Subject: Memory Mapped Files (update) (long)
Posted by korpela on Thu, 16 Jul 1998 07:00:00 GMT
View Forum Message <> Reply to Message

An update to the VARRAY software, which provides better memory mapped file support to IDL, is available at http://sag-www.ssl.berkeley.edu/~korpela/mmap/

VARRAY allows IDL under UNIX systems to access files larger than available physical or standard virtual memory as standard IDL variables. As an added benefit it provides shared memory and a simple interprocess communication method for IDL programs.

This release (1.03) fixes one bug, and adds support for compilation under Solaris. (Thanks to Angelos Vourlidas at the Naval Research Lab for providing both of these enhancements.)

--
Eric

A description of VARRAY usage follows.

MEMORY MAPPED FILES FOR IDL

Introduction

Shortly after beginning to use IDL, I became annoyed with a couple features of IDL. First, when working with many large images, I would often run out of virtual memory, despite having 127 MB available. Second, the ASSOC feature, which associates a file with an IDL array, does not work as I had hoped. Rather than easily allowing access to any element of any array contained in a file, it requires that elements be copied into temporary arrays, and then written back to the file array. Eventually I got tired of it both of these problems and decided to do something about it. I sat down and wrote _VARRAY_. It has the advantages of solving both problems, and a couple I never thought of.

VARRAY: Specifications

VARRAY is in external system routine in IDL. It is written in C and has been tested on IDL 4.0.1 and 5.0 under SunOS 4.1.3. It should, however, work with minor modifications under most UNIX systems that support the _mmap()_ and _ftruncate()_ functions. Making it work under Win32 is more of an effort, but should be possible. I have no idea whether there is a _mmap()_ equivalent under VMS.

Compiling VARRAY

VARRAY is shipped with a Makefile that may require some modifications depending upon the system used. Macros are provided for SunOS systems with GCC and ACC compilers (GCC is recommended), Solaris, and Linux systems. (The code has never been compiled or tested on a Linux system. Modifications are likely to be required.) Once the Makefile has been modified, _ VARRAY_ is compiled with the command:

make varray.so

VARRAY: Usage

The shared object file "varray.so" needs to be linked into the running IDL process using the LINKIMAGE routine. I place the following command in my IDL_STARTUP routine:

LINKIMAGE,'VARRAY','~/idl/mmap/varray.so',1,'varray',min_arg s=1,max_args=10,/keywords

Once the shared object is loaded, the _VARRAY_ function becomes available. The syntax of the _VARRAY_ function is:

array=VARRAY([filename],element,[dim1,dim2,dim3,...dim8],[/w ritable],[/status])

where

* "filename" is the name of the file you wish to associate with the array. If the filename is omitted, a writable temporary file with a unique name is created in the /tmp directory.

* "element" is a variable type of each element of the array. Currently only numeric scalar types are allowed.

* "dim1..dim8" are the dimensions of the array. If the file is writable and the dimensions are larger than the current file size, the file is _ftruncate()_d to the appropriate size.

* The "/writable" keyword specified that the file is writable and that the changes to array elements are shared and written to disk.

* The "/status" keyword causes _VARRAY_ to print the number of open files to the standard output. Currently _VARRAY_ supports 32 simultaneously open files.

For example, the command

```
A=VARRAY("test.dat",byte(0),512,512,/writable)
```

opens a file named "test.dat", creating it if it doesn't exist, and assigns it to a 512 by 512 byte array. The elements of this array can be accessed and written to. For example:

```
A(*,*)=byte(255*randomu(seed,512,512))
```

will write random values into the file. When the variable is deleted (i.e DELVAR,A) or reallocated (i.e. A=SOMETHING) all changes will be updated on disk.

Sparse Files

VARRAY supports sparse files. In a sparse file, only those portions of the file that contain non-zero data are written to disk. Try the following in IDL:

```
a=fltarr(8192,8192)
```

Chances are, you just saw the message (unless you had 256 MB free):

```
% Unable to allocate memory: to make array.
  Not enough memory
% Execution halted at:  $MAIN$
```

Now, with _VARRAY_ loaded try the following:

```
a=varray("test.dat",float(0),8192,8192,/writable)
help,a
```

You should see...

```
A               FLOAT     = Array(8192, 8192)
```

Going to UNIX and doing "ls -l", we see that the file is 268435456 bytes long and takes up 24k of disk space. Now convince yourself that the array is real by doing

```
a(4096,4096)=!pi
print,a(4096,4096)
```

You'd better see 3.14159. Checking the file size again you'll see that it's still 268435456 bytes long, but now it takes up 40k of disk space. Check that things are repeatable by deleting the variable, and reloading it.

```
delvar,a
```

```
a=varray("test.dat",float(0),8192,8192,/writable)
print,a(4096,4096)
```

You should still see 3.14159. If you want to sit around for a long time you can even "print,total(a)".

Shared Memory

A happy circumstance of _VARRAY_ is that it allows memory to be shared between IDL processes. If you map a file with the /writable keyword, the changes will be shared with any other process that maps the file. As an example, start two idl processes and link "varray.so" to them. In each, enter:

```
a=varray("shm.dat",fix(0),100,/writable)
```

Now, in one enter "a(0)=1" then in the other, enter "print,a(0)." Presto, interprocess communication. Of course there's no protection for simultaneous access, so for each variable I would recommend that one process read and the other write.

Bugs and Stuff-To-Do

There's always another bug or feature. Here are a few you should note:

* Arrays that are not mapped as writable use up swap space, as the system ensures that enough swap space is available to support changes to the array values. Thus, in order to save swap space, files must be mapped "/writable."

* In the above shared memory example, if the array in the reader process is not mapped "/writable" and is written to, the array looses it's mapping to the file, and the interprocess connection disappears.

* A planned "OFFSET" keyword, which specifies an offset in the file for the mapping to begin, has not yet been implemented. Therefore, only headerless files are supported for now.

Release History

* Version 1.00 First release
* Version 1.01 Fixed several function definitions
* Version 1.02 Background syncronization
* Version 1.03 Bug Fix, removed third parameter from the _munmap()_ call. Added Solaris section to Makefile.

Acknowledgements

My thanks to Angelos Vourlidas at the Naval Research Lab, for

providing the Solaris binary and for pointing out the bug in the
munmap() call.

--
Eric Korpela | An object at rest can never be
korpela@ssl.berkeley.edu | stopped.
Click for home page.
