Comparison and Simulation of IPC in Akaros and Tornado

Sheetal Shalini (14CO142), Navya R S(14CO126)

February 22, 2016

Chapter 1

Basic Comparison

IPC in Akaros

1. Shared memory in intra-core and message passing in inter-core
2. Unbounded Concurrent Queue (UCQ)
3. Event messages sent by kernel and received by process.
4. Infinite number of event messages can be sent from the kernel to the processes.

IPC in Tornado

1. Mssage passing between server and client through microkernel
2. Server client implementation
3. Request sent by client to server, via the microkernel.
4. Any no.of requests can be sent to the servers.

Chapter 2

IPC in Akaros

2.1 Implementation of UCQ(Unbounded Concurrent Queue), used for Inter Process Communication in Akaros.

Akaros is a research operating system designed for single-node, large-scale SMP and many-core architectures. Inter-process communication (IPC) in Akaros is of two types: intra-core and inter-core. Within a core, the kernel communicates with the processes via UCQs (Un-bounded Concurrent Queue), which are linked, mmapped pages of arrays of event messages that operate on the concept of shared memory. Between cores, communication is done via message passing alone, using mail-boxes. An event queue consists of an event mailbox and various configuration parameters. A process can request a payload and use the UCQ, or request a bit and use the bitmap. Each core has two mailboxes (1 private, 1 public) for various messages. The primary feature of Akaros is a new process abstraction called the "Many- Core Process" (MCP). The MCP embodies transparency, application control of physical resources, and performance isolation. It consists of virtual cores pinned to physical cores.

Event delivery for MCPs must be:

1. user-controlled
2. reliable
3. fast, scalable and safe
4. for either a bit or a payload element

The first data structure created for event delivery in Akaros was called Bounded Concurrent Queue, or BCQ. It was a multi-producer, multi-consumer, fixed-size, ring buffer of elements. It operated on shared memory. However, it did not satisfy the requirements easily - the fact that BCQs could overflow and hence allow

messages to go missing made related programming aspects difficult (although not impossible).This led the developers of Akaros to create an improved data structure, called Unbounded Concurrent Queues, or UCQs. This is the data structure used in intra-core communication to deliver events from the kernel to the processes. It is also used as the mailbox buffer in inter-core communication. The amount of messages the queue can hold is "unlimited", unless user memory runs out. UCQs use linked, mmapped (memory mapped) pages of arrays holding event messages. The basic aspects of UCQs are:

1. Messages from the kernel to the processes
2. Multiple producers, multiple consumers
3. Messages are in the form of a queue (FIFO)
4. No practical bound on the number of messages - queue expanded as and when necessary.

   We have demonstrated multiple pages, with multiple locations in them, where the event messages can be kept. When the kernel wants to send event messages to the UCQ, they are kept in the first free location available. We have considered 5 locations to be available in every dynamically allocated page. The number can vary. Once all the locations in a page have been occupied, a new page is created and memory is dynamically allocated to it. The kernel can then continue sending event messages to these new locations. When a process wants to receive event messages, we first check if the process number is a valid one, and then we also check if there is atleast one event message available in the UCQ. If there isn't, then we wait for the kernel to send an event message. But if there is, then the first available event message is received by the process, and that location is made empty. So the next time the kernel wants to send a message, it is loaded into that previously emptied location. Infinite no.of pages can be dynamically created, depending on the no.of event messages that the kernel wants to send. This is how UCQ's are implemented!

Chapter 3

IPC in Tornado

Tornado uses protected procedure calls (PPC) in order to perform inter process communication among processors within a cluster. The PPC is based upon the following model

(a) the client requests are always handled on the local processor

(b) there are as many threads in the server as the number of client requests.

Separate worker processes are used in the server to service the client calls. Worker processes are created dynamically as needed.

Following are the factors for chosing separate worker processes. First, it simplifies exception handling. Although we would like a PPC to appear similar to a traditional procedure call, we would prefer its failure modes to more closely follow those of a message exchange (for example, an exception raised against the client while executing in the server should not effect the server). The more pragmatic factor is that having a separate worker process service PPC calls fits more naturally with the traditional process model upon which our operating system is based.

Following benefits are obtained by this approach. a) there is no sharing of data, hence cache coherence traffic is eliminated, b) no locking is required because the resources are exclusively owned by the local processor(apart from disabling interrupts, which is a natural part of system traps). In addition, such a multiprocessor IPC facility must preserve the locality and concurrency of the applications themselves so that the high performance of the IPC facility can be fully exploited.The model dictates that requests are always handled on the same processor as the client, allowing the server to keep state associated with the client's requests local to the client. By eliminating memory, network, and lock contention, the PPC facility imposes no constraints on concurrency for the system.