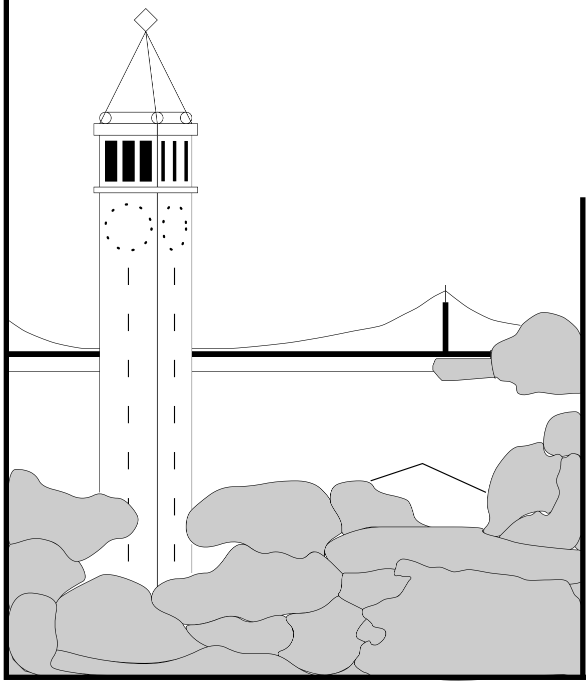
Tracing Windows95

Min Zhou and Alan Jay Smith



Report No. UCB/CSD-99-1037

January 1999

Computer Science Division (EECS)
University of California
Berkeley, California 94720

Tracing Windows95 *

Min Zhou and Alan Jay Smith †

January 1999

Abstract

Most published research on system behavior and workload characterization has been based on either Unix systems or large, usually IBM or IBM-compatible, mainframe systems. It is reasonable to believe that user behavior and workloads are different for PC systems. Further, the aspects of system design most needing study have changed from the mainframes dominant in the 1960s and 1970s, and the Unix systems that became so popular in the 1980s to the PCs that seem to be rapidly taking over many or most aspects of computing. Windows95 is currently the most widely used computer operating system, and is very similar to the newly released Windows98. In this paper, we describe our tracer, which runs on Intel Pentium based PCs running the Microsoft Windows95 operating system. Following the discussion of our tracing methodology, Windows95 operating system, and a tracing tool we developed for Windows95, we give some descriptive and statistical results based on the traces collected from 29 PC users.

1 Introduction

System tracing is widely used for obtaining realistic workloads for computer system analysis and performance tuning. It is a standard technique in computer architecture and operating system research. The traces are often used to characterize the behavior of the system as well as the workload represented by the traces, and can be further used in benchmark development. The traces can also be applied to trace driven simulation to evaluate various algorithms and design-prototypes. Obtaining a set of valid and suitable traces is often the most difficult and important part of trace based analysis. [Smit81]

Multi-user time-sharing computer systems such as UNIX systems and mainframe computer systems have been extensively studied. Much comprehensive system tracing and related work has been done on these systems. [Smit81] [Bake91] [Oust85] [Cost85] [Hans85] Far less system tracing and analysis have been done for personal computers, which have already become the most widely used type of computer. We have/had in mind a number of research studies directed towards the PC environment, and accordingly, our first step has been to write and run a tracer for such systems. Since the most dominant PC operating system and architecture are Microsoft Windows95 and the Intel x86 based Architecture (IA) respectively, we chose Intel based PCs running Windows95 as our tracing target systems. We will call Intel based PCs running the Microsoft Windows95 operating system as "PC systems" in the rest of this paper. We note that the newly released Windows98 is very similar to Windows95, and we believe that user behavior and workload characteristics for Windows98 should also be very similar to those for Windows95.

We believe that traces taken from PC systems will differ from those taken from multi-user time-sharing systems in a number of ways:

First, the workload on a PC system with a single active interactive user is likely to be different from the other systems, both because there is only one user, and because the applications are likely to be different. The interleaved activity of a number of users will differ from the individual streams of activity. Even in the case of a single-user Unix workstation, it is common to have multiple processes active at a given time. Further, we believe that PC users are likely to run much shorter and less computationally intensive jobs. The PCs are more likely to be used by the users for their private work. This is very much the case for those home PC users.

Second, time-sharing UNIX operating systems and Windows95 operate differently. Unlike UNIX,

*The authors' research has been supported in part by the State of California under the MICRO program, and by Microsoft, Fujitsu Microelectronics, Cirrus, Toshiba, Quantum, Sun Microsystems, and Intel Corporation.

†Computer Science Division, EECS Department, University of California, Berkeley, CA 94720-1776

Windows95 was designed to be backward compatible with old software applications including old MS-DOS applications and 16-bit Windows applications. For instance, Windows95 uses an improved version of the MSDOS-FAT format file system, and Windows95 also supports several different memory models. Windows95 is not a true preemptive operating system nor is it a secure operating system, and it is not as stable as larger time-sharing systems as well. [Schu95] [Oney96] [Petz96] [Nort97] We will further discuss Windows95 and its file system, virtual machine, memory system and process scheduling in the next major section.

Third, time-sharing systems and PC system do not always share the same type of application software. Large time-sharing system applications are more likely to be scientific computation oriented or enterprise server software. The most popular UNIX workstation testing workload are the SPEC benchmark suite [Spec98] and TPC database benchmark suite [TPC98]. PC software applications are more likely to be graphic user interface (GUI) oriented, personal information processing intended, and more interactive with the users. MS-Word and Lotus-123 are such examples.

Finally, hardware and software developers of workstation and mainframe computers often emphasize such issues as performance, capacity and reliability. In comparison, PC developers are more concerned with cost, convenience, power consumption, etc.

For this project, we have designed and developed a Windows95 PC system tracer, WMonitor, which collects traces of user and file system activity. In this part of the project, our objective has also been to characterize the PC workload. Our tracing guideline is to achieve a reasonable compromise among the requirements of comprehensiveness, flexibility, minimum user interference and simplicity of analysis.

The selection of users traced will have a significant impact on the characteristics of the workload collected. In this paper, we report on traces collected from 29 real users, including engineers, scientists, managers, home users and school students, using a variety of system configurations. Since we are interested in PC workload characteristics, which include those of users and file systems, our trace events include user inputs, application switches, and file system calls.

The rest of this paper is organized as follows: Section 2 discusses and compares some previous related research. Section 3 discusses our tracing methodology, our tracer design implementation, as well as the tracer installation and experiments. We give information about the users and the systems being traced in the same section. Section 4 discusses the description and the statistics of the traces that we collected. The trace description is followed by the discussion of the tracer overhead and limitations. Finally, section 5 summarizes our tracing and gives the directions of our future work. In the appendices, we provide an overview of Windows95 and a more detailed discussion of the trace formats.

2 Related Work

In this section, we discuss some previous related system tracing work. We concentrate on related work by these others and others at Berkeley, but cite some other work as well; this is a small fraction of the dozens to hundreds of trace based system studies. We also briefly highlight the major differences and similarities between those related tracers and our Windows95 tracer.

2.1 Tracing UNIX Workstations and Mainframe Computers

Da Costa [Cost85] presented a BSD 4.2 UNIX file system tracing package. See also [Zhou85]. Ousterhout et al [Oust85] [Bake91] traced the Sprite distributed file system via kernel instrumentation. In most cases, UNIX tracers were implemented via modifying the UNIX file system kernel. Zivkov et al. [Zivk96] used several sets of mainframe computer traces to characterize disk referencing patterns and study disk caching. Their DB2 disk reference traces were collected from IBM DB2 customer sites using DB2PM, an IBM DB2 performance monitoring package, and GTF, an IBM general tracing package. The IMS disk referencing traces were generated from IMS online system log files. The GCOS traces were collected by instrumenting the GCOS operating system.

The above projects were all primarily focused on the file systems of time-shared multi-user computer systems. Similarly to our Windows95 file system tracing, these file system tracing projects were also done at the logical level, and file system call function names and logical addresses were recorded with time stamps. Most sets of traces were similar to our traces, which cover the file system activities over a period of a few days to one week. Our tracing is different from other tracing projects in that our targets are single user PC systems, and our tracing does not need to modify the kernel code of Windows95.

Hanson et al [Hans85] used a modified UNIX C Shell to trace the user inputs in the UNIX shell command environment. She also combined UNIX accounting information with the user input data in her research on UNIX shell usage characterization. The command line user interface has largely been replaced by the graphical user interface in a modern operating system environment, such as MS-Windows, MacOS and OpenWin. Compared to Hanson's UNIX shell usage tracing, our Windows tracing also collects information on mouse inputs and window switches in addition to the keyboard inputs.

Ruemmler [Ruem93] used a kernel-level trace facility built into HP-UX to trace physical disk I/Os and described the direct disk access patterns. In comparison, our Windows95 tracing does not collect information on computer system activities at the physical level, such as the disk drive direct I/Os studied in [Ruem93].

2.2 Tracing PC systems

Douglis et al [Doug94] traced the file system level disk activities of Apple Powerbook computers. Lorch et al [Lorc97] traced and profiled the system resource usage on Power-PC systems with two tracing tools: "StateProfiler" and "PowerMeasure". These are both MacOS specific.

Li et al [Li94] traced the file system level disk activities of DOS/Windows-3.1. Zhou et al [Zhou96] also traced the user and disk activities of Windows-3.1. These tracing tools used the DOS TSR (terminate and Stay Resident) technique, which is rarely used in Window95. In Windows95, TSR programs can only run from a DOS prompt. The roles of TSR programs have been replaced by virtual device drivers. Lee et al [Lee98] traced and characterized several Windows applications under Windows NT on the x86 processor. They used a binary instrumentation engine, Etch, for the x86-Windows NT in their trace collection. Instruction set level desktop application performance was studied from the perspectives of computer architecture. These desktop applications were contrasted to the programs in the integer SPEC95 benchmark suite.

Intel Corporation [Inte97] developed a Windows "PowerMonitor" to monitor the Windows system device drive access and the processor activities. The implementation of Intel's "PowerMonitor" has taken advantage of the features of a performance counter inside Intel's Pentium processors. Similar to Intel's "PowerMonitor", Chen et al [Chen96] presented a Windows tool which studies the Pentium processor's performance. These two Pentium PC tools are primarily used to monitor the processor activities. They only provide profiling information.

Microsoft also provides a system performance monitor tool with Window95, "System Monitor", or "SysMon" [Chon95]. It provides very rich performance metrics on a variety of system resources: file system read/write, virtual memory page faults, swapfile use, disk cache, processor usage, free memory, thread usage, etc. In comparison, our Windows95 tracer only monitors the logical level file system calls and user input activities. However, Windows95 "SysMon" has a major disadvantage, like Intel "PowerMonitor", is it is a real-time monitor with no data capture capability.

Other relevant PC profiling and tracing tools include Intel's VTune [Inte98], TracePoint Technology's HiProf [Trac98], and Rational Software's Visual Quantify [Rati98]. The VTune tool collects, analyses, and provides Intel Architecture-specific software performance data from the system-wide view down to a specific module, function, and instruction in the application codes. HiProf provides detailed profiling information on applications built with Microsoft's Visual Basic as well as Visual C++ to run on Windows. Visual Quantify is a performance profiling tool for Windows application and software component performance analysis.

3 Methodology

In order to obtain a valid set of traces which can appropriately represent personal computer workload characteristics, three major tracing issues need to be addressed: first, what information should be monitored and recorded; second, how to trace Windows95 and get all the information we need; third, what types of users and machines should be traced.

3.1 Tracing Objects

In this subsection, we discuss what data we collect and why. This tracing project was begun with three end-uses in mind for the data. First, we are studying power management in portable computer systems (see e.g. [Lorc97]), and we wanted to collect those activities reflecting certain aspects of power consumption: user activity and and disk activity. Second, we are interested in extending some of our previous studies in disk caching

[Zivk96] [Smit85] to PC-type systems. Third, we are also interested in characterizing the PC workload, which is the focus of the work described in this paper. A separate paper, which concentrates solely on workload analysis and characterization, and which goes well beyond what is presented here, is available as [Zhou99].

As we discussed earlier, we expect that the workload we observe on the PC will differ from previously studied systems: the operations of personal computer systems are more tightly coupled with user activities (for instance, there are almost no batch or background jobs in PC workloads); the PC workload is more bursty and more GUI oriented; and Windows95 usually does not behave in an optimized way because it was designed to support both 32-bit Windows applications and old MS-DOS as well as 16-bit Windows applications.

Our traces include two parts: user activity traces and file system traces. User activity traces consist of user keyboard input traces, user mouse input traces and active application software traces, i.e. the traces of user-input-focused windows where the user mouse inputs and keyboard inputs are accepted. Since the virtual memory swapping of Windows95 is implemented on top of the file system, our file system traces also include virtual memory swapping information. Our file system traces contain logical file system accesses; physical addresses could be derived with file maps.

In addition to satisfying the requirement of obtaining valid traces, our tracing should also minimize tracing overhead and any interference with the user, and the trace data should be easy to use in analysis. We recognize that these requirements conflict with each other. In our tracing design and implementation, we believe that a reasonable compromise has been achieved.

3.2 Tracing Windows95

In this section, we describe how we use Windows95 standard system services to obtain the user activity traces and file system traces. We will use one figure and several examples to illustrate the way our tracer, WMonitor, works. In appendix I, we provide a description of the Windows95 operating system; a reader not familiar with the appropriate Microsoft software may wish to read that section first.

3.2.1 WMonitor, the Windows95 tracer

Our Windows95 system tracing relies on two standard Windows OS features: our user activity tracing relies on the Windows message hook procedure support, and our file system tracing relies on Windows95's installable file system support.

For user activity tracing, we rely on the fact that windows inter-process communication heavily depends on Windows message passing. User application processes accept user inputs, such as mouse actions and keystrokes, in the form of Windows messages generated by the Windows OS. Windows hook is a mechanism by which a function can intercept user input messages or system event messages before they reach an application. The function can act on events, modify, or discard them. Functions that receive event messages are called filter functions and are classified according to the type of event message they intercept. For instance, a filter function might want to receive all keyboard or mouse event messages. For Windows to call a filter function, the filter function must be attached to a Windows hook, such as a keyboard hook. Windows provides the API of *SetWinodwsHookEx* and *UnhookWindowsHookEx* to the users to maintain and access filter functions. Attaching one or more filter functions to a hook is known as setting a hook. If a hook has more than one filter function attached, a chain of filter functions is maintained in the Windows OS kernel. The most recently attached function is at the beginning of the chain.

Three Windows message hooks are used: WM_KEYBOARD, WM_MOUSE, and WM_CBT (Computer Based Training), to monitor keyboard inputs, mouse inputs, and switches of user-input-focused window, respectively. Since WMonitor message filter functions will be mapped into the logical address spaces of other applications, to which the messages are sent, the WMonitor message filter functions need to be implemented in a dynamic linked library (DLL). Regular Windows EXE applications can be mapped into only one logical address space.

For file system tracing, we rely on the fact that Windows95 allows third party software and hardware vendors to write their own File System Drivers (FSDs) for their products as part of Windows95's file system. These FSDs are in the format of Windows virtual device drivers (VxDs). This file system support is also called Windows95's installable file system support. These installable file system VxDs can be dynamically loaded into the Windows95 kernel. All file system calls will be visible to the installable file system VxDs. A file system call will be processed by an installable file system VxD which claims to process this call. Our file system tracer is written as such an installable file system VxD. However, it does not claim any processing responsibility except examining each file system call.

Figure 1 shows the three major WMonitor modules and related system blocks. It also illustrates the control flow of tracing events. These three parts of WMonitor are:

- WM-VFS.vxd – a Windows95 installable file system virtual device driver which monitors the file system calls;
- MsgHK.dll - a dynamic linked library format module which includes a mouse message hook procedure, a keyboard hook procedure, and a window switch message hook procedure;
- WMonitor.exe - a Windows95 32-bit application which contains a tracer console module (the WMonitor graphic user interface), a tracing message processing module, a buffer management and online analysis module, and a WM-VFS user level call-back procedure.

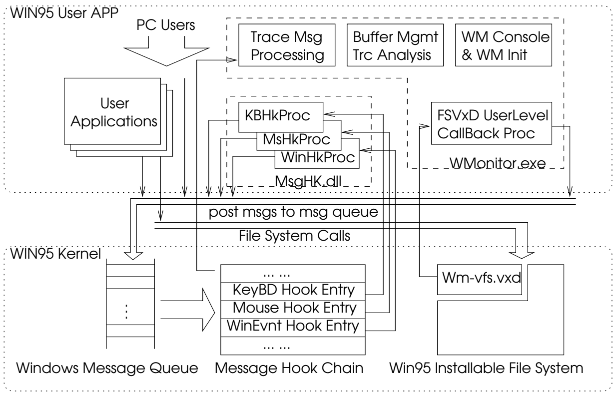


Figure 1: Windows95 Tracing Tool WMonitor Module and Related System Block Diagram

For the example of a mouse input: after a user inputs a mouse click, Windows95 will generate a mouse input message and put it into the Windows mouse message queue. Before this message reaches the current user-input-focused window, the WMonitor mouse message hook procedure has examined this message. A mouse tracing message will be generated from the original mouse message by the WMonitor message hook procedure. The mouse tracing message will be posted to the WMonitor tracing message processing module where this tracing message is processed. In the case of a file system call: after an application makes a file system call, and before the installable file system manager sends the call to a file system driver which will process it, WM-VFS.vxd will read this call and generate a user level procedure call back to WMonitor.exe. This WMonitor user level procedure will post a corresponding file system tracing message to the WMonitor tracing message processing module where this tracing message is processed similarly to the mouse tracing message. Every file system call due to WMonitor trace dumping, which is referred to as a tracer file system call in the rest of this paper, is also recorded for the tracer overhead analysis.

It is very important that the tracer does not affect the workload and regular user behavior. WMonitor is so designed that it will be automatically initiated upon the start of Windows95. There is no need for the users being traced to operate the tracer, since it runs in the background. WMonitor's trace buffer is less than 1 MB. As the reader will see in Table 1, all the PCs being traced have at least 16MB main memory and sufficient disk space. Therefore, WMonitor's file system calls have little impact on the regular user activities.

WMonitor is written in C++ and x86 assembly language. Microsoft Visual C++ 2.0 and Microsoft Drive Development Toolkit (MS-DDK) for Windows95 are our tracer development tools. We also referred to Walter Oney's Windows95 VxD sample code. [Oney96] WMonitor consists of 57K lines of C++ and assembly language code. WMonitor was developed at Intel Corp. when the author worked there as a summer intern.

3.3 Machines and Users Studied

Different PC users perform very different jobs. Laptop PC users may also behave differently from Desktop PC users. It is very difficult to define who are the "typical" PC users and what is the "typical" PC workload. We've attempted to collect as large a number of user traces as possible, over as wide a range of user types and machine types, including both laptop and desktop PCs. The users being traced include engineers, managers, assistants, students, home PC users, and some others. The workload being traced includes software development, computer aided design, logical synthesis and simulation, document writing, Web browsing, remote-dialup, PC game playing, etc.

Our Windows95 tracer was installed on a few home PCs and a number of industry PCs in several corporate sites including Intel Corp., Sony Corp., Toshiba Corp., and Fujitsu Corp. 29 sets of traces are discussed here. Each set of traces was collected from a separate PC machine/user over the period of a few days to a few weeks. Since the portion of the time that each machine was powered on varied a great deal, our tracing time for each user also varies a lot.

Table 1 lists the profile information of the machine and user being traced. We show both calendar time and tracing time. The calendar time for each user/machine is measured by the number of hours between the date/time of the first record and that of the last record. The tracing time for each user/machine is measured by the number of hours when the machine was both powered on and the tracer enabled. "Ratio(Trc/Cal)" is the ratio of tracing time to calendar time. "TrcEvent Count" in the table illustrates the total number of trace events for each user/machine being traced. We also give the average numbers (arithmetic mean value of the sample trace set of 29 traces) and standard deviations for the calendar time, the tracing time, the tracing time ratio, and the trace event count in the same table.

4 Trace Description

In this section, we explain the trace file format and the collected traces. First we describe the traces; details of the trace formats are in Appendix II. Second, we provide some overall trace statistics. Third, we consider tracing overhead. In the end, we discuss the limitations of our tracing work.

4.1 WMonitor trace files

There are two types of trace files: system activity profile log files and system activity trace record data files. WMonitor system profile log files are named as "WM001.log", "WM002.log", ..., "WM999.log". WMonitor trace record files are named as "WM001.dat", "WM002.dat", ..., "WM999.dat". 001, 002, ..., 999 are the three-digit trace sequence numbers. Each sequence number corresponds to a contiguous period of time when the traced PC is powered on and WMonitor is enabled. Both types of trace files are ASCII format text files.

4.1.1 WMonitor system profile logs

System activity profile log files record the following information: USER_ID, StartDate, StartTime, StopDate, StopTime, TotalSectionTime, and activity profiling information. The StartDate/StartTime and StopDate/StopTime are the calendar date/time when WMonitor starts and stops instrumenting the system activities, respectively. TotalSectionTime is the total trace calendar time in seconds between start time and stop time. Profiling information records the number of tracing events in each tracing interval; the default tracing interval is 5 minutes. The interval can be set by modifying the LOG_INTERVAL record in WMonitor initialization file "WMonitor.ini". The profiled trace events include keyboard events, mouse events, window switch events, file system read events, file system write events, file system open events, file system close events, file system seek events, file system delete events, file system directory call events, file attribute events, paging read events, paging write events, other file system call events, and total tracing events. The last activity profiling information record is the total numbers for each type of profiled tracing events. The format of date record is MM:DD:YY. The format of time record is HH:MM:SS. All the numbers in the log files are decimal numbers.

The following file is a WMonitor system profile log example:

```
USER_ID: 380
StartDate: 08/07/97
StartTime: 17:37:41
Time     Keybd Mouse WinEvnt FRead FWrite ... Total
17:42:41 409   251   23      7556  1056   ... 27699
17:47:41 89    33    8       422   73     ... 2005
17:52:41 131   0     0       0     0      ... 131
17:57:41 0     0     0       1     0      ... 2
18:02:41 238   6     0       0     0      ... 244
... ...
Total:   894   339   41      7983  1129   ... 30177
StopDate: 08/07/97
StopTime: 18:16:05
Total_Section_Time: 1704 seconds
```

If the users are only interested in the system activity profiling information, and the details of each tracing event can be neglected, the log files are sufficient enough to serve this purpose, and thus trace record data files can be disabled. By disabling trace record data files, the tracer overhead and trace disk space usage are greatly reduced, and the system activity statistics and profiling information can also be obtained more directly and quickly. For example, if tuning the standard workloads in system benchmarking is the only tracing goal, the log file should

Number	Brand	Model	Type	Memory	Disk (C:/D:/E:)	Calendar-Time	Tracing-Time	Ratio (Trc/Cal)	TrcEvent-Count	Company	User-Type
1	Toshiba	Protégé-610	laptop	16M	687M	199.5 h	30.96 h	16%	1116999	Intel	Engineer/Hdware.
2	Toshiba	Protégé-610	laptop	16M	687M	1028.2 h	54.77 h	05%	1990876	Other	HomeUser/Pilot
3	Digital	HiNote-UltraII	laptop	64M	1372M	344.2 h	116.47 h	34%	8122170	Fujitsu	Director
4	Fujitsu	Lifebk-v655tx	laptop	48M	1293M	503.5 h	153.50 h	30%	4753461	Fujitsu	Manager
5	Fujitsu	Lifebk-v655tx	laptop	48M	1293M	719.3 h	195.21 h	27%	4619375	Fujitsu	Manager
6	Fujitsu	Lifebk-v655tx	laptop	48M	1293M	185.1 h	36.74 h	20%	654949	Fujitsu	Manager
7	Fujitsu	Lifebk-v655tx	laptop	48M	1293M	201.5 h	46.16 h	23%	1129639	Fujitsu	Engineer/IC
8	Fujitsu	Lifebk-v655tx	laptop	48M	1293M	215.5 h	60.97 h	28%	605928	Fujitsu	Engineer/Docmnt.
9	Fujitsu	Lifebk-v655tx	laptop	48M	1293M	715.1 h	179.68 h	25%	4188428	Fujitsu	Engineer/Cad
10	Toshiba	Pentium-PC	laptop	24M	500M	215.3 h	15.13 h	07%	165920	Toshiba	Engineer/IC
11	Toshiba	Satellite-110ct	laptop	24M	775M	215.0 h	36.93 h	17%	613989	Toshiba	Engineer/Docmnt.
12	PC	Pentium-120	desktop	24M	500M/4G	128.5 h	21.22 h	17%	1216834	Intel	HomeUser/Engnr
13	PC	Pentium-90	desktop	32M	1.2G	86.3 h	19.63 h	23%	648849	Intel	Researcher
14	Dell	Dimention-133	desktop	32M	1547M	972.4 h	81.91 h	08%	1447485	Other	HomeUser/Studnt.
15	Compaq	Prolinea-5150	desktop	16M	2G	430.5 h	40.28 h	09%	3648197	Sony	Engineer/Sftware.
16	Sony	PCV-120	desktop	64M	2G/2G	374.2 h	27.82 h	07%	2603819	Sony	Engineer/Sftware.
17	Sony	PCV-120	desktop	64M	2G/2G	438.6 h	120.38 h	27%	4246332	Sony	Engineer/Sftware.
18	AST	MS-T 5166	desktop	64M	2G/2G/1G	459.7 h	51.60 h	11%	9415271	Sony	Engineer/Sftware.
19	Gtw2k	P5-166	desktop	64M	1.5G	377.8 h	307.98 h	82%	9475388	Sony	Engineer/Hdware.
20	Sony	PCV-120	desktop	32M	2G	380.3 h	352.98 h	93%	10053795	Sony	Engineer/Video
21	Sony	PCV-120	desktop	32M	2G	378.7 h	100.96 h	27%	2041658	Sony	Manager
22	Sony	P55c	desktop	32M	2G/2G/1.6G	191.8 h	56.90 h	30%	9166259	Sony	Engineer/Sftware.
23	Toshiba	Pentium-PC	desktop	24M	500M/500M	216.1 h	52.30 h	24%	5240669	Toshiba	Assistent
24	Toshiba	Pentium-PC	desktop	48M	500M/500M	230.8 h	29.96 h	13%	2482814	Toshiba	Engineer/CAD
25	Toshiba	Pentium-PC	desktop	64M	1.2G	215.6 h	46.27 h	21%	954125	Toshiba	Engineer/Progmer
26	Toshiba	Pentium-PC	desktop	48M	1G	238.1 h	59.49 h	25%	1776593	Toshiba	Engineer/Progmer
27	Toshiba	Pentium-PC	desktop	24M	500M/1.5G	188.6 h	42.02 h	22%	5332521	Toshiba	Engineer/Docmnt
28	Toshiba	Pentium-PC	desktop	48M	1.2G	596.1 h	49.17 h	08%	8067977	Toshiba	Engineer/CAD
29	Toshiba	Pentium-PC	desktop	48M	500M/500M	216.8 h	56.82 h	26%	2316542	Toshiba	Clerk
Average (arithmetic mean)						367.7 h	84.3 h	24.4%	3727478		
Standard deviation						238.9	82.6	19.2%	3154425		

Table 1: Profile Data for Machines and Users Traced

be sufficient.

4.1.2 WMonitor trace record data files

Trace record data files record the following data: USER_ID, StartTime, StopTime, and the trace records. The StartTime and StopTime read the Windows95 internal millisecond counter when the WMonitor starts and stops instrumenting the system activities. The Windows95 internal millisecond counter is the elapsed (integer) time in milliseconds since the Windows95 system's most recent start. Each trace record includes four data fields: time stamp, trace type, function name, and information detail. We will further discuss the trace record structure in Appendix II. We can determine the calendar date and time for StartTime, StopTime and each trace record based on the time-stamp and the readings of StartTime record and StartDate record in the corresponding WMonitor system profile log file.

The following file is a WMonitor trace record data file example:

```
USER_ID: 761
StartTime: 9E9C4F
Time  Type  Funct   Details
130   3     OPEN    C: [223] \DAT\WMONITOR.INI
0     3     WRITE   C: [223] 93
0     3     CLOSE   C: [223]
0     3     SEEK    C: [2A8] 4B400:B
0     3     READ    C: [2A8] 200
A     2     [e54] C:\WMONITOR\BIN\WMONITOR.EXE
0     3     READ    C: [271] 1000 MM
0     3     FATTR   C: \WINDOWS\SYSTEM\MFC40LOC.DLL
0     3     FDOPN   C: \WMONITOR\BIN\*.*
DB    0     K_DN    11
96    0     K_UP    91
0     3     FLCKS   C: [26B]
0     3     RENAM   C: \DAT\DATA.ZIP \RECYCLED\DC0.ZIP
0     3     DIR     C: QLGD \WMONITOR\BIN\MSGHK.DLL
0     3     DELET   C: \RECYCLED\DESKTOP.INI
0     1     START_MV
9E    1     STOP_MV
... ...
Stop_Time: 1D41D3C
```

With detailed information and time stamps for every trace event available, WMonitor trace record data files can be used in comprehensive workload analysis and trace driven simulation. Except for the calendar date and time information, WMonitor system activity profiling information log files can be reproduced from the trace record data files.

Detailed information on the trace records is found in Appendix II.

4.2 Trace Statistics

Item	Statistics
Number of Users	29
Number of Trace Files	1546
Total Data Size (compressed data)	2762 M bytes 180.7 M bytes
Total Records	103,461,579
Total Tracing Time	2,443.6 hours
Total Active Time: (idle_time<60 sec)	1,505.6 hours

Table 2: Trace Data Overall Statistics for 29 Traces

Table 2 describes the trace size. As shown in the table, the average size of the compressed trace per user is about 6M bytes. WMonitor does not compress the trace. The archived traces files were compressed with WinZip. The tracing time is defined as the duration during which the traced PC is powered on and WMonitor is enabled. The average tracing time per user is 52 hours. The PC users are active during 62% of the tracing time. We define a user as "active" if one or more tracing events have taken place during the past 60 seconds.

	Total Rec#(std*)	Percent	Rec#/hour
KeyRec	89864 (75611)	2.57%	1637.9
MouseRec	92387 (108497)	2.64%	1271.0
WinRec	3999 (3389)	0.11%	63.9
FSysRec	3227K (2774K)	92.57%	47121.8
MSwapRec	72702 (105648)	2.08%	1314.4
Total	3486K	100%	51408.8

Table 3: Trace Data Breakdown Statistics ("Rec#" is record number, "std*" is the standard deviation.)

Table 3 describes the frequency of tracing events. The data shown in the table are the arithmetic mean values of the 29 users' traces. File system trace records account for 92.6 percent of the total number of trace records. "KeyRec", "MouseRec", "WinRec", "FSysRec" and "MSwapRec" in the table represent keyboard trace record, mouse trace record, window switch trace record, file system trace record, and Windows95 virtual memory swapping file system call trace record, respectively. In this table and the rest of the paper, we will use the term "file system calls" for simplicity whenever we discuss the regular file system calls; this does not include the memory swapping and tracer file system calls. The PC users input by means of mouse devices as frequently as by keyboard. The PC users switch from one user-input-focused window, i.e. a foreground Windows process, to another window as often as about once per minute.

Category	APPLICATION (exe,dll)
WindowsSystem	EXPLORER, SHELL32, COMDLG32, WINHELP, MPRSERV, ...
MsOffice/GrpWare	WINWORD, ACCESS, POWERPNT, EXCEL, COREL70, ...
EngineerTool	MSDEV, DDRAW, ...
MiscTool	NOTEPAD, CALC, ... PHOTOSHOP, WINZIP, ACRORD32, ...
Browser	NETSCAPE, IEXPLORE
DosApplication	MASM, QUICKEN, ...

Table 4: Traced Application Categories

From the user activity traces, we are able to filter out the names of the most frequently accessed Windows applications. We divide these applications into six categories, which are illustrated in Table 4.

Table 5 shows the 35 most frequently run applications ("APPLICATION"). We determine the most frequented run applications by the total time (in seconds/hour) each application was traced to be running ("TIME"). Note that since we don't actually collect CPU traces or OS scheduler traces, we use the active window as the indication of which process is active. This leads to some errors (see [Zhou99] for a further discussion of this issue), but we believe that they are minor. In the same table, we also show the average number of times each application was invoked per hour ("Invoked"), the average numbers of user keyboard/mouse inputs per hour ("KeyEvnt"/"MouseEvnt"), the average number of file system calls per hour ("FSCall"), and the average number of memory swapping file system calls per hour ("MmSwap").

Figure 2 shows the measured idle behavior in PC systems. The Y axis on the left is the system idle time given a certain tracing event interval on X axis. For example, the figure shows that if all the idle periods of 1 second to 2 seconds are added up, a PC system idles 376 second in total per hour. For another example, if all the idle periods of 1 minute

APPLICATION	TIME	Invoked	KeyEvnt	MouseEvnt	FSCall	MmSwap
SCREEN-SAVER	978.26	0.71	0.4	205.8	92056.0	921.2
MSDOS-PROMPT	472.03	20.84	1588.4	3849.1	96143.1	2025.7
EXPLORER.EXE	295.31	10.47	818.3	5762.3	189544.3	6914.6
WINWORD.EXE	227.23	2.71	2669.8	1992.8	72541.5	3018.6
EUDORA.EXE	216.15	1.09	188.5	245.8	8802.2	78.5
MSDEV.EXE	215.72	3.18	333.6	184.8	11064.1	407.1
XVISION.EXE	172.70	0.65	786.0	25.0	5643.0	30.6
NETSCAPE.EXE	167.86	1.54	679.8	2412.9	71988.9	3042.5
SHDOCVW.DLL	139.78	2.10	179.7	1732.3	89917.8	2139.8
EXCEL.EXE	90.72	2.00	709.5	3189.7	93416.5	1906.6
XVL.EXE	76.23	0.32	283.3	87.9	1816.4	19.8
SPTNET32.EXE	66.24	1.39	117.9	18.4	227.3	4.5
POWERPNT.EXE	66.02	0.54	579.0	1625.2	46954.4	3047.3
NOTEPAD.EXE	44.14	0.58	1698.5	2972.2	34740.5	1329.8
EUDORA32.DLL	36.60	0.86	216.4	260.0	16557.8	26.5
HIRAMAIL.EXE	33.48	0.46	0.2	74.2	79.8	0.3
MSOFFICE.EXE	23.41	0.40	1.7	804.0	79919.8	3812.5
WINBIFF.EXE	20.77	0.64	248.9	70.7	52.3	1.1
BRYCE2.EXE	18.40	1.94	6.7	75.8	1841.2	54.4
TELNET.EXE	18.09	0.18	541.6	135.7	4342.6	157.4
RDD.EXE	12.71	0.53	7.6	146.1	3911.6	198.5
COMCTL32.DLL	12.33	1.22	278.7	5355.0	112844.3	2954.8
SUPERTAG.EXE	11.75	0.31	301.4	191.0	646.6	15.6
BPCAP.EXE	11.43	0.06	1.5	56.6	4427.8	88.2
PREMIERE.EXE	11.21	0.63	4.8	95.1	4297.8	58.1
WINHLP32.EXE	11.18	0.70	459.5	8335.2	142248.8	5054.0
SIDEKICK.DLL	11.04	0.06	47.8	12.7	5330.8	65.8
TSTCON32.EXE	10.63	1.29	6.0	140.7	3869.8	15.4
X.EXE	10.60	0.15	138.0	18.3	485.2	3.7
COMDLG32.DLL	9.87	0.85	1077.2	6585.8	108540.3	3338.8
SHELL32.DLL	9.39	2.18	700.4	5342.3	441804.0	11974.6
WINPROJ.EXE	8.65	0.04	10.0	24.2	2709.1	2.1
MSWORKS.EXE	7.89	0.36	136.5	111.8	537.7	41.4
MAILNEWS.DLL	7.40	0.15	687.5	434.3	6491.2	361.3
MPRSERV.DLL	6.04	0.15	1678.9	3821.0	64637.7	988.3

Table 5: The Most Frequently Used Applications ("APPLICATION" is the application name, "TIME" is the total seconds each application was traced per hour, "INOVKED" is the count of number of times each application was invoked per hour, "KeyEvnt/MouseEvnt/FSCall/MmSwap" are the counts of different events per hour.)

to 2 minutes are added up, a PC system idles 156 seconds in total per hour. The Y axis on the right is the percentage of cumulative system active time in the total tracing time. For example, if we set the tracing event interval to 1 second, a PC system is considered to be active for about 43% of the total tracing time. In this case, the system enters idle state from active state, if there is no tracing event taking place within one second after the previous tracing event has completed.

Table 6 gives the percentages of the top 13 most frequently invoked Windows95 file system calls. The percentages are calculated using the following formula:

$$((\sum_{user=1}^{29} \frac{FunctionCallCount}{AllFSCallCount})/29) * 100\%$$

Function Name	Percentage
SEEK	31.64%
READ	24.59%
FINDNEXT	12.18%
WRITE	5.83%
FINDOPEN	4.56%
FINDCLOSE	4.28%
FILEATTRIB	3.74%
OPEN	3.30%
CLOSE	3.14%
GETDISKINFO	3.09%
IOCTL16DRIVE	2.01%
FILETIMES	1.29%
DIR	1.01%

Table 6: Most Used File System Calls

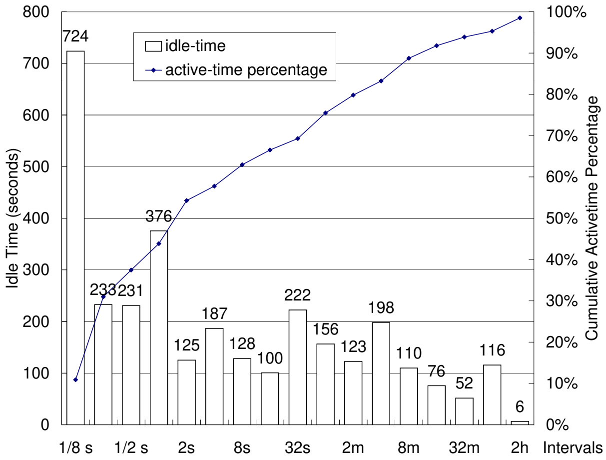


Figure 2: System Idle Time Distribution

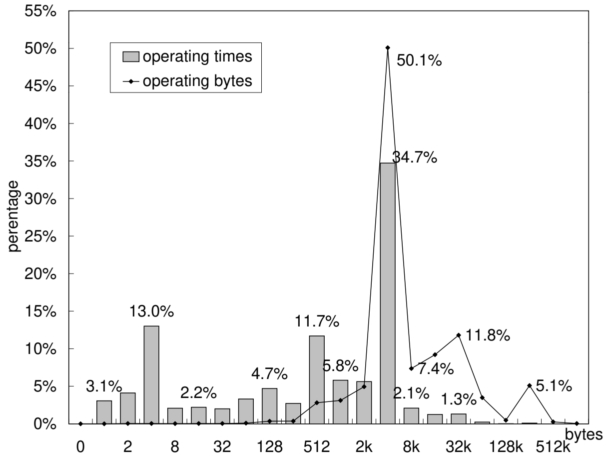


Figure 3: Read File System Call Block Size vs. Access Times and Total Bytes Read

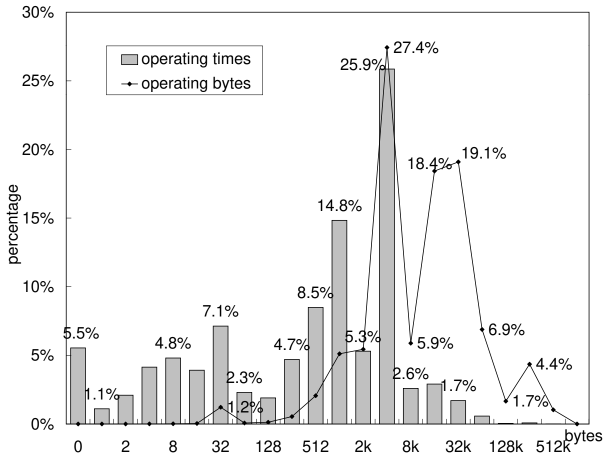


Figure 4: Write File System Call Block Size vs. Access Times and Total Bytes Written

where $FunctionCallCount$ is the count of a certain file system function calls of one user, $AllFSCallCount$ is the total count of file system calls of this user.

Table 7 shows the number of bytes per hour accessed by the file system calls of READ, WRITE, virtual memory READ, virtual memory WRITE, and the percentage of total bytes accessed by each, respectively.

Function	Bytes per hour	Percentage
FS Read	31183 K	71.58%
FS Write	5154 K	11.83%
MemSwap Read	4176 K	9.58%
MemSwap Write	3051 K	7.00%

Table 7: File System Traffic

The next two figures, Figure 3 and Figure 4, show the block size distributions and total read/write bytes for READ and WRITE file system calls. For example, as shown in Figure 3 50.1% of the bytes were transfered as part of the blocks with size of 2049 to 4096 bytes, and 34.7% file READ calls are with these block sizes.

As can be seen from these figures, most block sizes are intermediate, 4KB being the most popular. Since both the Windows95 virtual memory page size and the FAT-32 cluster size are 4K bytes, software designers tend to also use 4KB as the file read or write buffer size.

Figure 5 shows how frequently a PC file is accessed (with an "OPEN" file system call) in a 10-hour period. The X axis represents the number of accesses to one file in 10 hours. The Y axis on the left represents the number of files that have been accessed the number of times given on the X axis, and the Y axis on the right represents the percentage of such files out of the total number of files. For example, there were about 10 files that were accessed ("opened") an average of 16 times per 10 hours. It is intersting to note that an average of 74.8% of the total PC files have only been accessed once during 10 hours. Only 0.55% or 32 files are accessed 50 times or more during the same time period. In fact, a few files were accessed several hundreds times. Some were even accessed a few thousand times. Such a file can be a graphic file, a initialization information file, a temporary cache file, or even a user specific file. Some examples of these popular files are: CONTROL.INI, EUDORA.INI, SYSTEM.INI, UNIMATIC.INI, NPROTECT.LOG, LMOS.TMP, SCROLLBU.TMP, USER.BIN, M0CP9ESI.GIF,

GABENCHMARKS.PPT, USR04993.

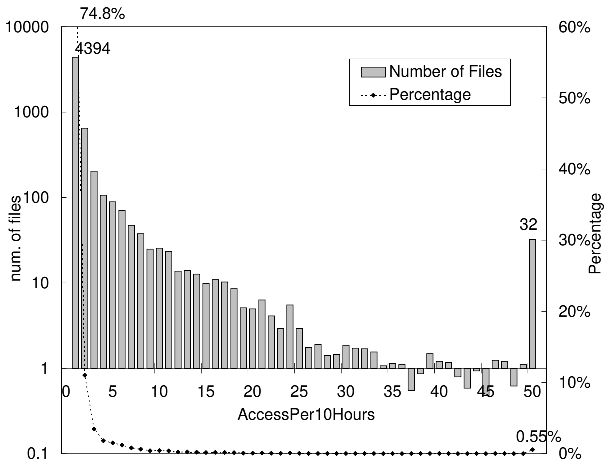


Figure 5: File Access (OPEN) Distribution (There are negative bars in the figure, i.e. some numbers are less than 1. This is because we are showing the number of accesses per 10 hours. Some files were accessed less than once per 10 hours on average.)

4.3 Tracing Overhead

Because we do not trace the processor activity and the user inputs are very sparse, file system tracing dominates our trace. Two types of tracing overheads exist: first, monitoring and generating the trace record; second, dumping the buffered trace records to the hard drives. Every single trace record is generated by only a few tens of lines C code. Compared to the regular file system function processing time, the overhead for generating the trace records is not significant enough to be considered. Therefore, the WMonitor overhead measurement will focus on trace record dumping, i.e. file system call counts contributed by the tracer versus the non-tracing counts of regular file system calls.

	Statistics
Tracer FS Operation Counts	86175
Regular FS Operation Counts	3227379
Tracer FS Overhead	2.66%
Standard Deviation of Overhead	0.351%
90% Confidence Interval	(2.55%, 2.76%)

Table 8: Tracer Operation Overhead

Table 8 shows the tracer overhead and the 90 percent confidence interval (over the 29 samples).

4.4 Limitations of the study

We note the following caveats and limitations in our tracing and analysis:

- Lack of processor activity information. We studied system activity based only on file system, virtual memory and user activities. Our system idleness did not consider processor status, i.e. whether the processor was busy processing a task, or the processor was in an idle instruction loop, or a "HLT" (halt) instruction was active. We did not collect the processor status trace, nor did we collect the Windows process/thread trace.
- The dependency on Windows messaging: User input tracing events are time stamped when the input message gets removed from the user-input-focused window rather than when it is generated. The file system traces are time stamped when the file system trace messages are processed.
- Lack of file system caching information.

It is difficult to estimate the system performance in the absence of processor and caching information. Processor activities were not collected since the original motivation for this tracer was to study power management of/at the user interface and the I/O system.

Since most of our analyses have used time granularities (e.g. 1/4 second) much larger than the time stamp quantum (.001 second), the lack of finer resolution of the time stamp should not be a problem. Even for detailed I/O studies, the .001 second quantum should be sufficient.

In addition, the diversity of PC workload and the user level burstyness shown in our traces make straightforward arithmetic mean values insufficient for detailed analysis. Further detailed PC workload studies and analysis breakdown into different workload categories are needed.

5 Summary and Future work

In this project we have presented a personal computer system tracer and we have discussed some major issues in personal computer system tracing. A set of PC user and file system traces have been collected from a variety of PC users. The traces collected can be used in a number of ways to provide insights to various aspects of personal computer system and user behaviors. The traces can be used in benchmark development as well as for trace driven simulation for different system resource management algorithm studies, file caching studies and power management analysis.

As an extension to this work, the processor activity profiling and system resource management could be integrated with our Windows95 tracer. The features of the Pentium processor's performance counter make the above proposal feasible. Another alternative solution is to follow what Microsoft "SYSMON" does, and use "Windows95's performance metrics stored under the key HKEY_DYN_DATA/PerfStats/StatData with the Windows95 standard Registry API. Given information about the processor activity and other system resource usage, we could establish a more complete PC user and system model. We would be able to analsis the processor activities, and how processors react to the user activities in a Windows environment.

We have presented some general statistics of PC users and systems in this paper. A much more extensive analysis of the workload appears in [Zhou99]. Related and more detailed trace data is currently being collected from WindowsNT systems at UC Berkeley by Jay Lorch. Future projects will use the data collected in these two projects for studies of disk and I/O system optimization, disk design, and power management.

6 Acknowledgments

The Authors would like to thank Intel Corp., for its technical support during the tracer development. The authors also thank Intel Corp., Toshiba Corporation., Fujitsu Microsystems Inc. and Sony Research Laboratories for running the tracer on a number of their machines. The authors owe special thanks to Takashi Miura, Gerry Atterbury, and Act Ratanakul for their help in trace data collection. Tushar Patel and Dennis Reinhardt at Intel Corp. were especially helpful during the tracer development and trace collection. Many other people deserve thanks as well for sacrificing their valuable time and system resources during the trace collection.

Bibliography

[Bake91] Baker, M., Hartman, J., Kupfer, M., Shirriff, K., Ousterhout, J.: Measurements of a Distributed File System *Proceedings of the 13th ACM Symposium on Operating Systems Principles*, Jul. 1991, 1-15.

[Chen96] Chen, J., Endo, Y., Chan, K., Mazieres, D., Dias, A., Seltzer, M., and Smith, M.: The Measured Performance of Personal Computer Operating Systems. *ACM Transactions on Computer Systems.*, Vol. 14, No. 1, Feb. 1996, 3-40.

[Chon95] Chong, H.: The Design and Tuning of Microsoft Windows 95. *Proceedings of the 21st International Conference for the Resource Management and Performance Evaluation of Enterprise Computer Systems*, Nashville, TN, Dec. 1995, 938-949.

[Cost85] Da Costa, H.: A File System Tracing Package for Berkeley UNIX. *Technical Report of Computer Science Division (EECS), University of California at Berkeley, No. UCB/CSD-85-235*, Jun. 1985.

[Doug94] Douglis, F., Kaashoek, F., Marsh, B., Cáceres, R., Li, K., Tauber, J. Storage Alternatives for Mobile Computers. *Proceedings of the First USENIX Symposium on Operating Systems Design and Implementation*, Monterey, CA, Nov. 1994, 25-37.

[Gold74] Goldberg, R.: Survey of Virtual Machine Research. *IEEE Computer*, Jun. 1974, 34-45.

[Hans85] Hanson, R.: A Characterization of the Use of The UNIX C Shell. *Technical Report of Computer Science Division (EECS), University of California at Berkeley, No. UCB/CSD-86-274*, Dec. 1985.

[Inte97] Intel Corp.: Intel Power Monitor. Available as URL http://www.intel.com/ial/ipm/.

[Inte98] Intel Corp.: VTune Performance Enhancement Environment. Available at URL http://support.intel.com/support/performancetools/vtune/.

[Lee98] Lee, D., Crowley, P., Baer, J., Anderson, T., and Bershad, B.: Execution Characteristics of Desktop Applications on Windows NT. *The 25th Annual International Symposium on Computer Architecture.*, Jul. 1998, 27-38.

[Li94] Li, K., Kumpf, R., Horton, P., Anderson, T. A Quantitative Analysis of Disk Drive Power Management in Portable Computers. *Proceedings of the 1994 Winter USENIX Conference*, San Francisco, CA, Jan. 1994, 279-291.

[Lorc97] Lorch, J., Smith, A.: Energy Consumption of Apple Macintosh Computer. *Technical Report of Computer Science Division (EECS), University of California at Berke-*

ley, No. UCB/CSD-97-961, Jun. 1997, to appear, IEEE MICRO.

[Nort97] Norton, P., Mueller, J.: *Peter Norton's Complete Guide to Windows95, Second Edition*, 0-672-31040-6, Sams Publishing, Indianapolis, IN, 1997.

[Oney96] Oney, W.: *System Programming for Windows95*, Microsoft Press, Redmond, WA, 1996.

[Oust85] Ousterhout, J., Da Costa, H., Harrison, D., Kunze, J., Kupfer, M., Thompson, J.: A Trace-Driven Analysis of the UNIX 4.2 BSD File System. *Proceedings of the 10th Symposium on Operating System Principles*, Orcas Island, WA, Dec. 1985, 15-24.

[Petz96] Petzold, C.: *Programming Windows95*, Microsoft Press, Redmond, WA, 1996.

[Rati98] Rational Software Corp.: Visual Quantify. Available at URL http://rational4.rational.com/products/visualq/.

[Ruem93] Ruemmler, C., Wilkes, J.: UNIX Disk Access Patterns. *Proceedings of the Winter 1993 USENIX Conference*, San Diego, CA, Jan. 1993, 405-420.

[Schu95] Schulman, A.: *Unauthorized Windows95: A Developer's Guide to Exploring the Foundations of Windows Chicago*, IDG Books, 1995.

[Smit81] Smith, A.: Long Term File Migration: Development and Evaluation of Algorithms. *Communications of the ACM*, Val. 24, No. 8, Aug. 1981, 521-532.

[Smit85] Smith, A.: Disk Cache-Miss Ratio Analysis and Design Considerations. *Proceedings of the 5th annual Symposium on Computer Architecture*, Apr. 1985, 242-248.

[Smit94] Smith, A.: Trace Driven Simulation in Research on Computer Architecture and Operating Systems. *Proceedings of the Conference on New Directions in Simulation for Manufacturing and Communications*, ed. Moriton, Sakasegawa, Yoneda, Fushimi, Nakano, Tokyo, Japan, Aug. 1994, 43-49.

[Spec98] The Standard Performance Evaluation Corporation: Frequently Asked Questions (FAQ). Available as URL http://open.specbench.org/spec/faq/

[TPC98] Transaction Processing Performance Council: Frequently Asked Questions (FAQ). Available as URL http://www.tpc.org/faq_general.html

[Trac98] TracePoint Technology Inc.: HiProf. Available at URL http://www.tracepoint.com/.

[Zhou85] Zhou, S., Da Costa, H., Smith, A.: A file System Tracing Package for Berkeley UNIX. *Proceedings of USENIX Conference and Exhibition*, Portland, Jun. 1985, 407-419.

[Zhou96] Zhou, M., Smith, A.: A Windows I/O Tracing Package for Notebook PC Power Management. Available as URL http://djinn.cs.berkeley.edu/mzhou/paper/tracer.ps.

[Zhou99] Zhou, M. and Smith, A.: Analysis of Personal Computer Workloads. *Computer Science Technical Report, UC Berkeley, submitted for publication*, Jan. 1999.

[Zivk96] Zivkov, B., Smith, A.: Disk Caching in Large Databases and Timeshared Systems. *Technical Report of Computer Science Division (EECS), University of California at Berkeley, No. UCB/CSD-96-913*, Sep. 1996.

Appendix I: Overview of Windows95

In this appendix, we summarize some major characteristics of the Windows95 operating system.

Windows95 is a 32-bit protected-mode operating system designed to run 16-bit and 32-bit application programs on Intel architecture based personal computers. Windows95 uses the VFAT format file system, a version of MS-DOS FAT file system with long filename support. Windows95 provides up to 4 gigabyte virtual memory. The actual virtual memory size depends on the physical memory and swap space available. Windows95 supports preemptive multitasking of Windows-based and MS-DOS-based applications. Windows95 runs only on PCs based on Intel architecture processors, 80386s or more advanced models. Windows95 does not attempt to provide a secure environment in which program and data can be insulated from another program's inattentive or intentional misbehavior.

A1.1 Windows95 virtual machine

The general concept of virtual machines dates back to early IBM mainframe computers and the work by Robert Goldberg. [Gold74] The virtual machines in the PC world were created when the early versions of Windows needed to support multiple MS-DOS applications and Windows applications running at the same time. [Oney96] [Petz96] A virtual machine created by software reacts to application programs the same way a real machine does, which enables the MS-DOS programs to own the keyboard, the mouse, the display screen, the processor, and the user's attention as if they were running on their own dedicated hardware. Specifically, in the kernel of the Windows95 operating system, a Virtual Machine Manager (VMM) manages all virtual machines. The VMM works with Virtual Device Drivers (VxDs) to simulate hardware devices and to provide system services to applications and to each other. There is at least one virtual machine running on a Windows95 system, the system virtual machine, which runs all Windows applications and the Windows95 system itself. One or more MS-DOS virtual machines running MS-DOS applications can co-exist on a Windows95 system.

A1.2 Windows95 memory model

Generally speaking, Windows95 supports three different memory models: Windows3.1 protected-mode segmented memory model, WindowsNT flat memory model, and Virtual-86 model. In the protected-mode segmented memory model, the processor uses a selector (which points to a segment descriptor entry in the memory descriptor table) and an offset pair to reference a memory location. The virtual memory is divided into segments of up to 64KB each. In the flat memory model, there is only one segment which contains all the programs. Virtual memory with a two-level page table paging scheme is used where each 32-bit address is split into three fields: page table directory pointer, page table pointer, and page offset. Each page frame is 4K bytes. In the virtual-86 mode, 20-bit addresses yield only 1MB of address space. A segment/offset pair is used to generate the 20-bit memory address.

A1.3 Windows95 processes and threads

Each Windows application occupies a process that consists of a dedicated address space and one or more threads of execution. Each thread corresponds to a sequence of program steps and the evolving state of processor registers and system objects associated with that sequence. Windows95 uses a priority-based scheme to preemptively multi-task threads.

Windows95 supports three types of applications: Windows 32-bit application programs, Windows 16-bit application programs, and MS-DOS application programs. Both 32-bit and 16-bit Windows application programs run on the system virtual machine while each MS-DOS application programs run on a separate MS-DOS virtual machine. The system virtual machine has one process for each program, and each 32-bit Windows program can consist of more than one thread. The additional virtual machines are for MS-DOS programs, and each contains exactly one process and one thread.

The 32-bit Windows programs adopt the flat memory model, wherein all code and data can be addressed in a single segment covering all of the virtual memory. The 16-bit Windows programs use the Windows3.1 segmented memory model, in which available virtual memory is subdivided into segments of up to 64 KB each. The 16-bit Windows programs load segment selectors into the processor's segment registers to access more than 64 KB of memory. The 32-bit programs participate in preemptive multitasking under the overall control of the scheduling subsystem of the virtual machine manager, while the 16-bit Windows applications must cooperatively multi-task amongst themselves – from this point of view, sometimes Windows95 is not viewed as a preemptive multitasking system. MS-DOS program multitasking depends on the scheduling among different virtual machines.

Most of the time, one or a few windows are associated with one Windows program. Similarly to the UNIX foreground process, a user-input-focused window in Windows is the foreground window to which the user input will be posted.

A1.4 Window95 file system

Windows95 uses an installable file system manager (IFS manager), the highest layer in the file system, to handle all file system calls from Windows 32-bit applications, Windows 16-bit applications and MS-DOS applications. We will discuss IFS in the next subsection in more detail. The IFS manager calls on file system drivers (FSDs) to support different file system formats. The file system formats currently supported by Windows95 include FAT-16 (File Allocation Table with 16 bit entries), FAT-32 (32 bit FAT entry version of FAT, used in the OSR2 (OEM Service Release 2) version of Windows95 and Windows98 and the CD-ROM file system. The FSDs in turn talk to disk drivers which interface with the hardware directly.

A FAT (including FAT-16 and FAT-32) format disk consist of a BOOT sector, a file allocation table, a root directory, and a cluster section. BOOT stores the basic information about the disk and for the use of system boot. The root directory stores the information describing each file entry in the top level directory. The disk cluster section is divided into separated clusters. The notion of a cluster, which is a contiguous collection of disk sectors, was introduced as the allocation unit. Each FAT table entry is used to maintain the status of a disk cluster, and the number of FAT table entries is equal to the number of the clusters on a disk. A FAT table is organized as a linear array containing multiple one-way linked lists. One list corresponds to a file or sub-directory. The FAT entry location of the head of each list is stored in the root directory or a sub-directory. In a FAT file system, sub-directories are stored as regular files.

FAT-16 uses a fixed FAT table size (32KB), 16-bit FAT table entries, and variable cluster sizes. FAT-16 supports up to 2GB per logical hard drive. A hard drive larger than 2GB needs to be partitioned into a few logical hard drives for a FAT-16 format file system. For example, FAT-16 uses 32KB cluster for a 2GB hard drive, 16KB cluster for a 1GB hard drive, ... , 4KB cluster for a 128MB hard drive, etc. Different from FAT-16, FAT-32 uses a variable FAT table size, 32-bit FAT table entries, and a fixed cluster size (4KB). It supports up to a 2TB hard drive. Our target systems all use the FAT-16 format file system for their hard drives.

The FAT file system used in Windows95 file system is called VFAT, virtual FAT – an improved version of the old MS-DOS FAT format file system plus long filename support. Similar to FAT, VFAT also can be classified as VFAT-16 and VFAT-32. The VFAT file system has two file names for each file, a DOS-8.3 format filename (maximum 8 bytes for the file name and maximum 3 bytes for the file name extension) and Windows95 specific long filenames which can be as long as 256 bytes.

A1.5 Windows95 Installable File System and IFS Calls

The file systems of both Windows3.1 and MS-DOS depend on MS-DOS's INT21 code to manage files on disk. Since MS-DOS INT21 is not reentrant, multiple processes cannot simultaneously perform file system calls without proceeding one at a time through this critical section. Windows95 relies on the Installable File System Manager to solve this problem and support asynchronous I/Os. All file system calls of Windows 32-bit applications, Windows 16-bit applications and MS-DOS applications go to the IFS manager. These file system calls include the accesses to the memory swap file as well. IFS manager calls on FSDs to implement diverse file systems like FAT and the CD-ROM file system. The FSDs talk to disk drivers which interface with the hardware components such as hard drive and floppies directly.

The IFS manager exports a number of virtual device driver level services for use by other parts of the system. These IFS services and Windows95 virtual device driver's dynamic loading, which was designed for plug-and-play, allows third party software and hardware vendors to write their own device drivers as part of Windows95 file system. One of the most important services provided by IFS manager is the IFS_Mgr_InstallFileSystemApiHook service. IFS_Mgr_InstallFileSystemApiHook takes the address of the user VxD hook procedure as an argument, and it returns the address of another hook procedure. A VxD hook procedure is a VxD procedure which will be trigged when the hook-targeting system service is invoked. All the VxD hook procedures should chain the call instead of just processing it to give other potential hooks their chance to examine each request to the targeted system services. Internally, the IFS manager maintains its own list of API hooks so that the users can add and remove the hooks in any order.

There are 31 most commonly used Windows95 installable file system calls generated by IFS manager. These calls are our file system tracing targets. Next we give the names and explanation of these installable file system calls.

- *FS_ReadFile* transfers data from the file to a memory buffer. The memory buffer can be filled asynchronously using one or more I/O requests. In a regular FSD implementation, Windows95 VCACHE facilities should be used to maintain a cache of disk records to minimize the physical I/O.
- *FS_WriteFile* transfers data from a memory buffer to the file. A cache of disk-sector-sized buffers containing the data should be maintained and the physical write operations should be performed asynchronously.
- *FS_FileSeek* is an advisory service that allows an FSD to optimize its prefetches of a file. This function is advisory because the read and write functions both supply a file position that overrides anything recorded by the FSD.
- *FS_OpenFile* takes indicated actions to open a file which matches the parsed pathname.
- *FS_CloseFile* flushes any output buffers to disk, deletes internal structures related to the file, and generally cleans up after a series of operations on an open file.
- *FS_CommitFile* flushes buffered data of a file handle to disk.
- *FS_EnumerateHandle* enumerates file handle information.
- *FS_HandleInfo* gets and sets information for a file by the file handle.
- *FS_LockFile* locks or unlocks a byte range in a file by the file handle.
- *FS_FileDateTime* sets or retrieves the timestamps which associate with an open file. There are three Windows95 file timestamps: creation time, last-modified time, and last-accessed time.
- *FS_DeleteFile* deletes the files whose parsed pathname appears in the request pathname.
- *FS_Dir* performs a function on a directory. Directory functions include creating, deleting, checking for the existence of a directory, or converting a directory name between its long-name form and its 8.3 form.
- *FS_DirectDiskIO* is called by IFS manager to handle MS-DOS INT 25h and INT 26h (absolute disk read and write) requests.
- *FS_DirectVolumeAccess* performs direct volume (file system storage resource logical unit) accesses.
- *FS_ConnectNetResource* connects or mounts a network resource.
- *FS_DisconnectResource* is the function to take the actions required when one of the FSD volumes is unloaded or deleted.
- *FS_FileAttributes* gets or sets the attributes of a file.
- *FS_FindChangeNotifyClose* and *FS_FindNextChangeNotify* search for file change notifies on a certain disk drive.

- *FS_FindFirstFile*, *FS_FindNextFile* and *FS_FindClose* go together to implement a normal file search. *FS_FindFirstFile* initiates a file search that can include wildcards, and creates a context handle. *FS_FindNextFile* continues the search with the context handle until no more matches are possible. *FS_FindClose* closes the context handle.
- *FS_FlushVolume* flushes any pending output data to the device.
- *FS_GetDiskInfo* retrieves information about the free space on a disk drive.
- *FS_GetDiskParms* returns the real-mode address of the MS-DOS disk parameter block.
- *FS_Ioctl16Drive* performs an I/O control operation on the volume.
- *FS_QueryResourceInfo* provides basic information about the file system to the IFS manager.
- *FS_RenameFile* renames one or more files. Wildcards in the source name can be specified by the user.
- *FS_SearchFile* is the MS-DOS equivalent of the *FS_FindFirstFile* family of functions.
- *FS_TransactNamedPipe* performs named pipe operations.
- *FS_UNCPipeRequest* performs UNC path based named pipe operations.

Appendix II: WMonitor trace record data files

Trace record data files record the following data: USER_ID, StartTime, StopTime, and the trace records. The StartTime and StopTime read the Windows95 internal millisecond counter when the WMonitor starts and stops instrumenting the system activities. The Windows95 internal millisecond counter is the elapsed (integer) time in milliseconds since the Windows95 system's most recent start. Each trace record includes four data fields: time stamp, trace type, function name, and information detail. We further discuss the trace record structure in next subsection. All numbers are hexadecimal numbers except the number in USER_ID record in a trace record data file. One trace record spans exactly one text line with the return and line-feed characters, $0D$ and $0A$ in ASCII code, as the line separator. We can determine the calendar date and time for StartTime, StopTime and each trace record based on the time-stamp and the readings of the StartTime record and the StartDate record in the corresponding WMonitor system profile log file.

The following file is a WMonitor trace record data file example:

```
USER_ID: 761
StartTime: 9E9C4F
Time  Type  Funct   Details
130   3     OPEN    C: [223] \DAT\WMONITOR.INI
0     3     WRITE   C: [223] 93
0     3     CLOSE   C: [223]
0     3     SEEK    C: [2A8] 4B400:B
0     3     READ    C: [2A8] 200
A     2     [e54] C:\WMONITOR\BIN\WMONITOR.EXE
0     3     READ    C: [271] 1000 MM
0     3     FATTR   C: \WINDOWS\SYSTEM\MFC40LOC.DLL
0     3     FDOPN   C: \WMONITOR\BIN\*.*
DB    0     K_DN    11
96    0     K_UP    91
0     3     FLCKS   C: [26B]
0     3     RENAM   C: \DAT\DATA.ZIP \RECYCLED\DC0.ZIP
0     3     DIR     C: QLGD \WMONITOR\BIN\MSGHK.DLL
0     3     DELET   C: \RECYCLED\DESKTOP.INI
0     1     START_MV
9E    1     STOP_MV
...  ...
Stop_Time: 1D41D3C
```

With detailed information and time stamps for every trace event available, WMonitor trace record data files can be used in comprehensive workload analysis and tracing driven simulations. Except for the calendar date and time information, WMonitor system activity profiling information log files can be reproduced from the trace record data files.

A2.1 WMonitor trace record structure

Each trace record in a WMonitor trace record data file includes up to four data fields: time stamp, trace type, function name, and detail information. The trace record data fields are separated by the character of ASCII code 09. Each trace record contains up to 539 bytes. When a trace record reaches its maximum length, two full Windows95 long file pathnames, each of 256 bytes, are included.

The time stamp records the time when a traced event triggers a WMonitor procedure. The incremental time stamp is used to reduce the record size. The absolute time stamp can be derived by accumulating the incremental time stamps and then adding the StartTime. The granularity of time stamp is one millisecond. The upper limit of this time stamp is 0xFFFFFFFF.

Trace type can have one of the following four values:

- 0 – keyboard input event
- 1 – mouse or other pointing device input event
- 2 – user-input-focused window switch event
- 3 – file system call event

Table 9 also lists all trace record types in a WMonitor trace record data file. Different types of trace records interpret the function name field and detail information field differently, which we will discuss in the following two sub-sections.

A2.2 User activity trace record

There are three types of user activity trace records: keyboard input record, mouse input record and user-

Trace type	Explaination	Data field
USER_ID	user id	user-identification
StartTime	start time	trace-starting-time*
Stop_Time	stop time	trace-stopping-time*
0	keyboard input	keyboard-event key-scancode
1	mouse input	mouse-event
2	window switch	[window-handle] application-name
3	file system call	see Table 10

Table 9: WMonitor Trace Records(* is the Windows internal milisecond counter readings when tracing starts/stops.)

function name	detail information field	IFS call name
READ	diskdrive fhandle[1] bytes vm_opt[2]	*FS_ReadFile*
WRITE	diskdrive fhandle bytes vm_opt	*FS_WriteFile*
FDNXT	diskdrive handle[3]	*FS_FindNextFile*
FCNNT	diskdrive	*FS_FindNextChangeNotify*
SEEK	diskdrive fhandle bytes position[4]	*FS_FileSeek*
CLOSE	diskdrive fhandle	*FS_CloseFile*
COMMT	diskdrive fhandle	*FS_CommitFile*
FLCKS	diskdrive fhandle	*FS_LockFile*
FTMES	diskdrive fhandle	*FS_FileDateTime*
PIPRQ	diskdrive	*FS_TransactNamedPipe*
HDINF	diskdrive fhandle	*FS_HandleInfo*
ENMHD	diskdrive	*FS_EnumerateHandle*
FNDCL	diskdrive handle[3]	*FS_FindClose*
FCNCL	diskdrive	*FS_FindChangeNotifyClose*
CNNCT	diskdrive	*FS_ConnectNetResource*
DELET	diskdrive filename	*FS_DeleteFile*
DIR	diskdrive method[5] filename	*FS_Dir*
FATTR	diskdrive filename	*FS_FileAttributes*
FLUSH	diskdrive	*FS_FlushVolume*
GDSKI	diskdrive	*FS_GetDiskInfo*
OPEN	diskdrive fhandle filename	*FS_OpenFile*
RENAM	diskdrive filename1 filename2	*FS_RenameFile*
SEARC	diskdrive filename	*FS_SearchFile*
QUERY	diskdrive	*FS_QueryResourceInfo*
DISCN	diskdrive	*FS_DisconnectResource*
UNCPR	diskdrive	*FS_UNCPipeRequest*
IOC16	diskdrive	*FS_Ioctl16Drive*
GDSPR	diskdrive	*FS_GetDiskParms*
FDOPN	diskdrive filename	*FS_FindFirstFile*
DSDIO	diskdrive	*FS_DirectVolumeAccess*

Table 10: File System Call Records (fhandle[1] is the file handle in the format of "[hex-number]". vm_opt[2] is the virtual memory operation indicator which can be null (i.e. " "), or one of "PG" or "MM". handle[3] of FDNXT and FNDCL is the file searching context handle. position[4] can be one of "begin", "end", or "current". method[5] can be one of "mkdir", "rmdir", "chechdir", "query8.3dir", or "querylongdir".)

input-focused window switch record.

- keyboard input record: For a keyboard input event, the function name is either "K_DN" (pressing a key) or "K_UP" (releasing a key). The 1 byte (7 valid bits) key scan code is stored in the detail information field. The 8th bit of this byte indicates the status of the key being accessed: 1 – up and 0 – down.

  For example, if the input is a capital ASCII "K", four keyboard trace events are recorded: 0x2A (scan code of "left_shift") K_DN, 0x25 (scan code of "k") K_DN, 0xA5 (scan code of "k" + 0x80) K_UP, and 0xAA (scan code of "left_shift" + 0x80) K_UP, where "left_shift" K_DN and "left_shift" K_UP are not generated if Caps_Lock is in function.
- mouse input record: For a mouse input event, the function name can be one of the following mouse events: L_DWN (pressing the left mouse button), L_UP (releasing left mouse button), L_CLK (double clicking the left mouse button), M_DWN (pressing middle mouse button), M_UP (releasing middle mouse button), M_CLK (double clicking middle mouse button), R_DWN (pressing right mouse button), R_UP (releasing right mouse button), R_CLK(double clicking right mouse button), START_MV(starting moving the mouse), and STOP_MV(stopping moving the mouse). The detail information field is empty for the mouse input event case.
- user-input-focused window switch record: For a window switch event, the window handle and the application software full pathname are stored in the function name field and detail information field, respectively.

A2.3 File system call trace record

Table 10 gives a list of file system function call names, the corresponding detail information fields, and installable file system call names. We discussed the installable file system calls previously.

File system call trace records include both regular file access call records and memory swap file access call records. Memory swap file access call records are mapped memory reads, mapped memory writes, memory paging reads, or memory paging writes. Memory swap file access records distinguish themselves from regular file access records by the last two bytes in the detail information field: either "MM" (Mapped Memory) or "PG" (memory PaGing).