Installing the SR Programming Language Version 2.3

Gregg Townsend

Department of Computer Science
The University of Arizona

October 7, 1994

Minor update August 12, 1999 for SR 2.3.2

General Notes

The SR Programming Language runs under several different Unix systems including those of Sun, Digital, Hewlett-Packard, Silicon Graphics, and IBM. A list is included in the **Systems** file in the main directory.

The standard distribution package requires about five megabytes of disk space. This expands to around ten megabytes during the build process. The final, installed files occupy about three megabytes.

The SR system is in the public domain and you may use and distribute it as you wish. We ask that you retain credits referencing the University of Arizona and that you identify any changes you make.

We can't provide a warranty with SR; it's up to you to determine its suitability and reliability for your needs. We do intend to continue to develop and maintain SR as resources permit and we would like to hear of any problems.

Unpacking the *tar* file

If you haven't done so already, create a new directory for building the SR system. This directory can be located anywhere and can be removed completely after the system has been built and installed. The new directory is called the ''main directory'' in these instructions and is assumed to be your current directory for all commands illustrated.

Make the main directory your current directory and unpack the tar file. This creates several files and subdirectories. The procedure for unpacking the tar file depends on its form.

If you have a tar file on tape, or already online in a disk file, enter

tar xf *file*

where *file* is the tape device or disk file name of the SR distribution.

If you have a pair of diskettes, unpack only Disk 1. Ensure that it is write protected (the tiny windows in the corners should both be open) and insert it in the diskette drive. Enter

dd if=*dev* **bs=36b | uncompress | tar xf –**

where *dev* is the raw diskette device. On a Sun, for example, this is **/dev/rfd0c;** or, if the volume daemon is running, use **/vol/dev/aliases/floppy0** after first running **volcheck**. Some systems may report ''Broken pipe'' or ''dd: write error'' at the end of the unpacking step; don't worry about this. It occurs because **dd**

The SR Programming Language comes from the Department of Computer Science, The University of Arizona, Tucson, Arizona 85721 USA. The implementation is available by anonymous FTP from ftp.cs.arizona.edu, and the SR Project may be reached by sending electronic mail to sr-project@cs.arizona.edu.

doesn’t stop at the end of the tar file.

System Components

The SR system consists of several components. **sr** and **srl** are the key programs used directly by an SR programmer. The SR runtime system is loaded implicitly by **srl**. Supplemental tools, example programs, and documentation are also provided.

sr is the compiler proper. It translates an SR program into a C program and then calls the C compiler to produce object code.

srl is the SR linker. It generates tiny C files of configuration information and invokes the C compiler to combine these with object files and the runtime library. The end product is an executable program.

srx is an executive program that supervises ‘‘distributed’’ SR programs: those that use **create vm()**. It is not called explicitly but is forked automatically by the runtime system.

srm assists in the construction of complex programs by inspecting SR source code and automatically creating a correct *make* description file.

srprof reads a trace file produced during the execution of an SR program and produces a report of event counts by line number or an annotated listing showing the counts.

srtex formats an SR program for typesetting by TeX; **srlatex** formats a program for LaTeX. **srgrind** does a similar job for use with *troff*. **srgrind** requires the *vgrind* program in order to function.

ccr2sr, **m2sr**, and **csp2sr** are preprocessors that convert CCR notation, monitor notation, and CSP notation (respectively) into equivalent SR programs.

The **library** contains resources, globals, and externals that can be linked with SR programs. It includes an option processor and interfaces to X Windows and to the XTANGO animation package.

srv runs verification tests to check that the SR system is functioning correctly. **srvi** installs new verification tests. These programs are part of the installation process and are not used by SR programmers.

Directory Structure

Important files in the main directory include:

README	General release information and last minute notes. READ THIS FILE!
Systems	Detailed system-specific information. READ THIS, TOO.
Configuration	File paths and other miscellaneous configuration data.
Makefile	The master file of *make* directives for building SR.
sr-mode.el	Lisp code implementing an SR editing mode for GNU Emacs.

There are also several source code files and other files used in the build process.

Subdirectories of the main directory are:

sr	Source code for **sr**, the SR compiler.
srl	Source code for **srl**, the SR linker.
rts	Source code for the SR runtime system, including **srx**.
library	Source code for the SR library routines.
csw	Platform-specific runtime code for context switching.
multi	Platform-specific runtime code for multiprocessing support.
srm	Source code for **srm**, the Makefile builder.
srgrind	Source code for **srgrind**, the troff formatter.
srtex	Source code for **srtex**, the TeX formatter.
srlatex	Source code for **srlatex**, the LaTeX formatter.
preproc	Source code for the SR preprocessors **ccr2sr**, **m2sr**, and **csp2sr**.
srv	Source code for **srv** and **srvi**, the verification tools.
links	A collection of symbolic links pointing to locations where the executables are built. This directory can be put in a search path to assist in testing.

vsuite	A suite of verification programs used by **srv**.
examples	Examples of SR programs, including programs from the SR book and programs that utilize library routines. Also included are examples of CCR, monitor, and CSP programs for use with the preprocessors. **examples** is actually a symbolic link to **vsuite/examples**.
ps	Preformatted documentation, in PostScript format. See the **README** file in that directory for details.
man	Individual *man* pages for the various programs in the SR system.
doc	Source for the rest of the documentation, in *troff* form.
notes	Some miscellaneous text files, not necessarily accurate or current, containing additional, informal documentation that may or may not be useful.

Configuring the SR System

Before you build SR, you must decide where to install it. Pathnames are embedded in the binaries so that (for example) *sr* can call *srl* and *srl* can find the runtime library. Five directories must be specified: one for the commands, one for hidden files such as the runtime library, and three for the *man* pages. All of the installed files have names beginning with **sr** or containing the string **2sr**.

If you plan to install SR as a local utility you might choose distinct directories such as **/usr/local/bin, /usr/local/lib/sr,** and then **/usr/man/manl** for all the *man* pages. Alternatively, everything can be collected in a single directory such as **/usr/sr/bin** or **/home/yourname/bin.** Do not use any *existing* directories within the SR distribution, but you can safely create a new **bin** subdirectory under the main directory.

If the directories you have chosen do not now exist, you must create them manually. The installation process does not create new directories.

To configure the system, edit the file **Configuration** in the main directory. Change the SRSRC definition to reflect the path of the main directory. Define installation directories as described above for SRCMD, SRLIB, MAN1, MAN3, and MAN5. All directories must be absolute paths (beginning with ‘/’), and no comments may appear on the definition lines.

If the X window system is installed, set XINCL to the parent directory of the X11 include directory. Usually, that is **/usr/include**. On a Sun running OpenWindows, set XINCL to **/usr/openwin/include**. If X is not installed, set XINCL to be empty.

If you wish to use the animation interface provided in the library, you must first obtain and install the XTANGO package from Georgia Tech. At this writing it is available by anonymous FTP from the /pub/people/stasko directory on **ftp.cc.gatech.edu**. Then define XTANGO as the name of the XTANGO **include** directory.

If the *vgrind* program is available on your system, set VFPATH and VGMACS to the absolute paths of its back end program and macro package. Typical paths for Berkeley-derived systems are /usr/lib/vfontedpr and /usr/lib/tmac/tmac.vgrind respectively. If *vgrind* is not available, set these definitions empty. Note: Without *vgrind, srgrind* does not function.

A few other values can also be configured in the main **Makefile**. MANEXT defines the file extension for installed *man* pages. The *xx*PATH definitions define the location of the C compiler and other utilities called from within SR commands, but do *not* affect the building of SR. The CFLAGS definition sets compilation options for use while building SR.

Some operating systems require additional changes to **Configuration** and/or **Makefile**. See the section in the **Systems** file describing your particular operating system.

Configuring for MultiSR

A multithreaded version of SR, called MultiSR, is available for Intel Linux, Sun Solaris, Silicon Graphics Irix, and Sequent Symmetry systems. MultiSR provides true concurrency on a shared-memory multiprocessor without requiring any programming changes. *Note:* MultiSR is *not* used on the Intel Paragon; multiprocessing there utilizes SR’s virtual machine facilities.

To configure MultiSR, edit the main Makefile and define

MULTI=linux−x86	for Linux on Intel x86
MULTI=solaris	for Solaris
MULTI=irix	for Irix
MULTI=dynix	for Sequent

A few system-dependent configuration changes are also needed as noted below.

Intel Linux:	(no other changes needed)
Sun Solaris:	Define LIBR=−lthread in the **Configuration** file.
SGI Irix:	Define LIBR=−lmpc in the **Configuration** file.
Sequent Symmetry:	Define LIBR=−lpps in the **Configuration** file; define CFLAGS=−Y in the main **Makefile**. Ignore warning messages about ‘‘Parallel library not detected’’ that occur during the build.

Building and Testing

To build the SR system, simply type

make

in the main directory. This builds all the components of the SR system within the SR directory structure, altering nothing outside the structure. The build process takes two to ten minutes on a typical modern workstation.

If you later need to change the configuration information, do so and again type **make**. This sort of rebuild goes relatively quickly because only a few files need to be recompiled.

After building the system, check that it is functional by entering

srv/srv −v quick

to run the ‘‘quick’’ set of SR tests. This prints some environmental information and then runs a small set of tests. **−v** causes each verification directive to be echoed. The ‘‘quick’’ tests may actually take a few minutes, even on a fast machine; quick/jumble, in particular, has grown to over 500 lines of SR and is compiled and executed twice.

No error messages are expected. If ‘‘expected 0, got 1 from $RSHPATH’’ appears for the **quick/vm** test, it indicates a problem running the *rsh* (or *remsh*) program. This is discussed below under **Configuring Virtual Machines**.

It is not necessary to complete the installation to manually test the newly built system. If you put the **links** subdirectory in your search path, you can compile SR programs by running **sr** and **srl** with the **−e** option. This causes them to load libraries and other files from within the source directory instead of from the ultimate, installed locations.

Installing the System

After the system has been built and verified, it must be installed in its ultimate destination as configured above. Type

make install

to copy the commands, *man* pages, and support files.

To verify a correct installation, type

srv/srv −v −p quick

to run the same tests as before, but using the installed files.

The install script places **sr_mode.el**, an SR editing mode for GNU Emacs, and **srlatex.sty**, a LaTeX style file for use with *srlatex*, in the directory configured as SRLIB. Emacs and LaTeX will not look for them there, however. If you use those programs, we recommend that you put symbolic links in the directories that they search. The symbolic links should point to the files in SRLIB.

Configuring Virtual Machines

At this point, a complete SR system has been built and installed. Without further reconfiguration, however, remote virtual machines may not work properly. This reconfiguration was deferred until now in order to have a working SR system as a testbed. If you don't have a network of machines or don't need to use **create vm() on** *n*, you can skip this section. If you're building SR on an Intel Paragon, you should also skip this section.

When a program creates a virtual machine and specifies a host machine on which to place it, SR uses a remote shell to run the program on the remote host. The remote shell program is *rsh*(1) on most systems or *remsh*(1) on some System V derived Unixes. Set the RSHPATH definition in the Configuration file to name the correct program. Under Solaris 2, use **/bin/rsh**.

Verify that *rsh* or *remsh* is working by entering

```
rsh `hostname` date
```

If necessary, substitute **remsh** for **rsh** and/or **uname −n** for **hostname.** If this doesn't print the date, there is a configuration problem outside of SR; seek local assistance.

The name of the executable file passed to *rsh* is the same for all remote hosts and is controlled by an **srmap** file. This contains patterns for matching the program's filename and corresponding templates for generating the remote filename for *rsh.* The format of **srmap** is described in its *man* page, **man/srmap.5**.

If your network provides transparent access to remote disks, it should be possible to make remote execution work automatically by specifying templates that generate host-independent filenames. The **srmap.az** file in the main directory is an example of how this is done at Arizona.

Without remote disk access, users of multiple virtual machines will need to manually copy their programs to the remote hosts (*e.g.*, using *rcp* or *rdist*) before beginning execution. The configuration in **srmap** controls where the programs must be placed, so a simple and straightforward method is desirable. One way to do this is to generate a path relative to the user's home directory. For example, if **srmap** contains the line

```
sequoia:/usr?/*/**     ~$2/$3
```

then when a program **/usr3/username/path** is run on host **sequoia**, SR expects to find copies of the executable in **~username/path** on remote machines.

Edit **srmap** in the main directory and set up a configuration appropriate to your local situation. Refer to the **srmap** *man* page for a detailed description of the format, and use the Arizona configuration as a starting point and an example.

Make sure that your search path includes the directory where you just installed **sr**, and type **rehash** if your shell requires it. Go into the **examples/remote** subdirectory and type

```
sr remote.sr
```

to compile a test program. Set the environment variable SRMAP to the *absolute* pathname of the new **srmap** file.

Begin with a simple test by typing

```
a.out
```

to run the program with no arguments. It should simply tell you the local hostname; this verifies that **srx** is accessible, and that the location of **a.out** matches one of the patterns in **srmap**. Then try giving the local hostname as an explicit argument; this verifies that the generated filename works on the present host.

Now add other hostnames as command arguments; if remote execution fails (perhaps as expected), then the diagnostics from *rsh* give the file name attempted. Copy **a.out** into other directories, and onto other hosts if necessary. Run it from various locations with various hostname arguments. Run it using absolute and relative paths. Try to test any special cases used in the **srmap** file.

When you are satisfied with the configuration in **srmap**, return to the main directory and type

```
make install
```

to reinstall the system including the revised **srmap** file.

Cleaning Up

After the system has been installed, nothing within the SR directory structure is needed to build or run SR programs. The structure can be backed up on tape and removed from the disk.

If you wish to keep the source code online, type

```
make clean
```

to remove executables and intermediate files from the build process. **Warning:** the cleanup process is a bit aggressive, and it removes all files within the directory structure that satisfy certain tests. If you have created files of your own within the structure, and you wish to preserve them, it would be prudent to first copy them elsewhere.

The **doc** and **ps** directories can be deleted manually if you do not wish to retain them.

The Full Verification Suite

The standard distribution of SR includes a few confidence tests and sample programs in the main *tar* file. These are the tests run by **srv/srv quick**, and should be sufficient to verify correct installation of an unmodified SR system.

A more extensive set of tests is available for those who wish to modify the system or transport it to a different system architecture. There are about 450 tests in 2400 files, requiring about three megabytes of disk space. The set includes a **timings** subdirectory of performance tests.

The full test set is provided as a second diskette, a second *tar* file on tape, or as a separate file for *ftp* distributions. This *tar* file, when unpacked in the main directory, creates several new subdirectories of **vsuite**. If **srv/srv** is run with no parameters it executes all of the tests. A full run typically takes between one and four hours.

Porting to Other Systems

It is possible to port SR to other system architectures besides those presently supported; some assembly language programming is required. 32-bit Unix systems with conventional memory models are most easily accommodated. Instructions for porting SR are contained in **doc/port.ms**.

Feedback

Please let us know of any problems you encounter so that we can continue to improve SR. We can be reached by electronic mail at:

sr-project@cs.arizona.edu

Our FAX machine is at:

+1 520 621 4246

Our mailing address is:

SR Project
Department of Computer Science
Gould-Simpson Building
University of Arizona
Tucson, Arizona 85721

We'll need to know what computer and operating system you are using, what version of SR, and your name, address, and telephone number.

Because of limited resources we can't promise to fix every problem, but we appreciate all comments and acknowledge all mail.

References

Gregory R. Andrews and Ronald A. Olsson, *The SR Programming Language: Concurrency in Practice* Benjamin/Cummings, 1993, ISBN 0-8053-0088-0.

Gregory R. Andrews, Ronald A. Olsson, et al., *An Overview of the SR Language and Implementation.* ACM Trans. on Prog. Lang. and Systems 10, 1 (January, 1988), 51-86.