A Machine-Independent Port of the SR Language Run Time System to NetBSD Operating System

Ignatios Souvatzis
University of Bonn, CS Dept., Chair V
<ignatios@cs.uni-bonn.de>

29th September 2004

1 Introduction

SR (synchronizing resources)[1] is a PASCAL – style language enhanced with constructs for concurrent programming developed at the University of Arizona in the late 1980s[2]. MPD (presented in Gregory Andrews' book about Foundations of Multithreaded, Parallel, and Distributed Programming[3]) is its successor, providing the same language primitives with a different syntax.

The run-time system (in theory, identical) of both languages provides the illusion of a multiprocessor machine on a single single- or multi- CPU Unix-like system or a (local area) network of Unix-like machines.

Chair V of the Computer Science Department of the University of Bonn is operating a laboratory for a practical course in parallel programming consisting of computing nodes running NetBSD/arm, normally used via PVM, MPI etc.

We are considering to offer SR and MPD for this, too. As the original language distributions are only targeted at a few commercial Unix systems, some porting effort is needed, outlined in the SR porting guide[4].

The integrated POSIX threads support of NetBSD-2.0 should allow us to use library primitives provided for NetBSD's phtread system to implement the primitives needed by the SR run-time system, thus implementing 13 target CPUs at once and automatically making use of SMP on VAX, Alpha, PowerPC, Sparc, 32-bit Intel and 64 bit AMD CPUs.

This paper describes work in progress.

2 Generic Porting Problems

Given the age of the software and the gradual development of the C language and the operating system environments available, some adaptation is to be expected. Fortunately, the latest distribution of SR (version 2.3.2) has already been portend to two relatively modern Unix-like environments (Solaris 2.2 and Linux), so the necessary changes turned out to be confined to a one area:

`gcc 3`, the system compiler of NetBSD-2.0, doesn't provide old `<varags.h>` variable argument functions anymore, so those had to be converted to `<stdarg.h>` syntax. Also, none of those functions had fixed arguments. Most of the functions had a first logical parameter `char *locn` which could be changed into a fixed parameter. A few functions had a first integer parameter (a count of the remaining parameters). In one

Implementation	Context switch times
i386 assembler	0.059 μs
SVR4 system calls	6.025 μs

Table 1: *Raw context switch times*

case (`sr_cat`), the loops extracting the parameters from the variable argument list had to be changed to be initialized with the newly introduced fixed parameter.

3 Verification methods

SR itself provides a basic and an extended verification suite for the whole system; also a small basic test for the context switching primitives.

The basic suite should be run to test an installation; the context switch tests and the extended suite are used to verify a new porting effort.[4]

3.1 Context Switch Primitives

The context switch primitives can be independently tested by running `make` in the subdirectory `csw/` of the distribution; this builds and runs the `cstest` program, which implements a small multithreaded program and checks for detection of stack overflows, stack underflows, correct context switching etc.

3.2 Overall System

When the context switch primitives seem to work individually, they — and the building system used to build SR, and the `sr` compiler, linker, etc. need also to be tested.

A basic verification suite is in the `vsuite/` subdirectory of the distribution; it can be extended with more tests from a seperate source archive `vs.tar.Z`. It is run by calling the driver script `srv/srv`, which provides normal and verbose modes, as well as using the installed vs. the freshly compiled SR system. The only test that is expected to fail is the `srgrind` source code formatter — it needs the `vgrind` program as a backend.

4 Performance evaluation

SR comes with two performance ealuation packages. The first, for the context switching primitives, is in the `csw/` subdirectory of the source distribution; after `make csloop` you can start `./csloop N` where N is the number of seconds the test will run approximately.

Tests of the language primitives used for multithreading are in the `vsuite/timings/` subdirectory of the source tree enhanced with the verification suite. They are run by three shell scripts to compile them, run them, and summarize the results in a table.

5 Establishing a baseline

There are two extremes possible when implementing the context switch primitives needed for SR: implementing each CPU manually in assembler code (what the SR project does normally) and using the SVR4-style `getcontext()` and `setcontext()` functions which operate on `struct ucontext`; these are provided as experimental code in the file `csw/svr4.c` of the SR distribution.

Test description	i386 ASM	SVR4 s.c.
loop control overhead	0.01 μs	0.01 μs
local call, optimised	0.07 μs	0.07 μs
interresource call, no new process	1.45 μs	1.39 μs
interresource call, new process	2.95 μs	22.20 μs
process create/destroy	2.46 μs	26.14 μs
semaphore P only	0.07 μs	0.07 μs
semaphore V only	0.05 μs	0.05 μs
semaphore pair	0.11 μs	0.11 μs
semaphore requiring context switch	0.39 μs	9.09 μs
asynchronous send/receive	1.71 μs	1.63 μs
message passing requiring context switch	1.90 μs	14.50 μs
rendezvous	2.65 μs	27.05 μs

Table 2: *Run time system performance. The median times reported by the SR script* `vsuite/timings/report.sh` *are reported.*

The first tests were done by using the provided i386 assembler context switch routines. After verifying correctness and noting the times (see tables 5 and 5), the same was done using the SVR4 module instead of the assembler module.

All tests were done on a 500 MHz Pentium III machine with 16+16 kB of primary cache and 512 kB of secondary cache, and 128 MB of main memory, running NetBSD-2.0_BETA as of end of June 2004.

The table shows a factor-of-about-ten performance hit for the operations that require context switches; note, however, that the absolute values for all such operations are still smaller than $30\,\mu s$ on 500 MHz machine and will likely not be noticable if a parallelized program is run on a LAN-coupled cluster: on the switched LAN connected to the test machine, the time for an ICMP echo request to return is about 250 μs.

6 Possible improvements using NetBSD library calls

While using the system calls `getcontext` and `setcontext`, as the `svr4` module does, should not unduly penalize an application distributed across a LAN, it might be noticable with local applications.

However, we should be able to do better than the `svr4` module without writing our own assembler modules, as NetBSD 2.0 (and up) contains its own set of them for the benefit of its native Posix threads library (`libpthread`), which does lots of context switches within a kernel provided light weight process ([5]). The primitives provided to `libpthread` by its machine dependent part are the two functions `_getcontext_u` and `_setcontext_u` with similar signatures to `getcontext` and `setcontext`.

There are a few difficulties that arise while pursuing this.

First, on one architecture (i386) `_setcontext_u` and `_getcontext_u` are implemented by calling through a function pointer which is initialized depending on the FPU / CPU extension mode available on the particular CPU used (on i386, 8087-mode vs. XMM). from this. On this architecture, `_setcontext_u` and `_getcontext_u` are defined as macros in a private header file not installed. The developer in charge of the code has indicated that he might implement public wrappers; until then, we'd have to check all available NetBSD architectures and copy the relevant files.

Second, there is no context initializing function at the same level as `_setcontext_u` and `_getcontext_u`. `makecontext` looks like it would be good enough but this has to be analyzed further.

7 Work items left to do

7.1 Building a package for `pkgsrc`

To ease installation, a prototype package for the NetBSD package system has been built. It needs a bit of refinement, though, but will be available soon. (As the NetBSD package system is available for more operating systems than NetBSD, a bit more work is needed.)

7.2 Implementing and testing multithreaded SR

SR can be compiled in a mode where it will make use of multiple threads provided by the underlying OS, so that it can use more than one CPU of a single machine. This has not been implemented yet for NetBSD, but should be.

References

[1] Gregory R. Andrews and Ronald A. Olsson, *The SR Programming Language: Concurrency in Practice* (Benjamin/Cummings, 1993

[2] Gregory R. Andrews, Ronald A. Olsson, Michael H. Coffin, Irving Elshoff, Kelvin D. Nilsen, Titus Purdin and Gregg M. Townsend, *An Overview of the SR Language and Implementation*, 1988, ACM TOPLAS Vol. 10.1, p. 51-86

[3] Gregory R. Andrews, *Foundations of Multithreaded, Parallel, and Distributed Programming,* Addison-Wesley, 2000 (ISBN 0-201-35752-6)

[4] Gregg Townsend, Dave Bakken, *Porting the SR Programming Language*, 1994, Department of Computer Science, The University of Arizona

[5] Nathan J. Williams, *An Implementation of Scheduler Activations on the NetBSD Operating System*, in: Proceedings of the FREENIX Track, 2002 Usenix Annual Technical Conference, Monterey, CA, USA, http://www.usenix.org/events/-usenix02/tech/freenix/williams.html