Single task, No Input - Project Milestone
- Section 17.6 - Dispatcher handles Sequential Tasks - Project Milestone
- Section 17.7 - Dispatcher handles Concurrent Tasks - Project Milestone
- Section 17.8 - Packet Communication, Broadcast and Barriers - Extra Credit

Virtual Machines

- Grace Hopper: The way to pull a heavier load is not to grow a bigger ox but to hitch more oxen to the load.
- Seymour Cray: Which would you rather have plowing a field, two strong oxen or 1024 chickens?
- Granularity of processing and degree of parallelism; communication bandwidth (really throughput and response time)
- Workstation-level collection of hetereogenous machines is a practical option right now. Process granularity of processing. TCP/IP packet level of communication.

WVM - Workstation Virtual Machine

- Virtual Machine (VM) concept on a workstation collection of machines - let's call it WVM for Workstation Virtual Machine.
- VM libraries for a WVM: MPI - Message Passing Interface, PVM Parallel Virtual Machine.
- Cross-platform abstractions for *Computation*, *Task*, and *Message Passing* provided by the library. PVM library presents a VM abstraction and hides the individual workstations.
- It is hard to develop programs on WVMs using raw PVM or MPI
- Higher level programming services can be built on top of the PVM or MPI libraries.
- Grid Computing Architectures are evolving for higher level parallel programming abstractions on top of PVM or MPI level libraries (USP p582).

Not Too Parallel Virtual Machine Library (NTPVM)

- Build our own prototype PVM-like framework with Computation, Task, and Message Passing abstractions for progamming computations on a WVM.
- We're not going to *use* a PVM framework, we're going to build one.
- Figure 17.1 illustrates a PVM application built on the three main abstractions (plus IO).
- Task is really Unix process. Computation is a collaborating collection of tasks.

VM to Host mapping

- Figure 17.2 illustrates a mapping from the PVM abstractions to underlying host resources for computation and communication.
- Notice computations can be spread over distinct hosts
- Notice that all task interaction on each host is mediated by the PVM daemon pvmd
- Notice that the PVM daemons all talk directly to each other and tasks on their own host but only indirectly to tasks on other hosts.
- The PVM daemons act as communication brokers.
- Claim: the PVM architecture described here is analogous to distributed services offered by the Common Object Request Broker Architecture (CORBA) of the object oriented world.

VM to Host mapping - the PVM daemon

- So Figure 17.2 illustrates the central role of the **pvmd** PVM deamon on each host platform in routing messages, mapping tasks to computations, and keeping track of computational abstractions, that transcend individual host machines.
- The NTPVM project is to build a **dispatcher** that prototypes the PVM daemon.
- The NTPVM dispatcher creates and manages tasks on a single host, routes inter-task messages and maintains the mapping of tasks to computations.
- The NTPVM does not, as far as I can tell, provide any services to support the computation level of abstraction (eg, inter-computation communication).
- That is, there seems to be no common IO model for a computation as a whole, as implied by the arrows in Figure 17.1.

The NTPVM Dispatcher

- There is one NTPVM Dispatcher per host, just like in the PVM architecture.
- The NTPVM Dispatcher receives requests through its stdin and responds through its stdout.
- The NTPVM Dispatcher creates and manages communication with all the tasks on a particular host.
- The NTPVM project specification seems to be designed to support multiple dispatchers on a distributed collection of machines, just like the PVM of Figure 17.2 (USP p584-585).
- The stdin and stdout of the dispatcher can be redirected to network communication ports to provide the inter-host communication.
- The project in this chapter is simplified to a single dispatcher that controls all the computations and the tasks are all on one machine.
- See Figure 17.3 for the schematic of the NTPVM dispatcher.

The NTPVM Dispatcher Communication Architecture

- The dispatcher mediates all task creation and communication across multiple distributed hosts.
- Task communication uses *packets* that are read and then routed or acted on by the dispatcher.
- The dispatcher communicates with the outside world (other dispatchers) by reading packets from its standard input and writing packets to its standard output.
- Tasks can communicate across hosts using packets via their per-host dispatchers.
- For this project we use only one dispatcher and send hand-constructed packets from our controlling terminal
- The dispatcher keeps a table of tasks that it managing on its host.
- The dispatcher has a packet protocol that ii uses to talk with the other dispatchers.

The NTPVM Per-host Dispatcher-Task Communication

- See Figure 17.4 for the schematic of dispatcher-task communication on a particular host.
- The dispatcher on a particular host creates tasks as child processes and sets up pipes for communication through stdin and stdio of the child.
- Tasks can communicate across hosts via their per-host dispatchers, but this is not implemented in this project.
- The dispatcher communicates with the outside world by reading packets from its standard input and writing packets to its standard output.
- Dispatchers communicate with their own host tasks by ASCII streams through stdin and stdio.
- later versions of the project push packet processing into the individual tasks.

Packet Types

- There are six types of packets: NEWTASK, DATA, BROADCAST, DONE, TERMINATE, and BARRIER.
- Packets originate in tasks of computations and are targeted at tasks of computations.
- As brokers, dispatchers act on each type of packet in a specific way.
- NEWTASK packets ask dispatchers to create new tasks in computations.
- DATA packets ask dispatchers to route data to specific tasks in computations.
- DONE packets ask dispatchers to mark tasks of computations as completed and no longer receiving data.
- TERMINATE packets kill eliminate communication pipes and kill tasks - presumably after they are DONE.
- We'll worry about BROADCAST and BARRIER later.

Project Phases

- Phase 1 - 17.4 Setup dispatcher and task IO and testing. This is the hard part, but they give us all the code.
- Phase 2 - 17.5 Single task with no output - handle NEWTASK and DATA outgoing.
- Phase 3 - 17.6 One task at a time - handle NEWTASK, DATA, and DONE packets.
- Phase 4 - 17.7 Multiple tasks and computations with NEWTASK, DATA, and DONE.
- Phases 5-7 we'll worry about later.

Done

- Done.