Joe Hastings

Toliver Jue

6.191 Final Project Proposal

December 7, 2001

Libasync-mp: A Multiprocessor Aware Asynchronous Programming Library Based Upon Libasync

This paper describes extensions to a project Joe Hastings conducted along with Mark Rosen, Benjamin Pick, and William Friedman for 6.824, taught by Professor Robert Morris of the PDOS group at LCS, of the same name. Mark and Joe co-developed the code relevant to this paper while Benjamin and William worked on other aspects of that project. Mark, Toliver, and Joe are interested in extending this project or working on complementary projects under Professor Morris for our MEng theses. We feel that we obtained very promising results from our project and have made plans to discuss our results with Professor Morris. However, we have not received any commitments from Morris regarding his acceptance of the project as our advisor.

Toliver Jue worked on an independent project for 6.824, but has the same interests as Mark and Joe. Thus, we three would like to form a larger MEng project on asynchronous programming.

Overview of PDOS and libasync

The Parallel and Distributed Operating System Group (PDOS) is a multifaceted group with research in many topics related to Operating System architectures and services, networking, and languages. All these topics are crucial in the building of large-scale parallel and distributed systems.

Projects of the group include Click, SFS, and libasync. Click is a flexible and modular router that runs on many commodity systems. Different configurations are easily composed with small elements such as queues, packet schedulers, and dropping policies that are "clicked" together. For distributed systems, the re-configurability of Click proves important for components that may fail and configurations that need to adapt. The Self-Certifying Filesystem (SFS) is a secure filesystem with decentralized control. This filesystem is important to distributed systems that will need to talk to each other and access each other's

data. Finally, libasync is an asynchronous library that can be used by programmers to simplify many of the parallel constructs they may need to use.

Libasync is the focus of our Masters theses. Work done so far for a final project in a course (6.824) has been to make a multiprocessor version of Libasync, called Libasync-mp. Constructs added were the ability to more fully utilize multiple processors for asynchronous events to improve the efficiency of executing code.

Technologies Used in Libasync-mp

Unlike other MEng projects presented in 6.191, this project does not use technologies specific to the PDOS research group. Due to the nature of PDOS, their technology can be thought of as knowledge of how to use programming techniques to develop distributed and parallel programs. However, this project does take advantage of two cutting-edge technologies not developed at PDOS: the FreeBSD kevent API and POSIX Next Generation Threads. These two tools were essential to creating libasync-mp and we will discuss each of them in this section.

The FreeBSD kevent API is a replacement to the select/poll method traditionally used to write asynchronous code. The select / poll system calls present in all modern Unix systems (not limited to FreeBSD) allow a program to register the intent to be notified by the kernel whenever a particular file-descriptor (a unique identifier given to each file and socket a program uses) is ready to be read to or written from. However, this API is not re-entrant with respect to kernel threads, meaning that a multi-threaded program cannot safely use these calls without taking extreme care. While select / poll work great in the original libasync, the design of libasync-mp requires a thread-safe alternative. The actual implementation of kevent is far beyond the scope of this paper, but more information is provided in the report Joe prepared for 6.824.

POSIX Next Generation Threads are a means of providing M : N, user : kernel, thread mappings. They are not a technology per se but rather a POSIX standard library meaning that all modern Unix vendors must support the API if they claim to be POSIX compliant. To briefly describe what this means, a user-thread is an abstraction of a virtual process that exists only in the user-space portion of an

application's address space. The kernel, the piece of the OS responsible for allocating processes to different CPUs in a multiple-processor system (among many-many other things), thus is not aware of them. A kernel thread, however, is a virtual process that the kernel schedules directly. User-threads are efficiently handled by a threading library such as pthreads (POSIX threads) while kernel threads are very expensive to maintain because kernel operations are in-general orders of magnitude more expensive than user-level operations. Thus, POSIX Next Generation threads allow for the efficiency of user-level threads to take advantage of multiple processors by mapping M user-threads to N kernel-threads (typically N is the number of processors). POSIX Next Generation threads are important for the project because unlike many unixes (such as Linux), FreeBSD4.x does not support native kernel threads, meaning that it would be impossible for us to write code using multiple processors that shared an address space. However, the POSIX Next Generation library takes advantage of a hack to FreeBSD that uses a specialized function called rfork to give the illusion of kernel threads to the OS. Again, the specifics of these actions are well beyond the scope of this paper.

Specific Extensions of Libasync-mp

There are many challenges remaining to produce a multiple processor aware version of libasync. Libasync-mp currently does not support the same semantics or API of the original libasync due to time-constraints and complexity issues. This means that while it provides the same services, existing code must be re-written to take advantage of multiple processors. One possible MEng project in-and-of-itself would be the incorporation of our logic into libasync itself rather than our code existing in a separate library. Mark Rosen will most likely do this project with substantial help from myself.

Even after libasync-mp uses the same semantics as libasync, there is a complex notion of which programs can be safely re-linked with our library without any re-coding. In general, the move from a single processor system to multiple-processors introduces a plethora of synchronization problems, many of which are totally unexpected to the programmer. For this reason, we are interested in providing the synchronization support that libasync-mp will need to offer the programs that must be re-coded to work in a multiprocessor environment. Several of these challenges include:

1. The support of "virtual processor numbers" for different segments of code, indicating which code is safe/unsafe to run concurrently.

2. The support of "mutexed callbacks" which is beyond the scope of this paper, but provide a means of putting locks around specific callback functions as they are registered.
3. The possible writing of code to statically/dynamically detect synchronization errors either from source code or run-time analysis. Ultimately, such a solution might allow programs to take advantage of libasync-mp without major rewriting of the problem-areas, or at least to help the programmer find such areas.

The full-scale production of libasync-mp to obey libasync semantics and provide support for synchronization protection is a daunting task with more than enough room for multiple MEng projects.

Possible Timeline

Assuming that we can obtain an advisor, we would like to begin expanding upon the code Joe and Mark wrote for 6.824 immediately and to make substantial progress during IAP 2002. We do not plan on having a great deal of time to work on the project during the Spring semester although we will be taking classes that will aid in our ability to perform the project. We then plan on working during the summer of 2002 and hopefully having a somewhat debugged version of the library produced by the beginning of the fall semester. This library will require a great deal of testing before it can be safely applied to large-scale projects such as SFS that take advantage of the existing libasync. Hopefully, testing during the fall semester of 2002 will facilitate a final re-coding during IAP 2003 and the project will be completed during Spring 2003.

Risks and Mitigations

The greatest risk facing the project right now is the current lack of advisor support. The PDOS group is interested in pursuing the project but has not yet decided to grant the project to our particular team. There are several graduate students at PDOS interested in pursuing similar projects themselves. Apart from lacking an advisor, the project's only other risk is our inability to produce a library with libasync-like semantics. Also, the synchronization issues may become too large to be addressed by an MEng level project. However, our initial short-term success during the 6.824 project has given us hope that we can reach the project's full goals.