MIT-SHM(The MIT Shared Memory Extension)

How the shared memory extension works

Jonathan Corbet, National Center for Atmospheric Research `<corbet@ncar.ucar.edu>`
Edited by Keith Packard

MIT-SHM(The MIT Shared Memory Extension): How the shared memory extension works

by Jonathan Corbet and Keith Packard

X Version 11, Release 7.7

Version 1.0

Copyright © 1991 X Consortium

Abstract

This document briefly describes how to use the MIT-SHM shared memory extension. I have tried to make it accurate, but it would not surprise me if some errors remained. If you find anything wrong, do let me know and I will incorporate the corrections. Meanwhile, please take this document "as is" (eman improvement over what was there before, but certainly not the definitive word.)

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE X CONSORTIUM BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Except as contained in this notice, the name of the X Consortium shall not be used in advertising or otherwise to promote the sale, use or other dealings in this Software without prior written authorization from the X Consortium.

X Window System is a trademark of The OpenGroup.

Table of Contents

Chapter 1. REQUIREMENTS

The shared memory extension is provided only by some X servers. To find out if your server supports the extension, use xdpyinfo(1). In particular, to be able to use this extension, your system must provide the SYSV shared memory primitives. There is not an mmap-based version of this extension. To use shared memory on Sun systems, you must have built your kernel with SYSV shared memory enabled -- which is not the default configuration. Additionally, the shared memeory maximum size will need to be increased on both Sun and Digital systems; the defaults are far too small for any useful work.

Chapter 2. WHAT IS PROVIDED

The basic capability provided is that of shared memory XImages. This is essentially a version of the ximage interface where the actual image data is stored in a shared memory segment, and thus need not be moved through the Xlib interprocess communication channel. For large images, use of this facility can result in some real performance increases.

Additionally, some implementations provided shared memory pixmaps. These are 2 dimensional arrays of pixels in a format specified by the X server, where the image data is stored in the shared memory segment. Through use of shared memory pixmaps, it is possible to change the contents of these pixmaps without using any Xlib routines at all. Shared memory pixmaps can only be supported when the X server can use regular virtual memory for pixmap data; if the pixmaps are stored in some magic graphics hardware, your application will not be able to share them with the server. Xdpyinfo(1) doesn't print this particular nugget of information.

Chapter 3. HOW TO USE THE SHARED MEMORY EXTENSION

Code which uses the shared memory extension must include a number of header files:

```
#include <X11/Xlib.h>          /* of course */
#include <sys/ipc.h>
#include <sys/shm.h>
#include <X11/extensions/XShm.h>
```

Of course, if the system you are building on does not support shared memory, the file XShm.h may not be present. You may want to make liberal use of #ifdefs.

Any code which uses the shared memory extension should first check to see that the server provides the extension. You could always be running over the net, or in some other environment where the extension will not work. To perform this check, call either

`Status` **`XShmQueryExtension`**`(`*`*display`*`);`

or

`Status` **`XShmQueryVersion`**`(`*`*display`*`, `*`*major`*`, `*`*minor`*`, `*`*pixmaps`*`);`

Where "display" is, of course, the display on which you are running. If the shared memory extension may be used, the return value from either function will be True; otherwise your program should operate using conventional Xlib calls. When the extension is available, \fCXShmQueryVersion\fP also returns "major" and "minor" which are the version numbers of the extension implementation, and "pixmaps" which is True iff shared memory pixmaps are supported.

Chapter 4. USE OF SHARED MEMORY XIMAGES

The basic sequence of operations for shared memory XImages is as follows:

1. Create the shared memory XImage structure

2. Create a shared memory segment to store the image data

3. Inform the server about the shared memory segment

4. Use the shared memory XImage, much like a normal one.

To create a shared memory XImage, use:

`XImage` **`*XShmCreateImage`**`(`*`*display`*`,` *`*visual`*`,` *`depth`*`,` *`format`*`,` *`*data`*`,` *`*shminfo`*`,` *`width`*`,` *`height`*`);`

Most of the arguments are the same as for XCreateImage; I will not go through them here. Note, however, that there are no "offset", "bitmap_pad", or "bytes_per_line" arguments. These quantities will be defined by the server itself, and your code needs to abide by them. Unless you have already allocated the shared memory segment (see below), you should pass in NULL for the "data" pointer.

There is one additional argument: "shminfo", which is a pointer to a structure of type XShmSegmentInfo. You must allocate one of these structures such that it will have a lifetime at least as long as that of the shared memory XImage. There is no need to initialize this structure before the call to XShmCreateImage.

The return value, if all goes well, will be an XImage structure, which you can use for the subsequent steps.

The next step is to create the shared memory segment. This is best done after the creation of the XImage, since you need to make use of the information in that XImage to know how much memory to allocate. To create the segment, you need a call like:

```
shminfo.shmid = shmget (IPC_PRIVATE,
          image->bytes_per_line * image->height, IPC_CREAT|0777);
```

(assuming that you have called your shared memory XImage "image"). You should, of course, follow the Rules and do error checking on all of these system calls. Also, be sure to use the bytes_per_line field, not the width you used to create the XImage as they may well be different.

Note that the shared memory ID returned by the system is stored in the shminfo structure. The server will need that ID to attach itself to the segment.

Also note that, on many systems for security reasons, the X server will only accept to attach to the shared memory segment if it's readable and writeable by "other". On systems where the X server is able to determine the uid of the X client over a local transport, the shared memory segment can be readable and writeable only by the uid of the client.

Next, attach this shared memory segment to your process:

shminfo.shmaddr = image->data = shmat (shminfo.shmid, 0, 0);

The address returned by shmat should be stored in *both* the XImage structure and the shminfo structure.

To finish filling in the shminfo structure, you need to decide how you want the server to attach to the shared memory segment, and set the "readOnly" field as follows. Normally, you would code:

shminfo.readOnly = False;

If you set it to True, the server will not be able to write to this segment, and thus XShmGetImage calls will fail.

Finally, tell the server to attach to your shared memory segment with:

```
Status XShmAttach (display, shminfo);
```

If all goes well, you will get a non-zero status back, and your XImage is ready for use.

To write a shared memory XImage into an X drawable, use XShmPutImage:

`Status` **`XShmPutImage`** `(`*`*display`*`,` *`d`*`,` *`gc`*`,` *`*image`*`,` *`src_x`*`,` *`src_y`*`,` *`dest_x`*`,` *`dest_y`*`,` *`width`*`,` *`height`*`,` *`send_event`*`);`

The interface is identical to that of XPutImage, so I will spare my fingers and not repeat that documentation here. There is one additional parameter, however, called "send_event". If this parameter is passed as True, the server will generate a "completion" event when the image write is complete; thus your program can know when it is safe to begin manipulating the shared memory segment again.

The completion event has type XShmCompletionEvent, which is defined as the following:

```
typedef struct {
    int type;              /* of event */
    unsigned long serial;  /* # of last request processed */
    Bool send_event;       /* true if came from a SendEvent request */
    Display *display;      /* Display the event was read from */
    Drawable drawable;     /* drawable of request */
    int major_code;        /* ShmReqCode */
    int minor_code;        /* X_ShmPutImage */
    ShmSeg shmseg;         /* the ShmSeg used in the request */
    unsigned long offset;  /* the offset into ShmSeg used */
} XShmCompletionEvent;
```

The event type value that will be used can be determined at run time with a line of the form:

int CompletionType = XShmGetEventBase (display) + ShmCompletion;

If you modify the shared memory segment before the arrival of the completion event, the results you see on the screen may be inconsistent.

To read image data into a shared memory XImage, use the following:

Status **XShmGetImage** (**display*, *d*, **image*, *x*, *y*, *plane_mask*);

Where "display" is the display of interest, "d" is the source drawable, "image" is the destination XImage, "x" and "y" are the offsets within "d", and "plane_mask" defines which planes are to be read.

To destroy a shared memory XImage, you should first instruct the server to detach from it, then destroy the segment itself, as follows:

```
XShmDetach (display, shminfo);
XDestroyImage (image);
shmdt (shminfo.shmaddr);
shmctl (shminfo.shmid, IPC_RMID, 0);
```

Chapter 5. USE OF SHARED MEMORY PIXMAPS

Unlike X images, for which any image format is usable, the shared memory extension supports only a single format (i.e. XYPixmap or ZPixmap) for the data stored in a shared memory pixmap. This format is independent of the depth of the image (for 1-bit pixmaps it doesn't really matter what this format is) and independent of the screen. Use XShmPixmapFormat to get the format for the server:

```
int XShmPixmapFormat(*display);
```

If your application can deal with the server pixmap data format (including bits-per-pixel et al.), create a shared memory segment and "shminfo" structure in exactly the same way as is listed above for shared memory XImages. While it is, not strictly necessary to create an XImage first, doing so incurs little overhead and will give you an appropriate bytes_per_line value to use.

Once you have your shminfo structure filled in, simply call:

```
Pixmap XShmCreatePixmap(*display, d, *data, *shminfo, width, height,
depth);
```

The arguments are all the same as for XCreatePixmap, with two additions: "data" and "shminfo". The second of the two is the same old shminfo structure that has been used before. The first is the pointer to the shared memory segment, and should be the same as the shminfo.shmaddr field. I am not sure why this is a separate parameter.

If everything works, you will get back a pixmap, which you can manipulate in all of the usual ways, with the added bonus of being able to tweak its contents directly through the shared memory segment. Shared memory pixmaps are destroyed in the usual manner with XFreePixmap, though you should detach and destroy the shared memory segment itself as shown above.