

Background Multitasking

(A Technology Report for IT Professionals)

Preface: This report is a general overview of technologies, intended to provide a better understanding of certain limitations inherent in modern operating systems. It is not intended as the presentation of a scientific study. Note, for sake of brevity and bearing, descriptions of operating system design and behavior, while wholly accurate, are presented in simplified form.

OVERVIEW

Multitasking is a method for sharing system resources so that multiple processes¹ can *appear* to run simultaneously. An example would be the ability to run both a word editor and a spreadsheet application at the same time.

However, a component in a general purpose computer, such as the CPU, can only execute one task at a given time. Multitasking presents the illusion that all execution is occurring simultaneously by implementation of a thread² scheduling system. It is in essence virtualization of the CPU, designed to fool applications into believing they own the CPU exclusively. More powerful and faster CPUs can execute larger tasks or execute more in a given span of time.

Multiprocessing extends the principles of multitasking to multiple CPUs, actually allowing threads to run simultaneously, where multitasking only feigns this behavior. Multiprocessing, like multitasking, is still limited in processing capacity to one action at a time on a given CPU.

For the purposes of this paper, multitasking will be used to refer to the operating system's thread handling.

Inside Multitasking:

There are a variety of Multitasking techniques in use in operating systems today. Modern computer systems such as Windows®, Linux®, UNIX® and Mac OS® X use a form of multitasking called *preemptive multitasking* with varying unique differences. Pre-emptive multitasking is a more effective mechanism to guarantee resource sharing as it differentiates processes (e.g. I/O or CPU) and integrates I/O waiting so as to not delay processes that do not need I/O device responses (such as data returned from a hard drive).

Windows NT's® (up through and including XP – i.e. NT 5.1) application of preemptive scheduling, while not a pure form, is more advanced, when compared to earlier MS-DOS® based versions of Windows, in that it better separates processes by the resources they use (e.g. CPU intensive versus I/O bound). Of key importance is that it also allows processes to be *prioritized*. Prioritization involves implementing a system by which one process, and its associated threads, can be deemed to be more, equally, or less important than another process. An essentially “round-robin” scheduling system then assigns resource time-slices available to the active processes.

However, the prioritization value system is relatively rote in that it relies on the process to generally define how important it is. It also applies a numeric value system (values of 0-31) in which a higher number indicates greater importance. A process with a value of 8 will receive greater resource access than a process with a priority value of 4, but less than a process running at a priority of 13.

Processes that share a number value (e.g. word editor and spreadsheet that both have a value of 8) are considered equal, and must essentially “split” resources equally.

Ideal resource sharing systems would know more about unrelated processes that share a priority value, and be able to further subdivide resources. Then, based on actual needs at a given time, they could more effectively share resources, rather than relying on a relatively rote hard-coded priority value. Windows NT preemptive multitasking does not provide this.

Preemptive multitasking does provide a very basic implementation of dynamic resource assignment in order to ensure better system resource dissemination between high and low priority processes. The NT thread scheduling mechanism

¹ A process is the control component that manages a related thread (or threads).

² A thread is the actual code, managed under the context of a process, that will be executed by the system.

can dynamically adjust (increase or decrease) a priority value for a given process relative to its assigned base priority and to its I/O-CPU burst³ characteristics (i.e. a CPU bound process versus an I/O bound process) to afford eventual access to resources. The justification for priority changes is simply dependent upon whether a process is being *starved* resource time due to an influx of high priority processes. In these cases, the scheduler can dynamically reduce the priority of processes and increase the right to access resources for others.

The phenomenon related to this occurrence is that the scheduler will actually provide a lower priority I/O bound process a larger time slice than it normally would if it hadn't been starved. While the use of interrupts⁴ allows the blocking of I/O bound processes, it's important to note that a process will not be starved for more than 3-4 seconds. With processors operating in the gigahertz, a few seconds is retrospectively eons, but it will occur.

As one can see this system is beneficial to ensure that all processes get access to resources and that all processes are ordered in at least some simple form of relative importance.

However, some shortcomings can be uncovered with this design. It can be shown that some applications active on a system, that are allocated resources every 3-4 seconds, can and should be starved resources until no other process requires them. And, as suggested, prioritization amongst seemingly "equal" applications is relatively rote.

It becomes obviously inadequate when true low-priority applications that can wait indefinitely (or near-indefinitely) for resource allocation, are force-fed resources.

INVISITASKING™ – “True Background Processing – Delivered”

InvisiTasking is new technology designed to enhance multitasking and address some of the shortcomings related to the lack of “enough information”. Having more information allows better sharing of resources. InvisiTasking is specifically designed to address the “background” application to ensure it truly does run in the background and does not interfere with higher priority processes such as transaction processing⁵, print queuing and quite frankly, anything else other than wasteful system idle time.

Unlike multitasking, which is integrated in the operating system's kernel, InvisiTasking currently works in user-mode. This means it will not presently solve all the issues inherent in the core of multitasking for a background application, but it will mitigate the inherent shortcomings to the point they are no longer issues. It, of course, requires an application to be written to leverage this technology.

Inside InvisiTasking:

As CPU and I/O resources are almost never fully utilized, InvisiTasking's transparency is achieved by undetectably tapping into these unused system resources. Software engineers sometimes attempt to share resources by choosing lower CPU priorities to run under, and past efforts have been made at throttling disk and network I/O. However the goal of truly transparent software has never before been achieved until now.

To accomplish true transparency one needs to be able to monitor CPU, memory and the more significant hardware bottlenecks of the disk drive and network. InvisiTasking takes a pro-active approach to instantly detect resource usage while maintaining complete granular control over its own activity, ensuring that it *never* pre-empts users or services.

The key of course, is to prove that InvisiTasking theory holds true in practice...

³ A burst is an unexpected, occasional transmission of data.

⁴ An interrupt is a notification, to a program, that something has happened (e.g. user type a key on the keyboard). It can trigger a subsequent action(s) or simply be ignored.

⁵ The action of performing an I/O operation that changes data in the file system.

BENCHMARKING INVISITASKING

To demonstrate the effectiveness of InvisiTasking, a couple of different benchmarking tests were performed.

A 32-bit tool called DkCopy was written by Diskeeper® Corporation to demonstrate the benefit of InvisiTasking. DkCopy is a simple, straightforward file copy tool that copies a file from one location to another (standard mode). DkCopy in this standard mode is representative of typical business applications such as SQL, Lotus Notes®, Microsoft® Office, etc.

To demonstrate the benefit of InvisiTasking, the utility also includes a special qualifier to apply InvisiTasking technology to its file copy operations. Applying this qualifier can turn DkCopy into a true background application.

Two independent tests with slightly different metrics were completed. Both tests used the same platforms and environments.

InvisiTasking Test 1

The purpose of Test 1 is to test how quickly, by measuring time-to-complete, the standard copy operations of DkCopy completed when *another* process was also running. The ramification of executing a second program concurrent with the primary program should naturally show impact upon the first program.

In this case that *concurrent* application was also DkCopy, implementing the switch that enables InvisiTasking. Again, this test is designed to detect any impact that DkCopy might experience when another program was running concurrently.

DkCopy with InvisiTasking enabled represents a background application, such as a maintenance process, for which the developer intends to prevent any negative overhead that would degrade performance of typical business applications.

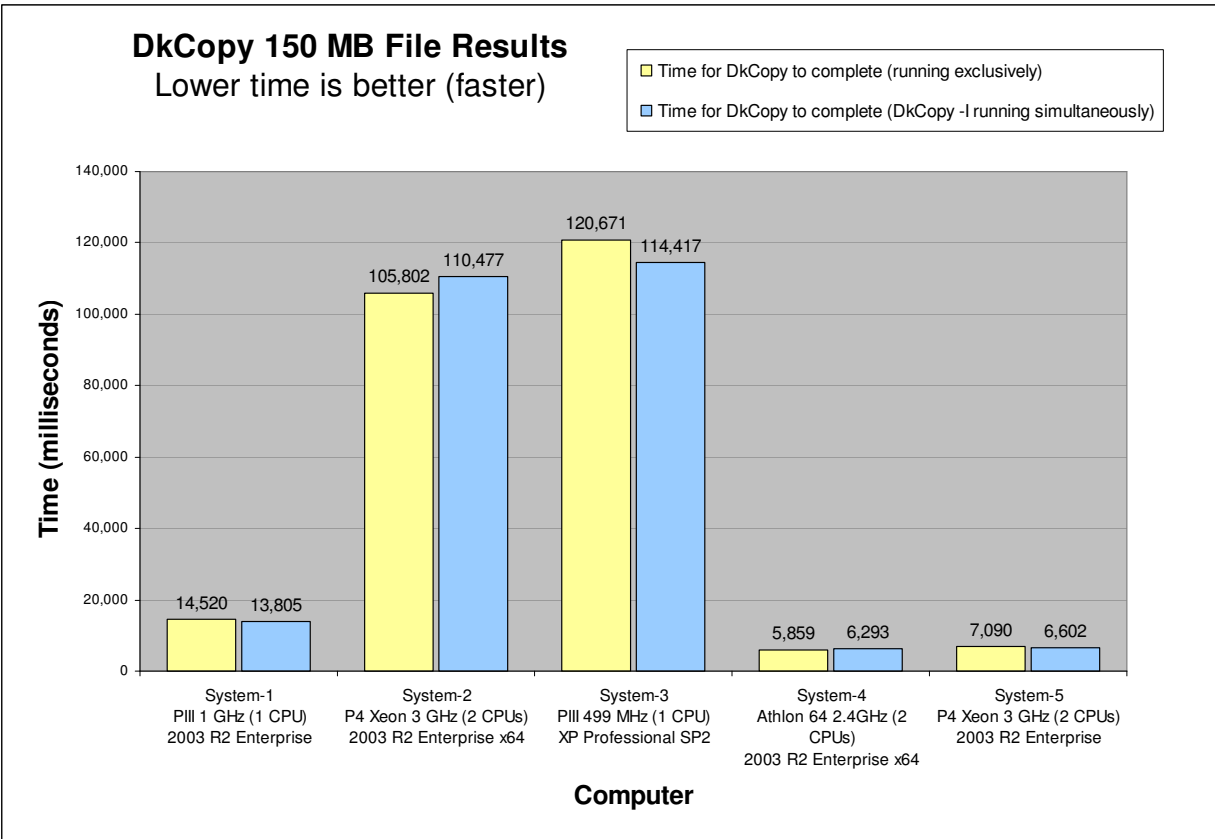
Test 1 Procedure:

1. DkCopy is run using its normal operations to copy a file [DkCopy.exe]. This is to establish a baseline for the time this tool will take to complete its actions.
 - a. Test 1A copying a 150MB file
 - b. Test 1B copying a 890MB file
 - c. Text 1C copying a 6GB file
2. Two instances of DkCopy were run simultaneously. One operating normally [DkCopy.exe] and one using the InvisiTasking option [DkCopy.exe -I].
 - a. Test 1A copying a 150MB file
 - b. Test 1B copying a 890MB file
 - c. Text 1C copying a 6GB file

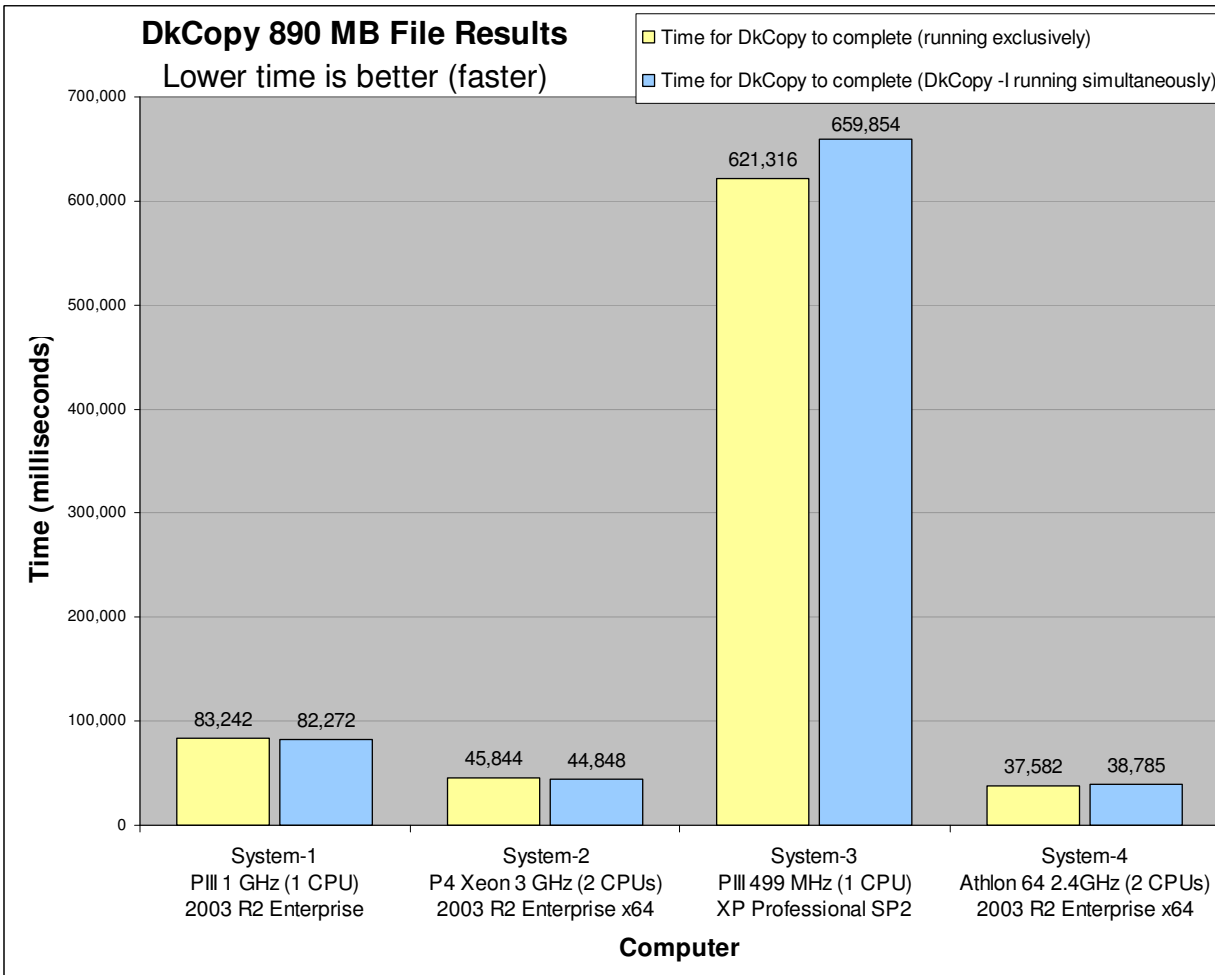
In the tests (stage 1 and 2), each of the DkCopy tests were run a total of six times for each of three different file sizes (150MB, 890MB, 6GB), with the results recorded. The highest and lowest scores were discarded and the remaining 4 scores averaged⁶ and represented in the chart shown. The second instance of DkCopy (with InvisiTasking enabled) was performed in such a way that ensured no I/O conflict or caching issues would skew results. The system was reset to an identical state in between all test runs to ensure identical starting environments for comparative testing.

⁶ The lowest and highest values were dropped, and the remaining four values were used in the average times.

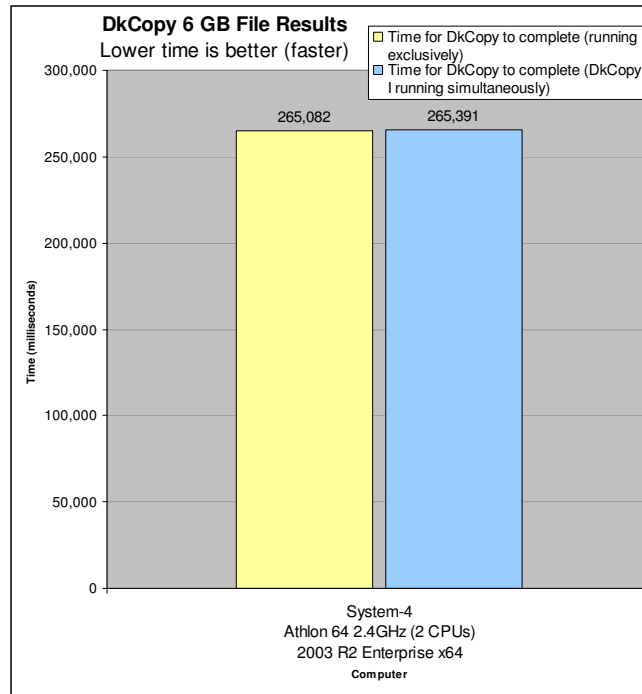
Test 1 Results:



DkCopy (150MB file)



DkCopy (890MB file)



DkCopy (6GB file)

Test 1 Summary

The graphs represent the time the copy took to complete, so lower numbers indicate faster runtimes. Varying levels of hardware were used ranging from Intel Pentium III 500 MHz processors to Dual Athlon 64 2.4 GHz processors.

As the results show InvisiTasking incurred only minimal performance impact in half the cases, and actually slightly *enhanced* performance (execution time) in the other half of tested cases.

InvisiTasking Test 2

The purpose of Test 2 was to measure the system performance via a third party benchmarking tool. The well known benchmarking software product, PCMark®, was selected to measure PC performance, as it is reputable and non-proprietary. PCMark (PCMark05 Revision 1 Build 0) uses a scoring system, testing major system components such as CPU, memory, hard disks, and graphics, to rate performance.

Test 2 Procedure:

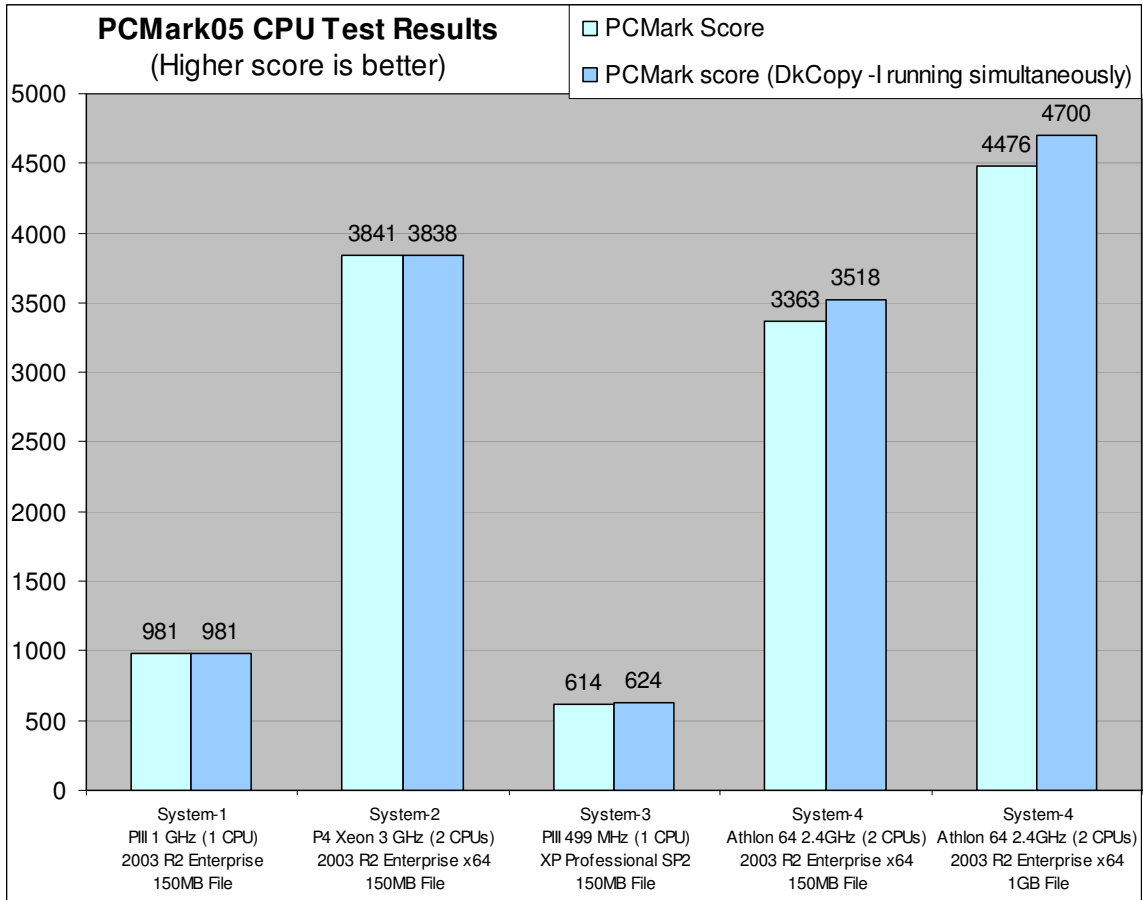
As in Test 1, the system was reset to an identical state in between all test runs to ensure identical starting environments for comparative testing. The same system states created in Test 1 were used for Test 2.

1. PCMark05 was run on each machine to establish a baseline of performance running this tool alone.
2. PCMark05 was run simultaneously with (DkCopy -I) on each machine. This test will measure the impact on system performance (as measured by PCMark) that an InvisiTasking enabled process will have.

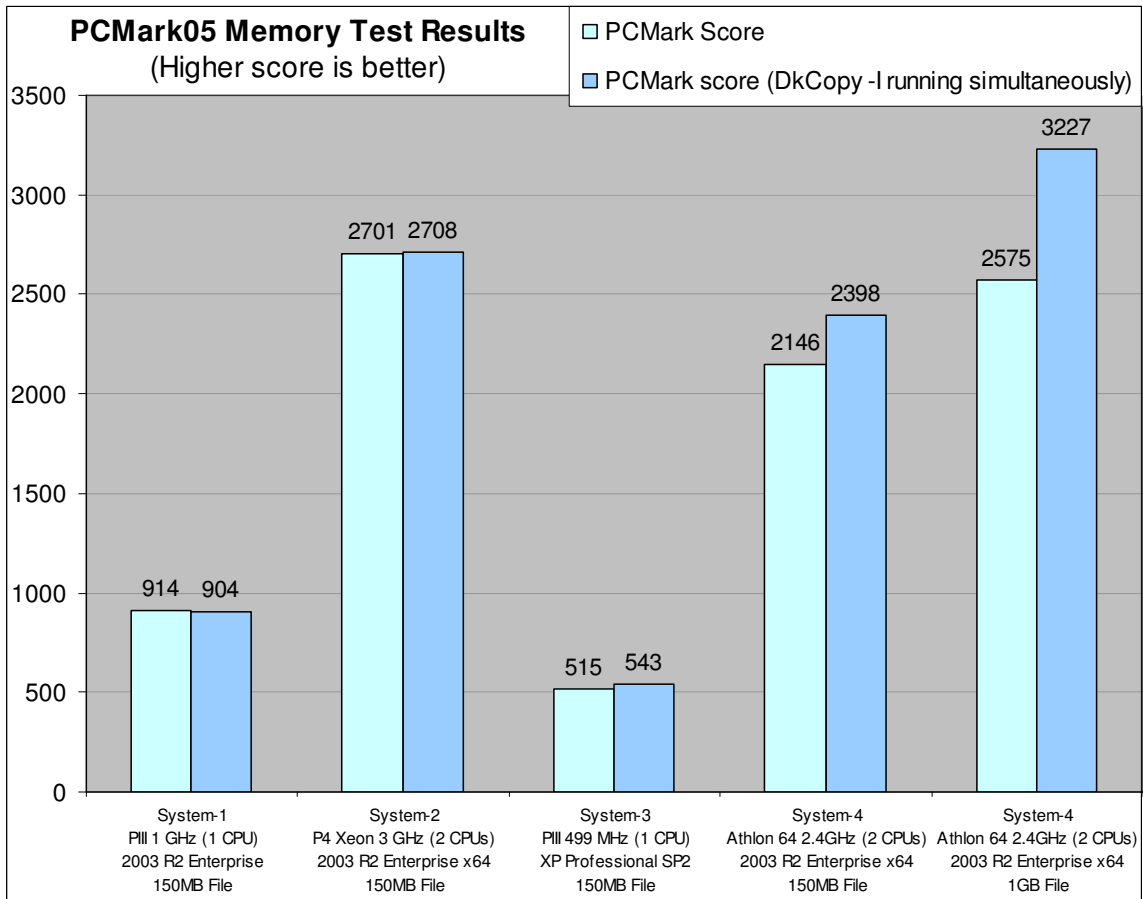
The PCMark05 tests⁷ were run four times on each machine, with the results averaged and represented in the chart below. As the numbers presented are PCMark05 scores, higher scores are better.

⁷ See Appendix A for detailed list of criteria that composes the PCMark score.

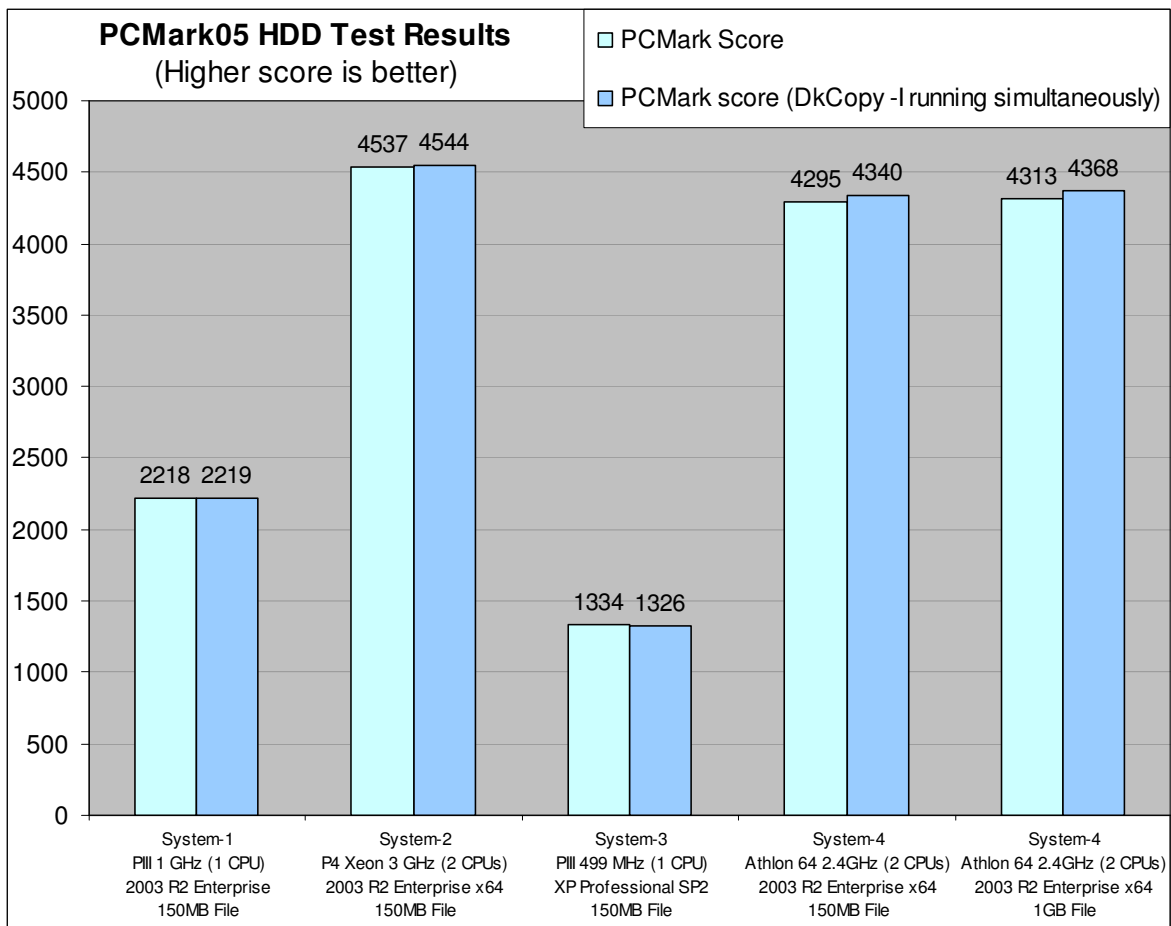
Test 2 Results:



PCMark CPU Performance



PCMark Memory Performance



PCMark Hard Drive Performance

Test 2 Summary

PCMark confirms that earlier execution-time tests hold true when evaluating specific system components. Not only did InvisiTasking *not impede* computer performance, it actually enhanced it in 4 out of 5 cases. Generally the more powerful the system, the more significant the net gain.

INVISITASKING CONCLUSION

The benchmarking tests demonstrate proof-of-concept for the InvisiTasking technology. They show that an InvisiTasking-enabled process does not interfere with other running applications. This was proved in Test 1 where the time to complete for the DkCopy operation was, principally, not impacted.

Test 2 established that system operations are also wholly available to perform a wide variety of typical actions such as CPU processing, memory accesses and modifications, and hard drive access, all while an InvisiTasking-enabled application is fully active.

Diskeeper Corporation also documented performance improvements in *unrelated* services without any corresponding slows in the rest of the system processes. While testing I/O intensive processes with InvisiTasking prototypes, Diskeeper Corporation confirmed an exceptionally significant added benefit affecting processes outside of InvisiTasking's sphere: *InvisiTasking introduces an efficiency into the way resources are allocated by the kernel.*

From a system overhead perspective, InvisiTasking enabled applications can essentially be said to "not exist" – they are truly invisible on a system.

Diskeeper Corporation
7590 N. Glenoaks Blvd.
Burbank, CA 91504
800-829-6468
www.diskeeper.com

© 2006 Diskeeper Corporation. All Rights Reserved. Diskeeper, The Number One Automatic Defragmenter, and InvisiTasking are either registered trademarks or trademarks of Diskeeper Corporation. Microsoft, MS-DOS, Windows, and Windows NT are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries. Lotus Notes is a registered trademark of IBM. PCMark is a registered trademark of Futuremark Corporation. Mac OS is a registered trademark of Apple Computer, Inc. UNIX is a registered trademark of The Open Group. Linux is a Registered Trademark of Linus Torvalds. All other trademarks are the property of their respective owners.

Appendix A
PCMark05 Detailed Test Metrics
(Additional product details at www.futuremark.com)

CPU Test Suite

File Compression
File Decompression
File Encryption
File Decryption
Image Decompression
Audio Compression
Multithreaded Test 1 / File Compression
Multithreaded Test 1 / File Encryption
Multithreaded Test 2 / File Decompression
Multithreaded Test 2 / File Decryption
Multithreaded Test 2 / Audio Decompression
Multithreaded Test 2 / Image Decompression

Memory Test Suite

Memory Read - 16 MB
Memory Read - 8 MB
Memory Read - 192 KB
Memory Read - 4 KB
Memory Write - 16 MB
Memory Write - 8 MB
Memory Write - 192 KB
Memory Write - 4 KB
Memory Copy - 16 MB
Memory Copy - 8 MB
Memory Copy - 192 KB
Memory Copy - 4 KB
Memory Latency - Random 16 MB
Memory Latency - Random 8 MB
Memory Latency - Random 192 KB
Memory Latency - Random 4 KB

HDD Test Suite

HDD - XP Startup
HDD - Application Loading
HDD - General Usage
HDD - Virus Scan
HDD - File Write