# Process Capturing

Liviu Iftode Kai Li

Princeton University
Department of Computer Science
Princeton, NJ 08544

**Abstract**

A challenge in mobile computing is to provide continuous interactions between users and their running applications. An interesting model is to allow a user's computing environment to migrate between a stationary computer and a mobile computer while applications are running. To achieve this goal, it is necessary to extend existing operating systems at several levels. This paper focuses on the process management level and proposes a simple mechanism called process capturing. We have implemented a experimental shell and showed that process capturing is a simple and effective mechanism to support computing environment migration.

## Introduction

As the computing technology enters the stage of ubiquitous computing, new types of interactions between users and applications are emerging. The tremendous development in mobile computers and wireless communications allows users to remotely access computing resources while moving from one location to another. With sufficient systems support, it would be possible for users to move computing environments such as shells and windows with mobile computers and to continuously interact with their running applications.

One way to achieve continuous interaction is to use the process migration techniques [PM83, WHFG92, DO87] to migrate applications. Unfortunately, many applications on stationary computers may not be migratible due to the differences in capacity and performance between stationary and mobile computers.

Another approach is to keep an application running on a stationary computer and carry its computing environment on a mobile computer. This approach assumes that the computing environment and the control of the application can move between the stationary computer and mobile computer while the application is running. The method essentially virtualizes the computing environment or working session to make it independent of locations. We call this method computing environment migration.

It is difficult to migrate a computing environment dynamically from one system to another. Many ideas and issues have recently been proposed [Wei91, DM92, Duc92]. The structures and mechanisms of existing operating systems need to be modified to support the functionality of computing environment migration. File systems may need aggressive caching strategies and support for disconnect operations [HHRB92, KS92]. Network connections may require transparent protocols [TYT91, IDM91, CPR92] for mobile computers. Window systems are also required to support transparent migrations [Wei91, Duc92].

This paper proposes a process capturing mechanism to support computing environment migration. The mechanism is a simple extension to the existing Unix process management. We have implemented a process capturing mechanism in Ultrix 4.3 operating system and a capture shell. The shell can migrate shell environments between sessions on the same machine. This capability allows users to continuously interact with their running applications transparently via a remote telnet session on a mobile computer. Our experience indicates that the process capturing is a simple, and effective mechanism for computing environment migration.

## The Capturing Mechanism

In order to migrate a computing environment, the operating system must provide a means to pass environment information and transfer the control of applications from one machine to another.

The existing Unix process management is insufficient for computing environment migration. The Unix process handling is based on static relationships among processes. The relationships are defined at the process creation time. The creator is defined as the parent process and the created process is the child of the creator. The child process is usually a copy of its parent with a different process identification descriptor. As a result, the child inherits the environment of its creator, including opened file descriptors, controlling terminal, signal handling, environment variables and so on. The parent-child relationship also implies a process termination order; the parent process normally will not terminate until all its children terminate. The static relationships among processes make it difficult to transfer the control of an application from one machine to another.

What we need is an extension to the Unix process management that allows a process to become the new parent of other processes. We call such a mechanism *process capturing*. When a process capturs another process, it also captures all the children of the captured process together with their execution environment.

A simple way to implement the process capturing mechanism is to add two system calls to the Unix process management:

- `capture( p )`
  The calling process takes process `p` away from its initial parent. The capturing process becomes the new parent of the captured process and takes over the control of the "terminal" in the context of Unix. Since `p` inherits the Unix terminal of its parent, the opened files being directed to the old terminal will be redirected to the new terminal. Other related process attributes such as opened file descriptors, controlling terminal, signal handling, environment variables and so on will be carried over. In addition, the capturing process will synchronize with the captured process on their terminations. The system call returns the old parent process identifier.

- `release( p )`
  The calling process returns process `p` to its initial parent. If the initial parent of `p` does not exist or the initial parent is the calling process itself, then `release` returns `p` to a default process. This call can be issued by process `p` itself to initiate the control transfer back to its initial parent. The new parent of process `p` will be informed about the release operation. After releasing, the calling process can continue its execution. A subsequent wait on process `p` will return as if process `p` terminated its execution.

In both operations, if a captured process `p` has a tree of descendant processes, then the whole tree of descendants will be moved along with `p`. This can be implemented easily with a top-down recursive algorithm and it greatly simplifies process control for computing environment migration. When a process and its child processes are captured or released, their whole environment will be changed at the same time.

Figure 1 shows the effect of a process capturing operation. When the capture takes place, the captured process and a tree of its child processes will be moved over. The execution environment such as opened

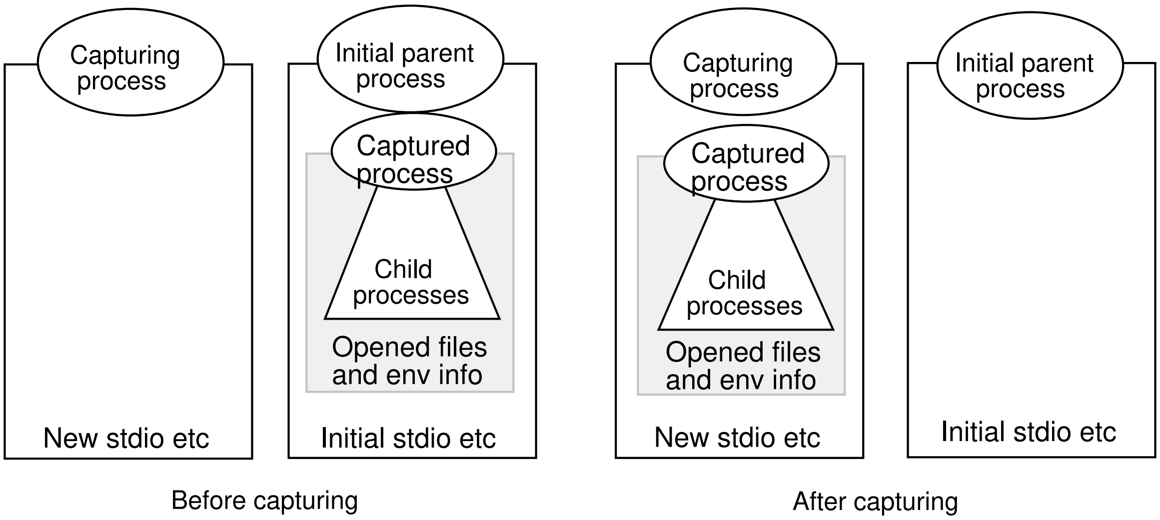


Figure 1: Effect of a process capturing.

file descriptors, controlling terminal, signal handling and so on will be moved along to ensure the correct environment migration.

Process capturing is similar to process migration in the sense that the control of a process is passed away, but it is conceptually different. Process migration moves a process to a different machine whereas process capturing simply transfers the control of the process, modifies the parent-child relationships, and mutates process attributes.

## Implementation Experience

We have designed and implemented the process capturing mechanism in the Ultrix 4.3 kernel on a DECstation 5000/200. We added two new system calls, `capture` and `release` described above. In order to demonstrate the effectiveness of the process capturing mechanism, we have also implemented a simple capture shell.

The shell supports continuous interactions between users and applications. The capture shell can be used to take over a shell under another login session and show all previous interactions including inputs, outputs and related shell commands.

Figure 2 shows three cases: running applications on a stationary machine, invoking a migrating shell on a mobile computer and logging out, and interacting applications on the stationary machine again.

In the first case, a user starts applications under the capture shell on a stationary computer, the same way as using any Unix shell to run applications.

In the second case, the user leaves the stationary machine, and logs in from a remote machine (a mobile computer or a remote workstation) via a `telnet` connection. The user runs the migrating shell with an argument of the `pid` of the shell previously started. It then captures the shell process and its children previously run in the first case. The new shell will show all previous inputs and outputs of the applications, as well as related shell commands issued as if the user had used the remote machine to startup the applications. The user may interact with the running applications and start new applications on the remote machine. Finally, the user logs out. The logout operation will release the shell back to its initial parent process. As a result, the control and the environment of the shell will be passed back.

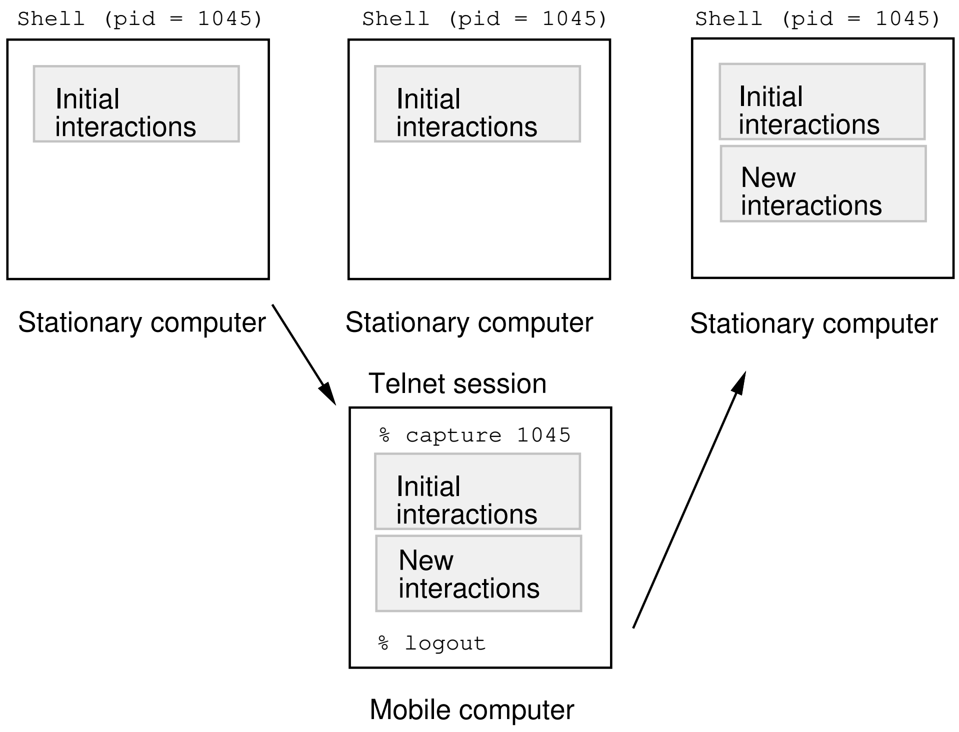


Figure 2: A capture shell

The third case is that the initial capture shell on the stationary machine continues running and show all interactions (inputs, outputs and commands) including those issued in the `telnet` session.

The implementation of the shell is simple. It implements a subset of the Unix `csh`. The shell uses the `capture` and `release` system calls to move the shell environment between stationary and remote connections. The remote connection uses the `capture` call to "capture" the shell and its child processes running on the stationary machine. As the user logs out, the shell on the remote machine calls `release` which return the shell and its child processes to the initial parent process on the stationary connection.

Although the capture shell is used to migrate shell environments between sessions (local login sessions and `telnet` sessions, no process is migrated. Only the control and relationships between processes are changed at process control level. In order to achieve this goal, our implementation of the shell replicates its outputs. In fact, the shell must collect outputs from all its descendant processes. For this purpose, a special file attribute, like COLLECT, is defined and it will be inherited by all descendant processes.

## Summary

This paper shows that the existing Unix operating system is inadequate for computing environment migration, and proposes a process capturing mechanism. Our implementation experience indicates that process capturing is a simple and effective mechanism for computing environment migration and that the mechanism requires only minimal extensions to the existing Unix process management.

Our implementation experience is very limited. We have only implemented a toy prototype shell to demonstrate the usefulness of the process capturing mechanism. We are in the process of collecting performance data on shell capturess and analyzing the performance implications when standard outputs are replicated for transparent capturing.

This paper focuses only on the process capturing mechanism and ignores many important problems related to computing environment migration. For example, we have not addressed issues related to window systems, network connections, security, and architectural support.


### Acknowledgement

We would like to thank Karin Petersen and James Plank for their helpful comments on the draft of this paper. This work is support in part by the NSF Grant CCR-9020893, ARPA and ONR under contracts N00014-91-J-4039, and Digital Systems Research Center and Digital External Research program.


## References


[CPR92] D. Cohen, J. Postel, and R. Rom. IP Addressing and Routing in a Local Wireless Network. In *Proceedings of Infocom*, pages 626–632. IEEE, 1992.

[DM92] F. Douglis and B. Marsh. The Workstation as a Waystation: Integrating Mobility into Computing Environments. In *The Third Workshop on Workstation Operating Systems*, 1992.

[DO87] F. Douglis and J. Ousterhout. Process Migration in the Sprite Operating System. In *Proceedings of the 7th International Conference on Distributed Computing Systems*, pages 18–25, September 1987.

[Duc92] D. Duchamp. Issues in Wireless Mobile Computing. In *The Third Workshop on Workstation Operating Systems*, 1992.

[HHRB92] P. Honeyman, L. Huston, J. Rees, and D. Bachmann. The Little Work Project. In *The Third Workshop on Workstation Operating Systems*, 1992.

[IDM91] J. Ioannidis, D. Duchamp, and G. Maguire, Jr. IP-based Protocols for Mobile Internetworking. In *Proceedings of SIGCOMM '91*, pages 235–245, September 1991. Describes IPIP and IMCP.

[KS92] J. Kistler and M. Satyanarayanan. Disconnected Operation in the Coda File System. *ACM Transactions on Computer Systems*, 10(1):3–25, Feb 1992.

[PM83] M.L. Powell and B.P. Miller. Process Migration in DEMOS/MP. In *Proceedings of the ninth Symposium on Operating Systems Principles*, pages 110–119, 1983.

[TYT91] R. Teraoka, Y. Yokote, and M. Tokoro. A Network Architecture Providing Host Migration Transparency. In *Proceedings of SIGCOMM '91*, September 1991. Also Sony CSL TR #SCSL-TR-91-004. Describes Sony's approach to mobileinternetworking.

[Wei91] M. Weiser. The Computer for the 21st Century. *Scientific American*, pages 94–104, September 1991. Ubiquitious computing defined?

[WHFG92] R. Want, A. Hopper, V. Falcao, and J. Gibbons. The Active Badge Location System. *ACM Transactions on Information Systems*, 10(1):91–102, January 1992.