Marsv for Plan 9

Kenji Okamoto [1], Yoshitatsu Suzuki [2]
Coll. Integrated Arts & Sciences,
Osaka Prefecture Univ.,
Sakai, Osaka, 599-8531, Japan

1 Introduction

After GUI was spread, we have to deal with huge sized programs written for investigation of planetary data. There are many commercial based utilities to make those smaller such that GUI libraries or more abstracted forms as language based IDL etc.. Those utilities are, unfortunately, suffered from very often updates, which makes us annoying sometime. We searched many operating systems and found Plan 9[1] from Bell Labs, which was newly designed from scrach with the aim of replacing Unix in a future, and has a very compact graphics system. Plan 9 is now open to researchers without fee. Based on this Plan 9 OS, we expected to make a smaller sized and well featured GUI program for our investigation of Mars MOLA and image data, and got such with total C source codes of about 10,300 lines. We call the program as marsv.
Marsv has no icon, and only has top menu bar where some menus are shown as their default values. Each menu item has a list of other similer menu items behind it, which is shown by mouse button3. Mouse button2 executes the item shown in the menu bar, etc, which is similer to pulldown menu of X, but not same. Marsv displays each window as a stratified and tiled window. The size of individual window can be changed by mouse at anytime and anywhere as if it has window manager like such in X.
Marsv reads many kinds of PDS formatted planetary images such as Viking, Voyager, Mars Global surveyer, Megellan etc. Mars MOLA grid data also can be read, and are laid over the corresponding Viking cube image processed by ISIS[2] from USGS, and we can make contour map on the image. We can also measure interactively the elevation, longitude and latitude of individual mouse point in the image by "Elevation" menu behind the "Contour" top menu. When MOLA data alone are read, by specifying the interesting area by mouse, those elevation data are converted to brightness, and marsv shows them as an image. Botton1 reads the elevation, longitude and latitude of the point, and button2 erases the display of those data. Annotations can be attached by "Pen" menu, and "Palette" menu changes the color of the contour map etc.. Marsv can also save the processed image to a postscript or Plan 9's PIC format file etc.

2 Implementation

Marsv is designed for concurrent parallel programming using the thread(2) library of Plan 9, which means it has full scalability. Plan 9 has two different level graphic libraries of draw(2) and control(2), where the former is the lowest level graphic library, and may correspond to X toolkit library judged from its range of functionarity, and the latter is a higher level library which deals with some widget sets such as menu, button, text input etc. We used somewhat revised version of this control(2), because we needed 3 button functions for our menu bar.
When marsv is running, we have three processes, two of which are for watching the events (channel) from keyboard and mouse, and another one contains threadmain()

[1] okamoto@granite.cias.osakafu-u.ac.jp
[2] now at Itochu Techno-Science Co., Tokyo, yoshitatsu.suzuki@ctc-g.co.jp

function of marsv program and many other threads called from the threadmain(). Those three processes are forked by use of light weight rfork(2) of Plan 9, and will not use so much memory space by sharing many of the memory. In our case marsv uses 828 Kb memory area for the main process when it starts. Those three processes uses preemptive time sharing, then, can run simultaneously, which enables rapid communication. It is the reason why those three preemptive time sharing processes are used for watching keyboard and mouse.
The last process consists many tasks, each of which is named as 'thread', and will be called and run when appropriate event message has been detected or more precisely, any appropriate channel has the message to the thread. The message will be passed through channel, which is a buffered or unbuffered queue for a fixed-size message. Only one of threads can run as an exclusive thread in the process at a moment we are concerning. It is the most simple case study of thread(2) library of Plan 9. Only the channel is the mean to communicate with each other for processes or threads.

2.1 Thread programming and marsv

Switching each thread is controled by sending or receiving a message through channel (ctl in the following example). Therefore, the "for loop" both in thread1() and thread2() will be blocked untill this thread will receive msg. In other words, those "for loops" are not making loop, but are blocked at the beginning line of msg = recvp(). Let us consider some typical sequences of processing here.

```
#include "wincomm.h"
    /*
      *   struct WinComm {
      *          ....
      *        Channel *ctl;
      *          ....
      *    };
      */
void
thread1(void *v) {
 WinComm *v1,*v2;

 v1 = (WinComm*)v;
 v1->ctl=chancreate(sizeof(char*), 0);
  for(;;) {
    msg=recvp(v1->ctl);
    switch(msg) {
      Case 1:
        threadcreate(thread2, (void*)v2,
        10*1024);
        chanprint(v2->ctl, "exec Case 3");
        chanprint(v2->ctl, "exec Case 4");
        break;
      Case 2:
        chanprint(v2->ctl, "exec Case 4");
        chanprint(v2->ctl, "exec Case 3");
        break;
     }
  }
}

void thread2(void *v) {
 WinComm *v2;

 v2 = (WinComm*)v;
 v2->ctl=chancreate(sizeof(char*), 0);
 for(;;) {
   msg = recvp(v2->ctl);
   switch(msg) {
     Case 3:
        job3();
        yield();
        break;
      Case 4:
        job4();
        break;
    }
 }
}
```

2.1.1 Example 1

Thread1 received "exec Case 1" message from its v1->ctl channel. Thread 1 starts to act from making new thread2, and then, tries to send a message of "exec Case 3" to that new thread's channel. This message wakes up the thread2, and then thread1 goes into sleep. Thread2 processes the job3(), then meets yield() command, which passes the thread control to another thread. If no other thread has been created, the thread which will be waked up is thread1, and then, next chanprint() function will be executed, which again tosses the control back to thread2, and job4() will be processed. Then, the control of thread is hold on thread2, and processing will be blocked untill the arrival of a new message to v2->ctl channel.
However, if there is another thread3 created in the same process, we don't know which of thread1 or thread3 will be waked up, when thread2 meets yield() command.

2.1.2 Example 2

Thread1 received "exec Case 2" message from its v1->ctl channel. Thread1 tries to send "exec Case 4" message to thread2. However, we don't know whether the thread2 has been created or not. If it has been done, the message will be received by the process. If not, we will meet error of "there is no v2->ctl channel" etc, and the process will meet serious problem.
If thread2 was created before, which indicates the messages must have been sequncial from "exec Case 3" to "exec Case 4", thread2 will be waked up, and job4() will be done. After that, thread2 has its thread control, which means thread2 will be blocked until the receivement of a new message to v2->ctl channel. We have to pay careful attention which thread we want to live longer. We tried to design all the threads simpler as possible as we can. However, we have to accept some nested threads because of complexity of the processing the planetary data. The deepest nest level of marsv is seven, and seen when saving the processed image to a file. The sequence of this nesting is from top level thread of threadmain to imagerthread, canvasthread, viewerthread, savethread, filebrowsethread and selectthread from top to bottom.

2.2 Windows flexible to resize

From the programmer's point of view, one of the most difficult or cumbersome is coding to make all the windows size-flexible, because Plan 9 doesn't have window manager, but only has a function named resizecontrolset() in control(2) library, the content of which must be defined by user. All the other windows also must be prepared for resizing by the user.
In marsv, we created struct Basicwin, which defines basic components of marsv windows, and complises six subwindows, which are layoutbtn, topmenubar, scrbtn, verscr, horscr and main windows. Figure 1 shows those components. The scrbtn means scroll button, layoutbtn is for layout button, and horscr for horizontal scrollbar, and verscr for vertical scrollbar.
Resizing of all the windows must be done harmoniously. In order to satisfy this requirement, all the windows in marsv have a tree structure of parent-child relationship, and are made as child windows from a parent, that is, subwindows. Any subwindow has its rectangle area inside of its parent window's one. This is shown in Figure 2. All the size of the reactangle are defined as relative values of its parent.
Two kinds of mechanisms to make subwindows in its parent were implemented,

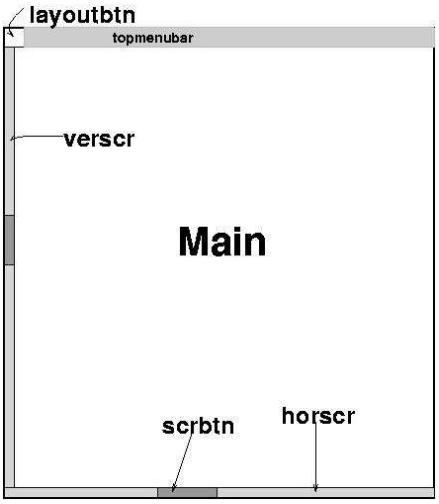


Figure 1: basic windows components

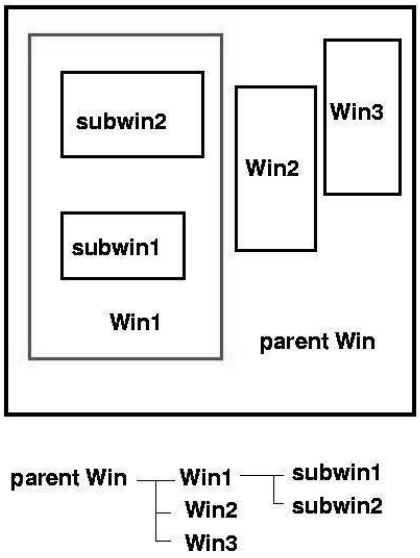


Figure 2: parent-child relations of windows

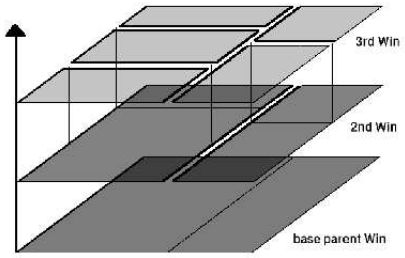
one is named fullsidesubwindow, and the other is innersubwindow.

2.2.1 Fullsidesubwindow

The name of fullside comes from the way this subwindow forms. When making fullside window in the direction of down, it uses full lateral width of its parent reactangle, and some part of vertical side expressed as percent. There are four directions, down, up, right and left as shown in Figure 3. Upper diagram of the figure shows how to make 3rd child subwindows consists of five small windows. First we make 2nd subwindow by parting the parent window vertically into two window using righr direction. The 2nd child subwindow will then be parted horizontally into 3rd child subwindows using down direction.

2.3 Connecting thread and window

After a resize flexible window is given, we have now to define the functionalities or actions of the window. The most basics of such include close and resize and draw the window. In addition, we also need a method to know when we need such action. These actions are triggered usually by mouse or keyboard actions from the computer user. This communication will be done by a message passing through channel in Plan 9 thread programming. Therefore, we need some mechanism to hold above attributes in a package to bring along with. For this purpose, we implemented struct WinComm and struct WinCommBase for marsv. WinComm has parent-child relationship tree structure as same as struct Win. It has, then, pointer members to WinComm, such as par, next, winconts. WinComm has its WinCommBase, and four communication



Making child windows by fullside window

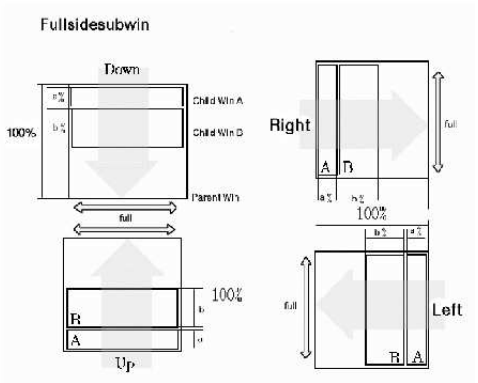


Figure 3: make fullsidesubwindow

channels, ctl, data, event, winctl. The first channel, ctl, is just a handle, and will be linked to approapriate other channel in actual use. We use ctl channel only for multiplexed receiving channel. Therefore, multiple struct WinComm can be used for sending messages, however, only one of them can receive the messages per thread without channel lockup.

A window which contains some or all of above basic components and and struct WinComm is called "Form". It is the most basic unit of marsv user interface. Above two structs have a pointer to the function of mount or unmount action, which enables a "Form" to connect to another "Form" which has no original genetic relation. It enables us to make some complicated user interface when we need it. From a geometric point of view, mount overlays a window to a part of another window. From a functional point of view, mount connects a WinComm or WinCommBase to another one, and makes parent-child relationship each other, which is maintained separately by struct WinCmChld of a member of WinComm struct. This mount or unmount function must be defined elsewhare. Figure 4 shows how the Viewer Form is mounted to the Main window of Imager Form, which is also mounted to the Main window of threadmain's subform, which is also mounted to the Main window of threadmain Form. In addition, struct WinComm has, of course, three pointers to common functions of close, resize and draw mentioned above, which must be defined elsewhare.

Most user actions will be done by pressing buttons, or pointing a interesting object by mouse, or inputting text data to a window from user's keyboard. We used the pre-defined widget sets for this purpose, which are provided by control(2) library of Plan 9. The struct WinComm and WinCommBase envelope those widget setts, and give some special meanings to them for marsv GUI program. WinCommBase has a particular Controlset(2) for this purpose.

3 Results

Figure 5 shows an example of contour map, and Figure 6 illustrates a marsv user interface. We used making contour lines algorithm by Shono et. al.[3].

4 References

[1]Pike, R, et. al.(1995) "Plan 9 from Bell Labs", 2nd ed. Plan 9 documents.

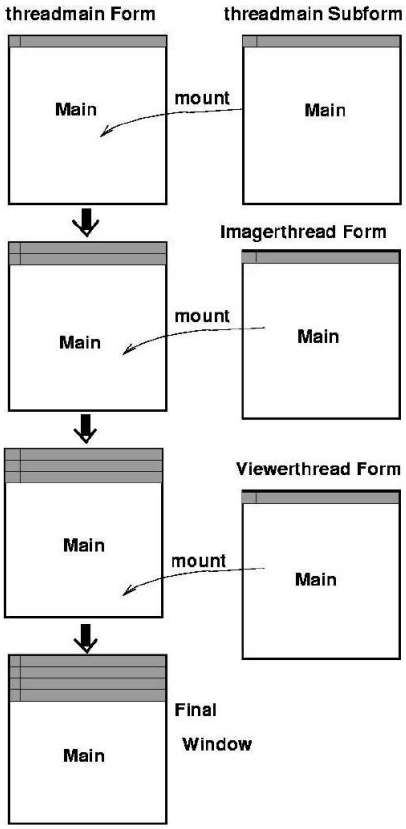


Figure 4: mount other window to Main window

[2]USGS(1996) "ISIS User's Guide"
[3]Shono, S. et. al.(1988) "Contour map by BASIC II" (in Japanese), Kyoritsu Shuppan.

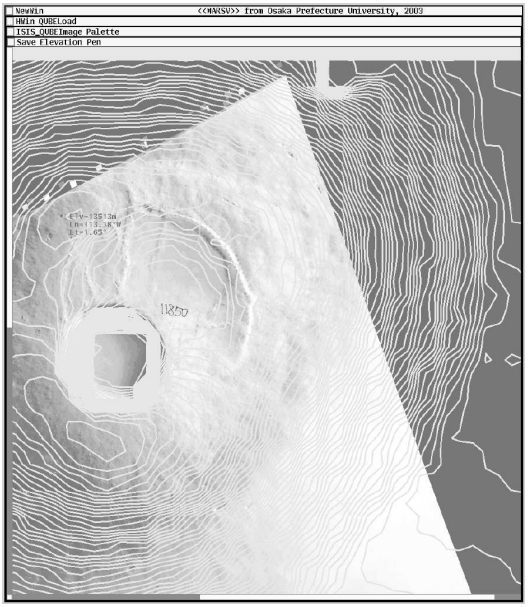


Figure 5: contour map

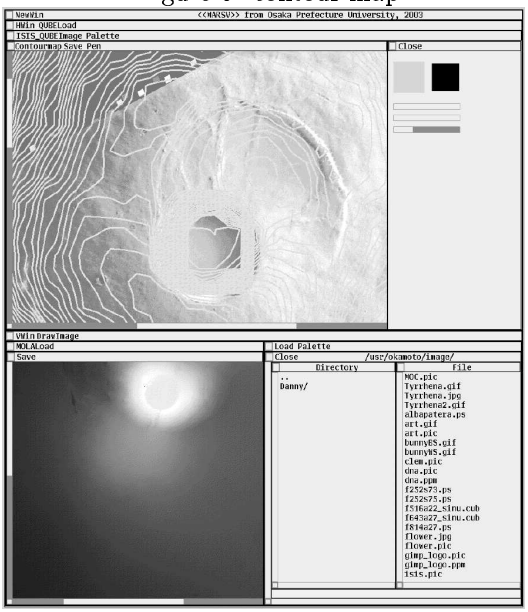


Figure 6: user interface