REVI : A Recreatable Text Editor

R. Sridharan*
Novell Software Development
(India) Pvt. Ltd.
Bangalore 560068, India
sridhar@novell.com

Jayant R. Haritsa
Dept. of Computer Science and Automation
Indian Institute of Science
Bangalore 560012, India
haritsa@csa.iisc.ernet.in

Abstract

Networks of workstations, wherein several machines share resources through a network-based file system and users can work at any console, are widely prevalent in modern computing environments. In such setups, users log out when their current work session is complete. A user who, at his next session, wishes to continue from where he left off the previous time, has to manually recreate the earlier environment, and this process is both cumbersome and error-prone.

To address the above problem, we have initiated a project called **ScreenMark**, whose goal is to provide an automated *screen–reproduction* facility. The special feature of ScreenMark is that, apart from "syntactic" screen reproduction of the layout of graphical objects, it also aims to provide "semantic" screen reproduction by recreating the *processes* that were executing in the graphical objects. We have begun the ScreenMark implementation by developing a recreatable text editor called **REVI** which creates, at login time, the exact replica of any vi editor processes that were executing at the previous logout. REVI supports location-independent computing and operates transparently with respect to the user. In this paper, we present the design and implementation of REVI.

1 Introduction

Assume that you are preparing the final draft of a paper on a computer terminal in a shared workstation environment. You have opened several windows on the screen and are executing different processes in each of these. For example, in window 1, you are editing the final draft, in window 2, you are running a simulation program and so on. You now have to leave to attend a seminar but expect to be back in an hour – when you return, you would like to immediately continue working from where you left off earlier. If you log out now, this is not possible since you have to recreate your environment from scratch when you return. An alternative option would be to lock the console (e.g., with the lockscreen [Sun86] command), but this would make the machine unusable by other users in the interim period. Moreover, it would require you to return to exactly the same machine at which you were working before.

Clearly, both the above options are not appealing. Instead, one would like to have a facility which automatically recreates the earlier environment without causing inconvenience to others. We feel that this facility would be of considerable utility in shared computing environments where each user has several *sessions* of computer usage in a day – this type of environment is especially common in educational and research laboratories. To address this problem, we have initiated a project called **ScreenMark**, whose goal is to provide an automated screen–reproducing facility (similar to a bookmark, which helps readers continue from where they left off previously).

*This work was done while the author was with the Dept. of Computer Science and Automation, Indian Institute of Science.

ScreenMark

A typical screen configuration consists of the layout of windows, icons and other graphical objects. This layout can be easily reproduced by using existing graphical system utilities. The special feature of ScreenMark is that, apart from "syntactic" reproduction of the layout of graphical objects, it also aims to provide "semantic" screen reproduction by recreating the *processes that were executing at logout* in each of the graphical objects. So, for example, if at logout time, a file was being edited in a window which occupied the top right quadrant of the screen, screen reproduction at the next login should recreate the window in the same position and have the same editor software running on the same file with the cursor at the same point in the file (as at the previous logout time). Since even the cursor is brought back to its old position, an illusion that the user had never logged out from the original machine is created, that is, it is as if "time had stood still in the interim period". An alternative viewpoint of this functionality is that users are effectively provided with "private" workstations, although they are working in a shared environment.

Another major design goal of ScreenMark is that screen reproduction should be *location-independent*. By this we mean that screen reproduction should be possible independent of which machine the user returns to in the network, as long as this machine is compatible with the machine from which the user previously logged out. This is an extremely useful feature since it gives users the freedom to choose the most convenient physical location at which to recommence their work. In this sense, ScreenMark is a logical extension to MIT's Project Athena [Arf92] where only file and resource access are made transparent to the user's location.

In order to make it viable, ScreenMark should provide the above functionalities efficiently, and this translates into a design goal of low overhead in space and time. Finally, ScreenMark should be transparent to the user in that no special actions need to be taken to recreate the screen. A discussion of why the existing features of operating systems, in particular Unix [Bac92], are not sufficient to satisfactorily meet all the design goals of ScreenMark, is available in [Sri95].

REcreatable VI (REVI)

We have begun the ScreenMark implementation by developing a recreatable text editor called **REVI** (**RE**creatable **VI**) based on the popular editor vi [Joy84]. REVI creates, at login time, screen replicas of all the vi editor processes that were executing at the previous logout. In this paper, we present the design and implementation of REVI.

REVI has been designed such that an "exact" reproduction of the previous (logout) state of the editor process is obtained. By exact we mean that users are unable to make out any difference between their previous editor state (at logout) and the current editor state. Note however that this is really an "illusion" since REVI terminates the old editor process at logout time and creates a *new* editor process at login time (as explained in Section 3). Therefore, users will be able to recognize the change by looking at the process identifiers but with regard to the editor state itself, no change can be recognized. Our notion of exactness includes such detailed information as the position of the cursor and the internal buffer contents of the editor.

REVI is "location-independent" in the sense that the user can log in to *any* machine on the network and recreate his previous logout state with respect to the text editor processes, – the only constraint is that the current machine should be compatible with the machine from which he had logged out earlier.

To achieve its goals, REVI uses a **checkpointing** (state saving) mechanism similar to that used in the Condor remote job execution system [Lit88, Bri89]. All active editor processes are checkpointed at the instant of logout. Additional information regarding the graphical configuration such as the size and coordinates of graphical objects is also stored. This information is later used to recreate the editor processes when the user logs in again.

REVI is currently operational on networks of Sun and Silicon Graphics workstations at the Supercomputer Education and Research Center in the Indian Institute of Science.

2 Design of REVI

We have begun the implementation of the ScreenMark project by developing a recreatable text editor called **REVI** (**RE**creatable **VI**), based on the popular editor vi [Joy84]. The text editor was chosen for our implementation since it is extensively used and is a familiar utility program for most users. This section outlines the design goals of REVI and discusses their feasibility. We then describe the design approach to our implementation of REVI.

2.1 Design Goals

Ideally the user would like the system to automatically reproduce his earlier editing environment exactly. As mentioned in the Introduction, full screen reproduction could be divided into two distinct components, namely, syntax reproduction and semantic reproduction. These components are described below:

- **Syntax Reproduction**: All the vi windows that were being executed at the previous logout should be recreated at their original positions with their original sizes.
- **Semantic Reproduction**: The exact state of the vi processes at logout (including cursor position and internal buffer contents) should be recreated in the corresponding windows.

The other important design goals are outlined below:

- **Location-independence**: The recreation of the vi windows should be possible independent of which machine the user returns to in the network, as long as this machine is compatible with the machine from which the user previously logged out. This is an extremely useful feature in shared computing environments since it gives users the freedom to choose the most convenient physical location at which to recommence their work.
- **Easy to Use**: The design should be user-friendly. This means that the user should not be required to execute a complicated set of commands in order to save his vi windows and to re-create them at the next login.
- **No inconvenience to other users**: The methodology of screen saving and re-creation should be such that it causes no inconvenience to future users of the machine.
- **Low Overhead**: The overhead in space and time should not be excessive. Space overhead arises from the disk space needed to store the information needed for screen re-creation. Time overhead appears in the time taken to log in and log out – these are increased due to the need for storing the re-creation information at logout and having to recreate the screen at login.
- **User-Level Implementation**: Implementation of re-creation should be done completely at the user level, without any modification of the operating system kernel. This is important since the kernel sources are not always available and modifying the kernel ties the design to a specific operating system, whereas a user-level implementation can be ported to other operating systems with relative ease.
- **No Source Modification**: Ideally, re-creation should be implemented from outside the vi process, that is, without requiring any modification of the vi source code. This would make the design applicable not only to vi but also to any other editor, and in fact, to any other program, irrespective of the semantics of the program.

2.2 Feasibility

We now discuss the feasibility of achieving the above design goals. In our implementation of REVI (described in Section 3), we have achieved all the above design goals except for the goal of no source modification. We found that implementing screen re-creation without source modification is not easily feasible, as explained below:

Screen re-creation involves re-creating the exact state of a process. The primary technique of controlling a process from outside is to use the file system interface to processes. This is called the **/proc** file system and is provided in recent releases of most flavors of UNIX. In this file system, images of all the active processes are stored as files. Special system calls are provided to access the regions of a process through this file interface. Since this file system contains the images of only the active processes, when the user logs in later, the original editor process would no longer have an image in **/proc** and hence this technique cannot be used to re-create the original editor process. Other techniques, such as using the ptrace system call, defeat the goal of transparency since they require the user to run the editor process as the child of a controlling process.

Due to the above constraint, we have been forced, in our implementation, to modify the vi source code at a few places to facilitate re-creation. One instance where this modification is needed is with respect to the input mode of editing. That is, if the user had inserted some text in the file just before logging out, then this text should be recreated when he logs in again. If source modification is not done, there is no way of knowing from outside the editor process what text had been inserted. Hence special functions have to be added to the source code of the editor program in order to save this information.

Another instance of source modification is with respect to temporary files: The temporary file used by vi is removed whenever the process completes execution or is aborted. But this temporary file needs to be saved in order for the editor commands to work properly after re-creation. This means that the vi program code has to be modified to prevent the temporary file from being removed.

In our implementation of **REVI**, we have tried to minimize the amount of source code modification.

2.3 Design Approach

In the design of REVI, screen re-creation is split into two distinct components – saving syntactic information and saving semantic information. These components are then linked together to achieve re-creation.

Saving syntactic information requires storing the layout of the window, its position, width, height etc. This could be done in a simple manner using existing utilities in a window system. For example, in the X Window System, the syntax information can be obtained by running xwininfo on the open window.

Semantic information is saved through the technique of **checkpointing**. Checkpointing is the technique of saving the state of a process in a machine-independent manner. The state of a process consists of the text, data and stack regions [Bac92]. In the technique of checkpointing, a signal is sent to the editor process that is running and a core file is obtained. This core file is dumped by the kernel and contains the image of the process as it existed when it received the signal. The data and stack regions are obtained from the core file and the text region is obtained from the original executable. These are combined to produce a checkpoint file, which is itself executable. When the checkpoint file is executed later, it starts executing from the saved state of the process. The exact manner in which the checkpoint file is created is described in Section 3.

Re-creation is done by first recreating the window based on the saved configuration information. The editor process is then re-created by executing the checkpoint file instead of the original executable. Since the checkpoint file starts executing with the saved data and stack regions, the previous state of the process (including the cursor position) is exactly re-created. Note that the process identifier and other characteristics of the new process are different from that of the original process but as far as the user is concerned, he gets back the previous editor state. The user cannot make out the difference between the previous state of the editor (at logout) and the current editor state.

In order to support re-creation of multiple windows and multiple editor processes, the link between the syntactic and semantic information needs to be stored. This is done by storing the command name and arguments of the editor process and the geometry information of the window in which the process was originally running.

3 Implementation

Having presented the design methodology of REVI in the previous section, we now move on to describing its implementation. REVI has been implemented on Sun-3 workstations running SunOS Release 3.5 and supporting the X Window System and also on Silicon Graphics Iris workstations running IRIX Release 4.0.5. The description that follows is based on the implementation on Sun workstations.

Source Modifications

As mentioned in Section 2, re-creation has been implemented with minor modifications to the editor source code. The text editor **elvis** was chosen for this purpose since it is available in source form in the public domain. Its interface is identical to that of the commercial text editor vi.

The elvis source code was modified so that the temporary file used by it is created in the user's home directory instead of the local */tmp* directory of the machine. This is done to achieve location-independence across machines.

A new option -x was added to elvis. This option is used during re-creation to perform the necessary initialization, which includes re-opening the original file which the user was editing since the old file descriptors are no longer valid. The pathname of the current working directory of the editor process is also saved during initialization. This is stored in the file */tmp/cwd_file_P*, where P is the pid of the editor process.

A function was added to store the identifier of the window in which the editor process is running. This identifier is obtained from the environment variable WINDOWID inherited by the editor process and is supplied as an argument to xwininfo to generate information about the position and size of the corresponding window. This geometry information is stored in a file called *wininfo_P* in the */tmp* directory (as mentioned before, P is the pid of the editor process).

Finally, another function was added to determine the editor mode the user is in (input mode or edit mode) at the time of logout and to store this information in a temporary file. The name of this file is */tmp/mode_info_P*. During re-creation, this information is used to set up the appropriate editor mode.

3.1 Screen Saving

We now move on to describing how screen information is saved at the time of logout: A program called SAVE is run at logout from the user's login (or) console window. This program creates a file called *re_vi* in the user's home directory and saves the syntactic and semantic information about all editor processes that are running at the time of logout. It then destroys all the windows, kills itself and logs the user out. The manner in which the SAVE program saves the syntax information and semantic information associated with each editor window is explained below.

Syntax Information

The SAVE program run at the time of logout reads the temporary file */tmp/wininfo_P* (P is the pid of the editor process) and extracts only the relevant information, namely, the position, width and height of the window. These are then combined to make up a string which is stored in the *re_vi* file in the user's home directory. The format of the string is WxH+X+Y where W denotes the width of the window, H the height, X the x–coordinate and Y the y–coordinate. During re-creation, this string is given as an argument to the terminal emulator program (e.g. xterm).

Semantic Information

Recreating a process implies getting back the previous state of the process. This means that the process state has to be saved when the user logs out. The state of a process consists of the contents of its address space and the contents of hardware registers related to the process [Bac92]. This state information is saved through the technique of **checkpointing**. This technique is implemented entirely at the user level, without any assistance from the operating system kernel.

To checkpoint a UNIX process, several things must be preserved. The text, data and stack regions are needed, in addition to the register contents. All this information is saved in a file called the checkpoint file. The name of the checkpoint file is *.revi.P* where *P* is the pid of the editor process and it is created in the directory from which the vi process was invoked. The manner in which the checkpoint file is created is explained below.

The SAVE program sends the signal SIGHUP to all the editor processes that are running at the time of logout. A signal handler for this signal has been incorporated in the editor source code. This signal handler saves information about the current window configuration and then sends itself the signal SIGTRAP, which results in a core dump. The SAVE program then combines the original executable file and the core file to produce the checkpoint file.

The checkpoint file is constructed as an executable file with a specific format. In this format, the data and stack segments are obtained from the core file and all other information is obtained from the elvis executable. The elvis executable file consists of a structure called *exec* at the beginning of the file whose elements are defined in the header file */usr/include/sys/exec.h*. The *exec* structure gives the sizes of the various segments of the executable. For example, the field *a_text* in the *exec* structure gives the size of the text segment. The text segment is located immediately after the *exec* structure in the elvis executable. Similarly, the offsets and sizes of the other segments are determined and this information is used to extract these segments from the executable.

In order to extract the data and stack segments from the core file, the structure of the core file should be known. This structure is defined in the header file */usr/include/sys/core.h*. The core file consists of a header called *core* at the beginning of the file. This header is followed by the images of the data and stack regions of the process at the time of the core dump. Finally, the *user* structure of the process (defined in the header file */usr/include/sys/user.h*) is dumped in the core file. Based on the header information, the text size, data size and stack sizes of the process are determined. These sizes are given by the fields *c_tsize*, *c_dsize* and *c_ssize* in the *core* structure. This information is used to locate the data and stack segments in the core file. The segments are then extracted from the core file.

Before writing the extracted information into the checkpoint file, the *exec* structure of the original elvis executable is first read in. The data sizes stored in the *exec* structure are then updated based on the information gathered from the core file. The updated *exec* structure, the segments obtained from the elvis executable and the data and stack segments obtained from the core file are then written into the checkpoint file, in that order.

When the checkpoint file is executed at user login time, it starts executing from the saved data and stack regions and hence the previous state of the editor process is re-created.

Linking Syntax and Semantics

The link between syntax information and semantic information is maintained by the SAVE program in a file called *re_vi* in the user's home directory. The format of this file is as follows: Firstly, the number of editor processes that were active at the time of logout is stored. For each such process, the directory in which the process was executing and the geometry information about the window in which the process was running are stored. This information is necessary since the previous state re-creation includes bringing the user back to the directory in which he was working before. In addition, all the command line arguments are stored since the processes have to be recreated with the same set of arguments. Finally, the name of the terminal emulator program in which the editor process was running is stored.

3.2 Screen Re–creation

We now move on to describing how editor windows are recreated at login based on the saved information detailed above. At the time of login, a program called RECREATE is run from the X–Windows initialization file, namely, *.xinitrc*. The program first reads the *re_vi* file in the user's home directory and determines the number of editor processes that were originally running. For each editor process, it first changes its own current working directory to the directory in which the process was running at the time of logout. It then forks off a terminal emulator program (based on the saved information) with the editor process running inside it with the appropriate arguments. While recreating the editor process, the corresponding checkpoint file is executed instead of the original executable. This ensures that the previous state (including the cursor position) is re-created. Also, the geometry of the editor window is specified based on the saved window geometry information.

The above procedure is repeated for all the editor processes that were running at logout and all of them are recreated. Hence, from the user's perspective, he is provided with exactly the same editing state that he was in at the time of his previous logout.

3.3 Summary

In summary, the REVI editor-recreating system provides the following advantages:

Continued user access to machines : The machine is free to be used by others after the current user saves his environment and logs out. This is the principal advantage over the lockscreen option, where the machine is locked and becomes unusable for others.

Complete release of system resources : There is no trace, in memory, of the current user's program state after he logs out. This is in contrast to the nohup option where the resources are retained even after the user logs out.

Location-independent computing : The state of the editor process can be re-created on any machine on the network, as long as it is compatible with the machine on which the user was working at the time of logout. This is particularly advantageous in a shared computing environment, where the user has transparent file and resource access on any networked machine.

Low disk space overhead : Compression is used to reduce the size of the checkpoint file before it is saved on disk. Since the checkpoint file is approximately the size of the address space of the process, it tends to be large. Compression results in significant amount of saving of disk space (about 60%). The details are available in [Sri95].

In addition to the above, REVI also supports the following features:

Editing of multiple files : If the user is editing several files with a single invocation of the editor program (for example, vi *.c) at the time of logout, then upon re-creation, the user is brought back to the particular file he was editing before and he can then move on to the remaining files in the usual manner.

Multiple saves and re-creations : Suppose the user has saved the state of his editor process (through the SAVE program) and has logged out. When he logs in again, the state of the edit session is recreated. If he makes some more changes to the file and again logs out, then, when he logs in next time, the re-creation is done with respect to the most recent saved state of the editor process.

The complete details of REVI are available in [Sri95]. In our future work, we plan to extend the concept of re-creation to other utility programs such as picture editors (e.g. xfig) and file scanners (e.g. more).

References

[Arf92] Arfman, J.M. et al., "Project Athena: Supporting Distributed Computing at MIT", *IBM Systems Journal*, Vol.31, No.3, 1992.

[Bac92] Bach, M.J., *The Design of the UNIX Operating System*, Prentice Hall (India), 1992.

[Bri89] Bricker, A. et al., "Condor Technical Summary", Computer Sciences Dept., Univ. of Wisconsin, 1989.

[Joy84] Joy, William N. et al., "An Introduction to Display Editing with vi", Computer Sciences Dept., Univ. of California, Berkeley, 1984.

[Lit88] Litzkow, M., et al., "Condor — A Hunter of Idle Workstations", *Proc. of the Intl. Conf. on Distributed Computing Systems*, June 1988.

[Sri95] Sridharan, R., "REVI: A Recreatable Text Editor", *Master's Thesis*, Dept. of Computer Science and Automation, Indian Institute of Science, January 1995.

[Sun86] *lockscreen Manual Pages*, Sun Release 3.5, Sun MicroSystems Inc., 1986.