

- 1 -

METHOD AND DEVICE FOR MONITORING THE
CREATION AND DESTRUCTION OF CHILD PROCESSES WITHIN
AN APPLICATION EXECUTING IN A COMPUTER SYSTEM

5 Technical Field

The present invention relates generally to tracking process creation during execution of a software application. More particularly, the present invention relates to monitoring the creation and
10 destruction of child processes and grand-child processes of an executing application.

Background

A conventional computer system 100 is illustrated
15 in Fig. 1, and has several hardware components, including a processor 110 such as a central processing unit (CPU), a memory 120 such as RAM, and a non-volatile storage medium 130, all in communication with one another. Other hardware components not shown
20 include disk drives, network devices, and other hardware devices known to those skilled in the art.

Conventional computer system 100 further includes software programs which may be stored on and executed using the several hardware components 110, 120, 130.
25 Specifically, the software in computer system 100 includes application 150, which resides in storage medium 130, operating system 160, stored in memory 120, and possibly other applications or programs stored in storage medium 130 and memory 120. When the computer

- 2 -

is active, application 150 may be found in memory 120. When the computer is off, operating system 160 may be found in storage medium 130. Thus, operating system 160 may be stored in various locations, as should be
5 understood by those skilled in the art.

Application 150 and operating system 160 interact with one another and possibly other software in system 100 to provide data access to a user and to control other hardware and software in system 100 as designated
10 by the user. The user of computer system 100 executes application 150 by instructing operating system 160 to execute, or "run," application 150. Operating system 160 then executes the requested application 150, and possibly other programs interleaved with application
15 150, using memory 120, processor 110, and other devices available to computer system 100.

Operating system 160 may be any conventional operating system known to those skilled in the art. Popular conventional operating systems used in personal
20 computers include 32-bit Windows ("Win32") operating systems such as those made by Microsoft Corp. These include Microsoft Windows NT, Windows 95 and Windows 98. Other conventional operating systems which may be used as operating system 160 include UNIX for use with
25 workstations, and DOS for use with personal computers.

A Win32-based application running in the system is called a "process" in terms of the operating system. Each process consists of at least a single thread of execution to be run by the operating system. A thread

- 3 -

is a unit of code that can get some operating time from the operating system to run concurrently with other units of code, and must be associated with a process.

Win32-based applications can take advantage of various techniques provided by Windows operating systems to facilitate the ability to run applications concurrently. One of these concurrency techniques is "multi-threading," in which portions of one or more processes, or "threads", are separately and concurrently executable.

Executing two separate threads concurrently, however, is sometimes not particularly efficient. These inefficiencies arise where one executing application or process, the "parent" process, spawns another "child" process, and particularly where the child process spawns grandchild processes. Such situations may present problems where the parent process is one of a series of processes or applications to be executed. For example, the parent process may terminate before one or more of its child processes or grandchild processes have terminated. If execution of some other application commences while the child processes of the terminated parent processes are still active, corruption may result in some computer systems, notwithstanding the multi-threading techniques provided by operating systems such as Windows 95.

Fig. 2 illustrates a series of applications stored for execution in a computer system, such as system 100 of Fig. 1. The series includes a first application

- 4 -

210, a second application 220, and a third application 230, each of which represents a stage of a larger installation process for installing a software program. Applications 210, 220, 230 may be stored on a storage
5 medium 130 or memory 120, as shown in Fig. 1. The execution of applications 210, 220, 230 is governed by an operating system 240, which is stored in memory 120.

In Fig. 2, applications 210, 220, 230 are to be executed in sequence, as controlled by operating system
10 240. Fig. 2 is drawn at a time when execution of first application 210 has begun, and at a time when first application 210 has created child processes 250, 260. In addition, child process 260 has spawned child processes of its own, or "grand-child" processes 270,
15 280.

In Fig. 2, the execution of first application 210 may terminate before child processes 250, 260, or grand-child processes 270, 280 have terminated. If the installation process does not wait for the executing
20 child or grand-child processes to terminate, operating system 240 may initiate execution of second application 220, the next stage in installation. Execution of second application 220, or possibly some other application, while one or more child or grand-child
25 processes of the first application 210 are still executing could cause corruption in the computer system in which the applications are used.

In Windows 95, the published Win32 API is supposed to delay the execution of other applications before

- 5 -

child and processes of a parent application are terminated. Win32 API, however, often does not work correctly because Win32 API does not track the creation of grand-child processes of the parent application. Thus, using only Win32 API, the execution of other applications may commence before the child and, particularly, the grand-child processes of the parent have terminated. In sum, existing techniques fail to provide reliable tracking of the creation and destruction of child processes and grand-child processes during the execution of applications in a computer system.

Summary

The present invention relates generally to monitoring the creation and destruction of child and grand-child processes within an application executing in a computer system controlled by an operating system. A child process has a corresponding child process database structure, and the executing application has a corresponding application process database structure. A list is provided for storing existing child process database structures corresponding to child processes existing within the executing application.

According to one aspect of the present invention, when a new process is created during execution of the application, a new process database structure corresponding to the new process is retrieved. Using the new process database structure, a parent process

- 6 -

database structure corresponding to a parent process of the new process is retrieved. The parent process database structure is compared with the application process database structure. If the parent process database structure is the same as the application process database structure, the new process is identified as an existing child process. If the new process is identified as the existing child process, the new process database structure is added to the list of existing child process database structures.

According to another aspect of the present invention, monitoring the creation and destruction of child processes within the application includes determining when a process is terminated during execution of the application. A terminated process database structure corresponding to the terminated process is retrieved. The terminated process database structure is compared with the existing child process database structures in the list. If the terminated process database structure is the same as one of the existing child process database structures, the terminated process is identified as a terminated child process. If the terminated process is identified as the terminated child process, the terminated process database structure is removed from the list of existing child process database structures.

According to another aspect, the present invention may be situated in a virtual device driver, which may reside between the operating system and an executing

- 7 -

application or process. The virtual device driver may be stored in a storage medium in the computer system. The VxD interfaces with both the operating system and the executing application.

5 According to yet another aspect of the present invention, the list is checked periodically to determine if the list has any entries. If the list has no entries, it is safe to assume that the existing child process and, possibly, related grand-child
10 processes, have exited and execution of the application is complete. Other applications in a series of applications, for example, during the installation of a software program, may then be executed with minimal risk of corruption due to concurrent execution of
15 applications in the series.

 These and other aspects and advantages of the present invention will appear in the following detailed description in which exemplary embodiments of the invention have been set forth in conjunction with the
20 drawings.

Brief Description

Fig. 1 is a generalized block diagram of a conventional computer system 100;

25 Fig. 2 is a generalized block diagram illustrating the execution of a series of software applications within conventional computer system 100;

Fig. 3 is a generalized block diagram of a computer system 300 incorporating a device and method

- 8 -

for monitoring the creation and destruction of child processes within an executing application, according to an exemplary embodiment of the present invention;

5 Fig. 4 is a generalized block diagram of a virtual device driver which monitors the creation of child processes within an executing application, according to an exemplary embodiment of the present invention;

10 Fig. 5 is a flow diagram of a method for monitoring the creation of child processes within an executing application, in accordance with an exemplary embodiment of the present invention;

15 Fig. 6 is a generalized block diagram of a virtual device driver which monitors the destruction of child processes within an executing application, according to an exemplary embodiment of the present invention; and

20 Fig. 7 is a flow diagram of a method for monitoring the destruction of child processes within an executing application, in accordance with an exemplary embodiment of the present invention.

Detailed Description

25 In the following detailed description, various embodiments of the device and method of the present invention for monitoring the creation and destruction of child processes within an executing application will be provided. Some embodiments of the present invention will identify practical applications, particularly of exemplary methods of the present invention, including a computer system and a computer-readable medium.

- 9 -

Embodiments of the present invention are described in detail with reference to Figs. 3-7.

Fig. 3 is a block diagram of a computer system 300 incorporating a device and method for monitoring the creation and destruction of child processes within an executing application, according to an exemplary embodiment of the present invention. Computer system 300 of Fig. 3 includes a processor 330 and memory 325. Processor 330 may contain a single microprocessor, or may contain a plurality of microprocessors for configuring the computer system as a multi-processor system. Memory 325, stores, in part, instructions and data for execution by processor 330. If exemplary systems of the present invention are wholly or partially implemented in software, including a computer program, memory 325 stores the executable code when in operation. Memory 325 may include banks of dynamic random access memory (DRAM) as well as high speed cache memory.

The system of Fig. 3 further includes a mass storage device 335, peripheral device(s) 340, input device(s) 355, portable storage medium drive(s) 360, a graphics subsystem 370 and a display 385. For purposes of simplicity, the components shown in Fig. 3 are depicted as being connected via a single bus 380. However, the components may be connected through one or more data transport means. For example, processor 330 and memory 325 may be connected via a local microprocessor bus, and the mass storage device 335,

- 10 -

peripheral device(s) 340, portable storage medium drive(s) 360, and graphics subsystem 370 may be connected via one or more input/output (I/O) buses.

5 Mass storage device 335, which is typically implemented with a magnetic disk drive or an optical disk drive, is a non-volatile storage device for storing data and instructions for use by processor 330. In another embodiment, mass storage device 335 stores a computer program which, when executed, implements an
10 exemplary method of the present invention for purposes of loading such program to memory 325. An exemplary method of the present invention also may be stored in processor 330.

15 Portable storage medium drive 360 operates in conjunction with a portable non-volatile storage medium, such as a floppy disk, hard disk, or CD-ROM, and may include one or more machine-readable media including semiconductor memories (e.g., EEPROM), or other computer-readable medium, to input and output
20 data and code to and from the computer system of Fig. 3. In one embodiment, an exemplary method of the present invention is stored on such a portable medium, and is input to the computer system 300 via the portable storage medium drive 360.

25 Peripheral device(s) 340 may include any type of computer support device, such as an input/output (I/O) interface, to add additional functionality to the computer system 300. For example, peripheral device(s) 340 may include a network interface card for

- 11 -

interfacing computer system 300 to a network, a modem, and the like.

Input device(s) 355 provide a portion of a user interface. Input device(s) 355 may include an alpha-
5 numeric keypad for inputting alpha-numeric and other key information, or a pointing device, such as a mouse, a trackball, stylus or cursor direction keys. Such devices provide additional means for interfacing with methods according to exemplary embodiments of the
10 present invention.

In order to display textual and graphical information, the computer system 300 of Fig. 3 includes graphics subsystem 370 and display 385. Display 385 may include a cathode ray tube (CRT) display, liquid
15 crystal display (LCD), other suitable display devices, or other means for displaying that enables a user to view the status of child processes as they are created and destroyed within an executing application. Graphics subsystem 370 receives textual and graphical
20 information and processes the information for output to display 385. Display 385 can be used to display objects, component interfaces and/or display other information that is part of a user interface.

The system 300 of Fig. 3 also includes an audio
25 system 350. In one embodiment, audio system 350 includes a sound card that receives audio signals from a microphone that may be found in peripherals 340.

Additionally, the system of Fig. 3 includes output devices 345. Examples of suitable output devices

- 12 -

include speakers, printers, and the like.

The devices contained in the computer system of Fig. 3 are those typically found in general purpose computer systems, and are intended to represent a broad category of such computer components that are well known in the art. The system of Fig. 3 illustrates one platform which can be used for practically implementing methods according to exemplary embodiments of the present invention. Numerous other platforms will also suffice, such as Macintosh-based platforms available from Apple Computer, Inc., platforms with different bus configurations, networked platforms, multi-processor platforms, other personal computers, workstations, mainframes, navigation systems, and the like.

Computer system 300 may further include a different display means for the monitor, such as CRT display, LCD display, projection displays, or the like. Likewise, any type of memory may be used for memory 325. Other interface means, in addition to the component interfaces, may also be used including alphanumeric keypads, other key information or any pointing devices such as a mouse, trackball, stylus, cursor or direction key.

The computer system 300 of Fig. 3 further includes a device driver 390, in which an exemplary virtual device driver ("VxD") constructed according to the present invention may be stored. Some operating systems, such as Windows 95, provide for the use of virtual device drivers, which are often used to improve

- 13 -

software performance. A virtual device driver is typically a 32-bit, protected-mode driver that manages a system resource, such as a hardware device or installed software, so that more than one application
5 can use the resource at the same time. The "x" in "VxD" represents the type of device driver. For example, a virtual device driver for a display device is known as a VDD, a virtual device driver for a timer device is a VTD, a virtual device driver for a printer
10 device is a VPD, and so forth.

VxDs support hardware devices for a typical computer, including disk controllers, serial and parallel ports, keyboard and display devices, and so on. If the state of the hardware device can be
15 disrupted by switching between multiple applications, the device generally has a corresponding virtual device and VxD. The VxD typically keeps track of the state of the device for each application and ensures that the device is in the correct state whenever an application
20 continues.

Although most virtual devices manage hardware, some manage only installed software, such as an MS-DOS device driver. Such virtual devices contain code to emulate the software or ensure that the software uses
25 data that applies only to the currently running application.

In Fig. 3, the VxD may be stored in memory 325, mass storage 335, or portable storage medium 360. The virtual device driver represents one embodiment of an

- 14 -

exemplary method in accordance with the present invention, explained in greater detail below, and may be used with an operating system such as Windows 95 to monitor the creation and destruction of child processes during execution of an application.

Fig. 4 illustrates an exemplary virtual device driver 400 in accordance with the present invention. VxD 400 is configured to determine whether a new process 425 is an existing child process 465 of executing application 410. Thus, VxD 400 is in communication with executing application 410 and operating system 405, which is stored in mass storage device 335 or memory 325 of computer system 300 of Fig. 3. Operating system 405 controls the execution of application 410.

In Fig. 4, executing application 410 is shown as the parent process of existing child processes 445 and 450. Executing application 410 may be controlled directly by operating system 405, or it may be the child or grand-child process created by one or more parent processes executing under the control of operating system 405. Thus, for example, if executing application 410 is the child process of another parent process (not shown), existing child processes 445 and 450 would be grand-child processes of that parent process. In this way, VxD 400 is configured to monitor the creation and destruction of both child processes and grand-child processes.

In Fig. 4, application 410 has a corresponding

- 15 -

application database structure ("APDB") 412 identifying the particular application 410. APDB 412 includes process identifiers which identify, respectively, application 410, any child processes of application 410, and any grand-child processes of application 410. For example, the following code represents an exemplary APDB used in a Windows 95 or Windows NT operating system:

```
10         typedef struct PDB
           DWORD Type;
           DWORD cReference;
           DWORD un1;
           DWORD pSomeEvent;
           DWORD TerminationStatus;
15         DWORD un2;
           DWORD DefaultHeap;
           DWORD MemoryContext;
           DWORD flags;
           DWORD pPSP;
20         WORD PSPSelector;
           WORD MTEIndex;
           WORD cThreads;
           WORD cNotTermThreads;
           WORD un3;
25         WORD cRing0Threads;
           DWORD HeapHandle;
           DWORD W16TDB;
           DWORD MemMapFiles;
           PENVIRONMENT _DB pEDB;
30         DWORD pHandleTable;
           struct PDB *ParentPDB;
           DWORD MODREFlist;
           DWORD ThreadList;
           DWORD DebuggeeCB;
35         DWORD LocalHeapFreeHead;
           DWORD InitialRingOID;
           DWORD crst[4];
           DWORD un4[3];
           DWORD pConsole;
40         DWORD tlsInUseBits1;
```

- 16 -

```
        DWORD tlsInUseBits2;
        DWORD ProcessDWORD;
        struct PDB *ProcessGrou
5         DWORD pExeMODREF;
        DWORD TopExcFilter;
        DWORD BasePriority;
        DWORD HeapOwnList;
        DWORD HeapHandleBlockLis
10        DWORD pSomeHeapPtr;
        DWORD pConsoleProvider;
        WORD EnvironSelector;
        WORD ErrorMode;
        DWORD pevtLoadFinished;
        WORD UTState;
15        }PDB, *PPDB;
```

In Fig. 4, operating system 405 includes a system process creation function 415 which controls the creation of new processes in the computer system 300. System process creation function 415 creates new process 425, having a corresponding new process database structure ("NPDB") 430, during execution of application 410. New process database structure 430 may include process identifiers which identify, respectively new process 425 and the parent process of new process 425.

In Fig. 4, upon creating new process 425, system process creation function 415 outputs a signal on line 435. In one example, where operating system 405 is Windows 95, this signal is a Create_Thread signal or Thread_Init signal.

In Fig. 4, operating system 405 further includes a get current process function 420. In the example where operating system 405 is Windows 95, function 420

- 17 -

is the VWIN32_GetCurrentProcessHandle function provided by the Windows 95 operating system. When called by calling part 475, get current process function 420 relays new process database structure 430 to VxD 400 over line 440.

In Fig. 4, application 410 has begun executing, and has spawned child processes, including child process 445 and child process 450. These processes 445 and 450 are considered "existing," as they have not yet terminated or otherwise been identified as terminated child processes of executing application 410. Existing child processes 445 and 450 have corresponding existing child process databases ("EPDBs"), respectively, illustrated in Fig. 4 as EPDB 0 and EPDB 1. Each existing child process database structure preferably includes process identifiers which identify itself, its parent process, and grand-parent processes or applications, if any.

In Fig. 4, a list 455 is provided for storing existing child process database structures corresponding to child processes existing within the executing application. As shown in Fig. 4, EPDB 0 and EPDB 1, corresponding to existing child processes 445 and 450, have already been added to list 455 in sequence, using methods described in greater detail below. List 455 may be realized in software or hardware, depending on the desired implementation. For example, in Fig. 3, list 455 may be implemented using a buffer in computer system 300, and maintained in

- 18 -

memory 325 or mass storage 335. List 455 is shown in Fig. 4 as having at least 4 entries (entries 0 through 3), and may have any number of entries as desired for the particular application of the program.

5 In Fig. 4, a pointer 460 is provided to identify one of the existing child process database structures in list 455, preferably the EPDB most recently added to the list. In Fig. 4, EPDB 1 was added to list 455 after EPDB 0, so pointer 460 points to EPDB 1. Pointer
10 460 is preferably embodied in a counter, which may be implemented in hardware or software of computer system 300. Pointer 460 is moveable to point to any one of various entries in list 455, as described in greater detail below.

15 In Fig. 4, VxD 400 is configured to monitor the creation of new process 425 and determine if new process 425 may be classified in block 465 as an existing child process of executing application 410. To this end, VxD 400 includes parts which perform the
20 specific operations set forth in flow diagram 500 of Fig. 5.

 In Figs. 4 and 5, execution of application 410 begins in step 510. VxD 400 includes a determining part 470, which determines when new process 425 is
25 created, in step 520. As shown in Fig. 4, determining part 470 communicates with system process creation function 415 to receive signals over line 435 that indicate that new process 425 has been created. Thus, in the example where operating system 405 is Windows

- 19 -

95, determining part 470 of VxD 400 receives the Create_Thread signal or Thread_Init signal over line 435 when new process 425 is created. Other various messages and signals may be passed to determining part 470 to notify VxD 400 of a thread's creation, as should be appreciated by those skilled in the art.

In Fig. 5, after determining in step 520 that new process 425 has been created, in step 530, VxD 400 calls "get current process" function 420 to get new process database structure 430 corresponding to new process 425, over line 440. As shown in Fig. 4, operation 530 is carried out by calling part 475, which communicates with get current process function 420 over line 440. In the example where operating system 405 is Windows 95, calling part 475 makes a call to the function VWIN32_GetCurrentProcessHandle to get the current process handle or database structure corresponding to new process 425, that is, NPDB 430.

In Figs. 4 and 5, NPDB 430 is passed to calling part 475 in step 530, when get current process function 420 is called. Within NPDB 430, a parent process identifier is included which identifies the parent process of new process 425. In this way, VxD 400 can use the parent process identifier to retrieve the parent process database structure (PPDB) of the parent process of new process 425. For example, in one embodiment, a pointer is passed with NPDB 430 which may then be used by VxD 400 to retrieve the PPDB from operating system 405.

- 20 -

As shown in Figs. 4 and 5, virtual device driver 400 further includes a comparing part 480, which communicates with calling part 475 and executing application 410. Comparing part 480 receives the PPDB from calling part 475, and then compares the PPDB with the APDB 412 corresponding to application 410, in step 540. In this way, VxD 400 determines if thread 425 is a child process of executing application 410. If more than one application is being executed, the PPDB is compared with the application database structures of the various applications.

In step 550 of Fig. 5, if comparing part 480 determines that there is no identity between the PPDB corresponding to the parent of new process 425 and APDB 412, the VxD recognizes that new process 425 is not a child process of executing application 410, and VxD disregards new process 425. The flow then returns to step 520. On the other hand, if the PPDB matches APDB 412, in step 550, the flow progresses to step 560. In step 560, an identifier part 485 of VxD 400 identifies new process 425 as an existing child process in block 465. This positive identification is then communicated to an adder part 490 of VxD 400.

In Figs. 4 and 5, following step 560, in step 570, adder part 490 retrieves NPDB 430 from determining part 470 and adds NPDB 430 to the list 455 of existing child process database structures, in entry "2." After adding NPDB 430 to list 455, pointer 460 is moved to point to NPDB 430 in entry "2." In the example where

- 21 -

pointer 460 is maintained using a counter, the counter is incremented by "1." The method of Fig. 5 then returns, from step 570, to step 520 to determine when the next new process is created during execution of application 410.

Exemplary virtual device drivers according to embodiments of the present invention are also capable of monitoring the destruction of threads within an executing application, as shown in Figs. 6 and 7. An exemplary virtual device driver 600 has parts similar to those of VxD 400, and also includes parts configured to monitor the destruction of child processes and grand-child processes within executing application 410, described in greater detail below.

In Fig. 6, an operating system 605, stored in mass storage device 335 or memory 325 of computer system 300 of Fig. 3, shares the features of operating system 405 in Fig. 4. For example, operating system 605 similarly controls the execution of application 410, and includes get current process function 420. In addition, operating system 605 includes a system process termination function 610 which controls the termination of processes while application 410 executes in computer system 300. In Fig. 6, system process termination function 610 terminates a process 615, having a corresponding terminated process database structure ("TPDB") 620, during execution of application 410. TPDB 620 preferably includes process identifiers similar to NPDB 430 of Fig. 4.

- 22 -

Upon the termination of process 615, system process termination function 610 outputs a signal on line 625. In one example, where operating system 605 is Windows 95, this signal is a `Terminate_Thread` signal. Using Windows 95, other message signals may be output to line 625 by system process termination function 610 to indicate the destruction of a thread, including `Thread_Not_Executable` and `Destroy_Thread` message signals. Various other similar message signals may be output to line 625 to notify VxD 600 of a thread's destruction, as should be appreciated by those skilled in the art.

In Fig. 6, application 410 has begun executing, and has spawned existing child processes 630, 635, and 640. Existing child processes 630-640 each have corresponding EPDBs, namely EPDB 0, EPDB 1, and EPDB2, as shown in Fig. 6. EPDBs 0-2 have been added to list 455 in sequence, using methods described above with respect to Figs. 4 and 5.

In Fig. 6, VxD 600 is not only configured to monitor the creation of child processes during execution of application 410, similar to VxD 400 of Fig. 4, but also configured to monitor the termination of process 615 and determine if terminated process 615 is any of one of existing child processes 630-640. To perform this monitoring, VxD 400 includes parts which perform the specific operations set forth in flow diagram 700 of Fig. 7.

In Figs. 6 and 7, execution of application 410

- 23 -

begins in step 710. VxD 600 includes a determining part 645 which determines when process 615 is terminated, in step 720. Determining part 645 communicates with system process termination function 5 610 to receive signals over line 625 indicating that process 615 has been terminated. In the example where operating system 405 is Windows 95, determining part 645 monitors line 625 for the Terminate_Thread message signal, or the Thread_Not_Executable or Destroy_Thread 10 message signals. By monitoring line 625, for any of these signals, VxD 600 is notified when a thread, such as process 615, is terminated.

In Fig. 7, after determining in step 720 that process 615 has been terminated, in step 730, VxD 600 15 calls "get current process" function 420 to get terminated process database structure 620 corresponding to terminated process 615, over a line 650, using calling part 655. In the example where operating system 605 is Windows 95, calling part 655 makes a call 20 to the function VWIN32_GetCurrentProcessHandle to get TPDB 620 as the current process handle or database structure corresponding to terminated process 615.

As shown in Figs. 6 and 7, virtual device driver 600 further includes a comparing part 660 which 25 communicates with calling part 655 to receive TPDB 620, and then compares TPDB 620 with the existing child process database structures EPDB 0-2 stored in list 455, in step 740.

In step 750 of Figs. 6 and 7, if comparing part

- 24 -

660 determines there is no match between TPDB 620 and any of the existing process database structures EPDB 0-2 stored in list 455, VxD 600 recognizes that terminated process 615 is not one of existing child processes 630-640, and the flow returns to step 720. In step 760, if comparing part 660 determines that TPDB 620 is the same as any of EPDBs 0-2 stored in list 455, an identifier part 670 identifies terminated process 615 as a terminated child process. For example, in Fig. 6, comparing part 660 determines that TPDB 620 is the same as EPDB 2 stored in list 455, so identifier part 670 identifies terminated process 615 as a terminated child process, namely child process 640.

In Figs. 6 and 7, the flow progresses from step 760 to step 770, in which a removing part 675 situated within VxD 600 removes TPDB 620 from list 455 of existing child process database structures EPDB 0-2. Thus, in Fig. 6, since EPDB 2 matches TPDB 620, EPDB 2 is removed from list 455. In other examples, where one or more other EPDBs stored in list 455 match TPDB 620, such EPDBs are also removed from list 455. After EPDBs are removed from list 455, pointer 460 is moved to point a remaining existing child process database structure most recently added to list 455. Thus, in Fig. 2, after EPDB2 is removed from list 455, pointer 460 is moved back by one entry to point to entry "1," EPDB 1. In another example, where EPDB0 is removed from the list, EPDB1 is moved to entry "0," EPDB2 is moved to entry "1," and pointer 460 is moved back by

- 25 -

one entry to point to EPDB2 in entry "1." If pointer 460 is realized using a counter, the counter is decremented by "1" or, in other examples, by the number of entries removed from list 455. The method of Fig. 5 7 then returns, from step 770, to step 720 to determine when another process is destroyed during execution of application 410.

According to exemplary embodiments of the present invention, if a child process is spawned by an 10 executing application, the child process database structure corresponding to the child process is added to the list 455 of monitored processes. When the child process is terminated, the child process database structure is removed from list 455.

15 Thus, exemplary embodiments of the present invention also provide for controlling the execution of a series of applications or processes in a computer system. For example, the installation process of a software program often includes the execution of a 20 series of installation applications contained therein. To avoid corruption in the computer system, it is often desirable to execute these installation applications in sequence, such that no two installation applications are executing concurrently.

25 When one of the applications in a series is executing, an exemplary VxD, such as VxD 400 or 600, periodically determines whether list 455 has any entries. In this way, if list 455 contains no existing child process database structures, the VxD recognizes

- 26 -

that the existing child process and, possibly, related grandchild processes, have exited and the execution of the application is complete. Thus, the VxD may then signal the operating system to execute the next application in the sequence, for example, the next step in the installation process.

In another exemplary embodiment, the operating system or program that started the execution of the application being monitored calls the VxD to determine if the list is empty. In the example where list 455 is embodied in a counter, the VxD determines whether the counter equals zero. The operating system may call the VxD in this fashion periodically, for example, every 5 seconds or so. If the counter does not equal zero, then the periodic calling desirably continues. Otherwise, it is safe to assume that the monitored application and the children thereof have terminated, and the next program in the series may be executed or installed.

It should be understood that the particular embodiments described above are only illustrative of the principles of the present invention, and various modifications could be made by those skilled in the art without departing from the scope and spirit of the invention. Thus, the scope of the present invention is limited only to the extent of the claims that follow.

- 27 -

CLAIMS

What is claimed is:

- 1 1. A method for monitoring the creation and
- 2 destruction of child processes within an application
- 3 executing in a computer system controlled by an operating
- 4 system, each child process having a corresponding child
- 5 process database structure, the executing application
- 6 having a corresponding application process database
- 7 structure, the method comprising:
- 8 providing a list for storing existing child
- 9 process database structures corresponding to child
- 10 processes existing within the executing application;
- 11 determining when a new process is created during
- 12 execution of the application;
- 13 retrieving a new process database structure
- 14 corresponding to the new process;
- 15 retrieving, using the new process database
- 16 structure, a parent process database structure
- 17 corresponding to a parent process of the new process;
- 18 comparing the parent process database structure
- 19 with the application process database structure;
- 20 identifying, if the parent process database
- 21 structure is the same as the application process database
- 22 structure, the new process as an existing child process;
- 23 and
- 24 adding, if the new process is identified as the
- 25 existing child process, the new process database structure



- 28 -

26 to the list of existing child process database structures.

1 2. The method of claim 1 further comprising:
2 determining when a process is terminated during
3 execution of the application;
4 retrieving a terminated process database
5 structure corresponding to the terminated process;
6 comparing the terminated process database
7 structure with the existing child process database
8 structures in the list;
9 identifying, if the terminated process database
10 structure is the same as one of the existing child process
11 database structures, the terminated process as a
12 terminated child process; and
13 removing, if the terminated process is
14 identified as the terminated child process, the terminated
15 process database structure from the list of existing child
16 process database structures.

1 3. The method of claim 1 wherein the list of
2 existing child process database structures is maintained
3 in a buffer in the computer system, the method further
4 comprising:
5 providing a pointer which points to one of the
6 existing child process database structures in the list;
7 and
8 after adding the new process database structure
9 to the list, moving the pointer to point to the new
10 process database structure.



- 29 -

1 4. The method of claim 2 wherein the list of
2 existing child process database structures is maintained
3 in a buffer in the computer system, the method further
4 comprising:
5 providing a pointer which points to one of the
6 existing child process database structures in the list;
7 after adding the new process database structure
8 to the list, moving the pointer to point to the new
9 process database structure; and
10 after removing the terminated process database
11 structure from the list, moving the pointer to point to
12 one of the existing child process database structures most
13 recently added to the list.

1 5. The method of claim 4 wherein the pointer is
2 maintained in a counter in the computer system, and
3 wherein:
4 moving the pointer to point to the new process
5 database structure includes incrementing the counter; and
6 moving the pointer to point to the existing child
7 process most recently added to the list includes
8 decrementing the counter.

1 6. The method of claim 2 wherein the executing
2 application is one of a series of applications stored
3 within the computer system for execution, the method
4 further comprising:
5 periodically determining whether the list



- 30 -

6 includes any existing child process database structures;
7 and
8 executing, if the list includes no existing
9 child process database structures, a next application in
10 the series of applications.

1 7. The method of claim 5 wherein the executing
2 application is one of a series of applications stored
3 within the computer system for execution, the method
4 further comprising:

5 periodically determining whether the counter has
6 a value of zero; and

7 executing, if the counter has a value of zero,
8 a next application in the series of applications.

1 8. The method of claim 6 wherein the series of
2 applications is at least a portion of an installation
3 program.

1 9. The method of claim 1 wherein the operating
2 system is a Windows operating system having a system
3 process creation function for creating the new process and
4 thereupon providing a Create_Thread signal, the operating
5 system further having a Get_Current_Process_Handle
6 function for providing the new process database
7 structure.

1 10. The method of claim 9 wherein determining when
2 the new process is created includes receiving the



- 31 -

3 Create_Thread signal.

1 11. The method of claim 9 wherein retrieving the new
2 process database structure includes calling the
3 Get_Current_Process_Handle function.

1 12. The method of claim 9 wherein retrieving the
2 parent process database structure includes using a pointer
3 in the new process database structure which identifies the
4 parent process database structure.

1 13. The method of claim 2 wherein the operating
2 system is a Windows operating system having a system
3 process creation function for creating the new process and
4 thereupon providing a Create_Thread signal, and a system
5 process termination function for terminating the process
6 and thereupon providing a Terminate_Thread signal, and a
7 Get_Current_Process_Handle function for providing the
8 terminated process database structure.

1 14. The method of claim 13 wherein determining when
2 the process is terminated includes receiving the
3 Terminate_Thread signal.

1 15. The method of claim 13 wherein retrieving the
2 terminated process database structure includes calling the
3 Get_Current_Process_Handle function.

1 16. A method for monitoring the creation and



- 32 -

2 destruction of child processes within an application
3 executing in a computer system controlled by a Windows
4 operating system, each child process having a
5 corresponding child process database structure, the
6 executing application having a corresponding application
7 process database structure, the Windows operating system
8 having a system process creation function for creating a
9 new process and thereupon providing a Create_Thread
10 signal, the operating system further having a
11 Get_Current_Process_Handle function for providing, when
12 called a new process database structure corresponding to
13 the new process, the method comprising:

14 providing a list for storing existing child
15 process database structures corresponding to child
16 processes existing within the executing application;

17 determining, upon receiving the Create_Thread
18 signal, that the new process has been created;

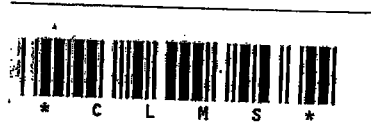
19 calling, after determining that the new process
20 has been created, the Get_Current_Process_Handle function
21 to retrieve the new process database structure;

22 identifying, using a pointer in the new process
23 database structure, a parent process database structure
24 corresponding to a parent process of the new process;

25 comparing the parent process database structure
26 with the application process database structure;

27 identifying, if the parent process database
28 structure is the same as the application process database
29 structure, the new process as an existing child process;

30 and



- 33 -

31 adding, if the new process is identified as the
32 existing child process, the new process database structure
33 to the list of existing child process database structures.

1 17. The method of claim 16, the operating system
2 further having a system process termination function for
3 terminating a process and thereupon providing a
4 Terminate_Thread signal, wherein the
5 Get_Current_Process_Handle function further provides, when
6 called, a terminated process database structure
7 corresponding to the terminated process, the method
8 further comprising:

9 determining, upon receiving the Terminate_Thread
10 signal, that the process has been terminated;

11 calling, after determining that the process has
12 been terminated, the Get_Current_Process_Handle function
13 to retrieve the terminated process database structure;

14 comparing the terminated process database
15 structure with the existing child process database
16 structures in the list;

17 identifying, if the terminated process database
18 structure is the same as one of the existing child process
19 database structures, the terminated process as a
20 terminated child process; and

21 removing, if the terminated process is
22 identified as the terminated child process, the terminated
23 process database structure from the list of existing child
24 process database structures.



- 34 -

1 18. A virtual device driver situated in a storage
2 medium of a computer system for monitoring the creation
3 and destruction of child processes within an application
4 executing in the computer system controlled by an
5 operating system, each child process having a
6 corresponding child process database structure, the
7 executing application having a corresponding application
8 process database structure, a list provided for storing
9 existing child process database structures corresponding
10 to child processes existing within the executing
11 application, the virtual device driver comprising:
12 a determining part which determines when a new
13 process is created during execution of the application;
14 a calling part which:
15 (i) retrieves a new process database
16 structure corresponding to the new process, and
17 (ii) retrieves, using the new process
18 database structure, a parent process database structure
19 corresponding to a parent process of the new process;
20 a comparing part which compares the parent
21 process database structure with the application process
22 database structure;
23 an identifier part which identifies, if the
24 parent process database structure is the same as the
25 application process database structure, the new process as
26 an existing child process; and
27 an adding part which adds, if the new process is
28 identified as the existing child process, the new process
29 database structure to the list of existing child process



- 35 -

30 database structures.

1 19. The virtual device driver of claim 18 further
2 comprising:

3 a determining part which determines when a
4 process is terminated during execution of the application;
5 a calling part which retrieves a terminated
6 process database structure corresponding to the terminated
7 process;

8 a comparing part which compares the terminated
9 process database structure with the existing child process
10 database structures in the list;

11 an identifier part which identifies, if the
12 terminated process database structure is the same as one
13 of the existing child process database structures, the
14 terminated process as a terminated child process; and

15 a removing part which removes, if the terminated
16 process is identified as the terminated child process, the
17 terminated process database structure from the list of
18 existing child process database structures.

1 20. The virtual device driver of claim 18 wherein
2 the operating system is a Windows operating system having
3 a system process creation function for creating the new
4 process and thereupon providing a Create_Thread signal,
5 the operating system further having a
6 Get_Current_Process_Handle function for providing the new
7 process database structure.



- 36 -

1 21. The virtual device driver of claim 20 wherein
2 the determining part is in communication with the system
3 process creation function and determines when the new
4 process is created by receiving the Create_Thread signal.

1 22. The virtual device driver of claim 20 wherein
2 the calling part is in communication with the
3 Get_Current_Process_Handle function and retrieves the new
4 process database structure by calling the
5 Get_Current_Process_Handle function.

1 23. The virtual device driver of claim 19 wherein
2 the operating system is a Windows operating system having
3 a system process creation function for creating the new
4 process and thereupon providing a Create_Thread signal,
5 and a system process termination function for terminating
6 the process and thereupon providing a Terminate_Thread
7 signal, and a Get_Current_Process_Handle function for
8 providing the terminated process database structure.

1 24. The virtual device driver of claim 23 wherein
2 the determining part is in communication with the system
3 process termination function and determines when the
4 process is terminated by receiving the Terminate_Thread
5 signal.

1 25. The virtual device driver of claim 23 wherein
2 the calling part is in communication with the
3 Get_Current_Process_Handle function and retrieves the



- 37 -

4 terminated process database structure by calling the
5 Get_Current_Process_Handle function.

1 26. A virtual device driver situated in a storage
2 medium of a computer system for monitoring the creation
3 and destruction of child processes within an application
4 executing in the computer system controlled by a Windows
5 operating system, each child process having a
6 corresponding child process database structure, the
7 executing application having a corresponding application
8 process database structure, the Windows operating system
9 having a system process creation function for creating a
10 new process and thereupon providing a Create_Thread
11 signal, the operating system further having a
12 Get_Current_Process_Handle function for providing, when
13 called a new process database structure corresponding to
14 the new process, a list provided in the computer system
15 for storing existing child process database structures
16 corresponding to child processes existing within the
17 executing application, the virtual device driver
18 comprising:

19 a determining part in communication with the
20 system process creation function which determines, upon
21 receiving the Create_Thread signal, that the new process
22 has been created;

23 a calling part which:

24 (i) calls the Get_Current_Process_Handle
25 function to retrieve the new process database structure,
26 and



- 38 -

27 (ii) retrieves, using a pointer in the new
28 process database structure, a parent process database
29 structure corresponding to a parent process of the new
30 process;

31 a comparing part which compares the parent
32 process database structure with the application process
33 database structure;

34 an identifier part which identifies, if the
35 parent process database structure is the same as the
36 application process database structure, the new process as
37 an existing child process; and

38 an adder part which adds, if the new process is
39 identified as the existing child process, the new process
40 database structure to the list of existing child process
41 database structures.

1 27. The virtual device driver of claim 26, the
2 operating system further having a system process
3 termination function for terminating a process and
4 thereupon providing a Terminate_Thread signal, wherein the
5 Get_Current_Process_Handle function further provides, when
6 called, a terminated process database structure
7 corresponding to the terminated process, the virtual
8 device driver further comprising:

9 a determining part which determines, upon
10 receiving the Terminate_Thread signal, that the process
11 has been terminated;

12 a calling part which calls the
13 Get_Current_Proccess Handle function to retrieve the



- 39 -

14 terminated process database structure;
15 a comparing part which compares the terminated
16 process database structure with the existing child process
17 database structures in the list;
18 an identifier part which identifies, if the
19 terminated process database structure is the same as one
20 of the existing child process database structures, the
21 terminated process as a terminated child process; and
22 a removing part which removes, if the terminated
23 process is identified as the terminated child process, the
24 terminated process database structure from the list of
25 existing child process database structures.

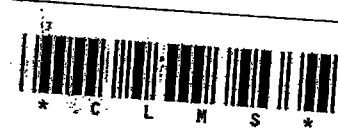
1 28. A processor readable storage medium having
2 processor readable code embodied on the processor readable
3 storage medium, the processor readable code for
4 programming a processor to perform a method of monitoring
5 the creation and destruction of child processes within an
6 application executing in a computer system controlled by
7 an operating system, each child process having a
8 corresponding child process database structure, the
9 executing application having a corresponding application
10 process database structure, the method comprising:
11 providing a list for storing existing child
12 process database structures corresponding to child
13 processes existing within the executing application;
14 determining when a new process is created during
15 execution of the application;
16 retrieving a new process database structure



- 40 -

17 corresponding to the new process;
18 retrieving, using the new process database
19 structure, a parent process database structure
20 corresponding to a parent process of the new process;
21 comparing the parent process database structure
22 with the application process database structure;
23 identifying, if the parent process database
24 structure is the same as the application process database
25 structure, the new process as an existing child process;
26 and
27 adding, if the new process is identified as the
28 existing child process, the new process database structure
29 to the list of existing child process database structures.

1 29. The processor readable storage medium of claim
2 28, the method further comprising:
3 determining when a process is terminated during
4 execution of the application;
5 retrieving a terminated process database
6 structure corresponding to the terminated process;
7 comparing the terminated process database
8 structure with the existing child process database
9 structures in the list;
10 identifying, if the terminated process database
11 structure is the same as one of the existing child process
12 database structures, the terminated process as a
13 terminated child process; and
14 removing, if the terminated process is
15 identified as the terminated child process, the terminated



- 41 -

16 process database structure from the list of existing child
17 process database structures.

1 30. The processor readable storage medium of claim
2 28 wherein the list of existing child process database
3 structures is maintained in a buffer in the computer
4 system, the method further comprising:

5 providing a pointer which points to one of the
6 existing child process database structures in the list;
7 and

8 after adding the new process database structure
9 to the list, moving the pointer to point to the new
10 process database structure.

1 31. The processor readable storage medium of claim
2 28 wherein the list of existing child process database
3 structures is maintained in a buffer in the computer
4 system, the method further comprising:

5 providing a pointer which points to one of the
6 existing child process database structures in the list;
7 after adding the new process database structure
8 to the list, moving the pointer to point to the new
9 process database structure; and

10 after removing the terminated process database
11 structure from the list, moving the pointer to point to
12 one of the existing child process database structures most
13 recently added to the list.

1 32. The processor readable storage medium of claim



- 42 -

2 28 wherein the pointer is maintained in a counter in the
3 computer system, and wherein:

4 moving the pointer to point to the new process
5 database structure includes incrementing the counter; and

6 moving the pointer to point to the existing child
7 process most recently added to the list includes
8 decrementing the counter.

1 33. The processor readable storage medium of claim
2 28 wherein the executing application is one of a series of
3 applications stored within the computer system for
4 execution, the method further comprising:

5 periodically determining whether the list
6 includes any existing child process database structures;
7 and

8 executing, if the list includes no existing
9 child process database structures, a next application in
10 the series of applications.

1 34. The processor readable storage medium of claim
2 28 wherein the executing application is one of a series of
3 applications stored within the computer system for
4 execution, the method further comprising:

5 periodically determining whether the counter has
6 a value of zero; and

7 executing, if the counter has a value of zero,
8 a next application in the series of applications.

1 35. The processor readable storage medium of claim



- 43 -

2 34 wherein the series of applications is at least a
3 portion of an installation program.

1 36. The processor readable storage medium of claim
2 28 wherein the operating system is a Windows operating
3 system having a system process creation function for
4 creating the new process and thereupon providing a
5 Create_Thread signal, the operating system further having
6 a Get_Current_Process_Handle function for providing the
7 new process database structure.

1 37. The processor readable storage medium of claim
2 28 wherein determining when the new process is created
3 includes receiving the Create_Thread signal.

1 38. The processor readable storage medium of claim
2 28 wherein retrieving the new process database structure
3 includes calling the Get_Current_Process_Handle function.

1 39. The processor readable storage medium of claim
2 28 wherein retrieving the parent process database
3 structure includes using a pointer in the new process
4 database structure which identifies the parent process
5 database structure.

1 40. The processor readable storage medium of claim
2 30 wherein the operating system is a Windows operating
3 system having a system process creation function for
4 creating the new process and thereupon providing a



- 44 -

5 Create_Thread signal, and a system process termination
6 function for terminating the process and thereupon
7 providing a Terminate_Thread signal, and a
8 Get_Current_Process_Handle function for providing the
9 terminated process database structure.

1 41. The processor readable storage medium of claim
2 40 wherein determining when the process is terminated
3 includes receiving the Terminate_Thread signal.

1 42. The processor readable storage medium of claim
2 40 wherein retrieving the terminated process database
3 structure includes calling the Get_Current_Process_Handle
4 function.

1 43. A processor readable storage medium having
2 processor readable code embodied on the processor readable
3 storage medium, the processor readable code for
4 programming a processor to perform a method for monitoring
5 the creation and destruction of child processes within an
6 application executing in a computer system controlled by
7 a Windows operating system, each child process having a
8 corresponding child process database structure, the
9 executing application having a corresponding application
10 process database structure, the Windows operating system
11 having a system process creation function for creating a
12 new process and thereupon providing a Create_Thread
13 signal, the operating system further having a
14 Get_Current_Process_Handle function for providing, when



- 45 -

15 called a new process database structure corresponding to
16 the new process, the method comprising:
17 providing a list for storing existing child
18 process database structures corresponding to child
19 processes existing within the executing application;
20 determining, upon receiving the Create_Thread
21 signal, that the new process has been created;
22 calling, after determining that the new process
23 has been created, the Get_Current_Process_Handle function
24 to retrieve the new process database structure;
25 identifying, using a pointer in the new process
26 database structure, a parent process database structure
27 corresponding to a parent process of the new process;
28 comparing the parent process database structure
29 with the application process database structure;
30 identifying, if the parent process database
31 structure is the same as the application process database
32 structure, the new process as an existing child process;
33 and
34 adding, if the new process is identified as the
35 existing child process, the new process database structure
36 to the list of existing child process database structures.

1 44. The processor readable storage medium of claim
2 43, the operating system further having a system process
3 termination function for terminating a process and
4 thereupon providing a Terminate_Thread signal, wherein the
5 Get_Current_Process_Handle function further provides, when
6 called, a terminated process database structure



- 46 -

7 corresponding to the terminated process, the method
8 further comprising:
9 determining, upon receiving the Terminate_Thread
10 signal, that the process has been terminated;
11 calling, after determining that the process has
12 been terminated, the Get_Current_Process_Handle function
13 to retrieve the terminated process database structure;
14 comparing the terminated process database
15 structure with the existing child process database
16 structures in the list;
17 identifying, if the terminated process database
18 structure is the same as one of the existing child process
19 database structures, the terminated process as a
20 terminated child process; and
21 removing, if the terminated process is
22 identified as the terminated child process, the terminated
23 process database structure from the list of existing child
24 process database structures.

1 45. A virtual device driver situated in a storage
2 medium of a computer system for monitoring the creation
3 and destruction of child processes within an application
4 executing in the computer system controlled by an
5 operating system, each child process having a
6 corresponding child process database structure, the
7 executing application having a corresponding application
8 process database structure, a list provided for storing
9 existing child process database structures corresponding
10 to child processes existing within the executing



- 47 -

11 application, the virtual device driver comprising:
12 determining means for determining when a new
13 process is created during execution of the application;
14 calling means for:
15 (i) retrieving a new process database
16 structure corresponding to the new process, and
17 (ii) retrieving, using the new process
18 database structure, a parent process database structure
19 corresponding to a parent process of the new process;
20 comparing means for comparing the parent process
21 database structure with the application process database
22 structure;
23 identifier means for identifying, if the parent
24 process database structure is the same as the application
25 process database structure, the new process as an existing
26 child process; and
27 adding means for adding, if the new process is
28 identified as the existing child process, the new process
29 database structure to the list of existing child process
30 database structures.

1 46. The virtual device driver of claim 45 further
2 comprising:
3 determining means for determining when a process
4 is terminated during execution of the application;
5 calling means for retrieving a terminated
6 process database structure corresponding to the terminated
7 process;
8 comparing means for comparing the terminated



- 48 -

9 process database structure with the existing child process
10 database structures in the list;
11 identifier means for identifying, if the
12 terminated process database structure is the same as one
13 of the existing child process database structures, the
14 terminated process as a terminated child process; and
15 removing means for removing, if the terminated
16 process is identified as the terminated child process, the
17 terminated process database structure from the list of
18 existing child process database structures.

1 47. The virtual device driver of claim 45 wherein
2 the operating system is a Windows operating system having
3 a system process creation function for creating the new
4 process and thereupon providing a Create_Thread signal,
5 the operating system further having a
6 Get_Current_Process_Handle function for providing the new
7 process database structure.

1 48. The virtual device driver of claim 47 wherein
2 the determining means is in communication with the system
3 process creation function and determines when the new
4 process is created by receiving the Create_Thread signal.

1 49. The virtual device driver of claim 47 wherein
2 the calling means is in communication with the
3 Get_Current_Process_Handle function and retrieves the new
4 process database structure by calling the
5 Get_Current_Process_Handle function.



- 49 -

1 50. The virtual device driver of claim 46 wherein
2 the operating system is a Windows operating system having
3 a system process creation function for creating the new
4 process and thereupon providing a Create_Thread signal,
5 and a system process termination function for terminating
6 the process and thereupon providing a Terminate_Thread
7 signal, and a Get_Current_Process_Handle function for
8 providing the terminated process database structure.

1 51. The virtual device driver of claim 50 wherein
2 the determining means is in communication with the system
3 process termination function and determines when the
4 process is terminated by receiving the Terminate_Thread
5 signal.

1 52. The virtual device driver of claim 50 wherein
2 the calling means is in communication with the
3 Get_Current_Process_Handle function and retrieves the
4 terminated process database structure by calling the
5 Get_Current_Process_Handle function.

1 53. A virtual device driver situated in a storage
2 medium of a computer system for monitoring the creation
3 and destruction of child processes within an application
4 executing in the computer system controlled by a Windows
5 operating system, each child process having a
6 corresponding child process database structure, the
7 executing application having a corresponding application



- 50 -

8 process database structure, the Windows operating system
9 having a system process creation function for creating a
10 new process and thereupon providing a Create_Thread
11 signal, the operating system further having a
12 Get_Current_Process_Handle function for providing, when
13 called a new process database structure corresponding to
14 the new process, a list provided in the computer system
15 for storing existing child process database structures
16 corresponding to child processes existing within the
17 executing application, the virtual device driver
18 comprising:

19 determining means in communication with the
20 system process creation function for determining, upon
21 receiving the Create_Thread signal, that the new process
22 has been created;

23 calling means for:

24 (i) calling the Get_Current_Proccess_Handle
25 function to retrieve the new process database structure,
26 and

27 (ii) retrieving, using a pointer in the new
28 process database structure, a parent process database
29 structure corresponding to a parent process of the new
30 process;

31 comparing means for comparing the parent process
32 database structure with the application process database
33 structure;

34 identifier means for identifying, if the parent
35 process database structure is the same as the application
36 process database structure, the new process as an existing



- 51 -

37 child process; and
38 adder means for adding, if the new process is
39 identified as the existing child process, the new process
40 database structure to the list of existing child process
41 database structures.

1 54. The virtual device driver of claim 53, the
2 operating system further having a system process
3 termination function for terminating a process and
4 thereupon providing a Terminate_Thread signal, wherein the
5 Get_Current_Process_Handle function further provides, when
6 called, a terminated process database structure
7 corresponding to the terminated process, the virtual
8 device driver further comprising:
9 determining means for determining, upon
10 receiving the Terminate_Thread signal, that the process
11 has been terminated;
12 calling means for calling the
13 Get_Current_Process_Handle function to retrieve the
14 terminated process database structure;
15 comparing means for comparing the terminated
16 process database structure with the existing child process
17 database structures in the list;
18 identifier means for identifying, if the
19 terminated process database structure is the same as one
20 of the existing child process database structures, the
21 terminated process as a terminated child process; and
22 removing means for removing, if the terminated
23 process is identified as the terminated child process, the



- 52 -

24 terminated process database structure from the list of
25 existing child process database structures.

1 55. A method for monitoring the creation and
2 destruction of child processes within a first application
3 executing in a computer system controlled by an operating
4 system, a second application stored on a storage medium in
5 the computer system for execution, each child process
6 having a corresponding child process database structure,
7 the executing application having a corresponding
8 application process database structure, the method
9 comprising:

10 providing a list for storing existing child
11 process database structures corresponding to child
12 processes existing within the executing application;

13 determining whether the list includes any
14 existing child process database structures; and

15 executing, if the list includes no existing
16 child process database structures, the second application.

1 56. A computer system having an application stored
2 therein for execution and controlled by an operating
3 system, a list provided in the computer system for storing
4 existing child process database structures corresponding
5 to child processes existing within the executing
6 application, the system comprising:

7 a processor;

8 a storage medium;

9 a virtual device driver situated in the storage



- 53 -

10 medium for monitoring the creation and destruction of
11 child processes within the application when executing in
12 the computer system, each child process having a
13 corresponding child process database structure, the
14 executing application having a corresponding application
15 process database structure, the virtual device driver
16 comprising:

17 a determining part which determines when a new
18 process is created during execution of the application;

19 a calling part which:

20 (i) retrieves a new process database
21 structure corresponding to the new process, and

22 (ii) retrieves, using the new process
23 database structure, a parent process database structure
24 corresponding to a parent process of the new process;

25 a comparing part which compares the parent
26 process database structure with the application process
27 database structure;

28 an identifier part which identifies, if the
29 parent process database structure is the same as the
30 application process database structure, the new process as
31 an existing child process; and

32 an adding part which adds, if the new process is
33 identified as the existing child process, the new process
34 database structure to the list of existing child process
35 database structures.

1 57. The computer system of claim 56 wherein the
2 virtual device driver further comprises:



- 54 -

3 a determining part which determines when a
4 process is terminated during execution of the application;
5 a calling part which retrieves a terminated
6 process database structure corresponding to the terminated
7 process;
8 a comparing part which compares the terminated
9 process database structure with the existing child process
10 database structures in the list;
11 an identifier part which identifies, if the
12 terminated process database structure is the same as one
13 of the existing child process database structures, the
14 terminated process as a terminated child process; and
15 a removing part which removes, if the terminated
16 process is identified as the terminated child process, the
17 terminated process database structure from the list of
18 existing child process database structures.

1 58. The computer system of claim 56 wherein the
2 operating system is a Windows operating system having a
3 system process creation function for creating the new
4 process and thereupon providing a Create_Thread signal,
5 the operating system further having a
6 Get_Current_Process_Handle function for providing the new
7 process database structure.

1 59. The computer system of claim 58 wherein the
2 determining part of the virtual device driver is in
3 communication with the system process creation function
4 and determines when the new process is created by



- 55 -

5 receiving the Create_Thread signal.

1 60. The computer system of claim 58 wherein the
2 calling part of the virtual device driver is in
3 communication with the Get_Current_Process_Handle function
4 and retrieves the new process database structure by
5 calling the Get_Current_Process_Handle function.

1 61. The computer system of claim 57 wherein the
2 operating system is a Windows operating system having a
3 system process creation function for creating the new
4 process and thereupon providing a Create_Thread signal,
5 and a system process termination function for terminating
6 the process and thereupon providing a Terminate_Thread
7 signal, and a Get_Current_Process_Handle function for
8 providing the terminated process database structure.

1 62. The computer system of claim 61 wherein the
2 determining part of the virtual device driver is in
3 communication with the system process termination function
4 and determines when the process is terminated by receiving
5 the Terminate_Thread signal.

1 63. The computer system of claim 61 wherein the
2 calling part of the virtual device driver is in
3 communication with the Get_Current_Process_Handle function
4 and retrieves the terminated process database structure by
5 calling the Get_Current_Process_Handle function.



- 56 -

1 64. A computer system having an application stored
2 therein for execution and controlled by a Windows
3 operating system, , the Windows operating system having a
4 system process creation function for creating a new
5 process and thereupon providing a Create_Thread signal,
6 the operating system further having a
7 Get_Current_Process_Handle function for providing, when
8 called a new process database structure corresponding to
9 the new process, a list provided in the computer system
10 for storing existing child process database structures
11 corresponding to child processes existing within the
12 executing application, the system comprising:

13 a processor;

14 a storage medium;

15 a virtual device driver situated in the storage
16 medium for monitoring the creation and destruction of
17 child processes within the application when executing in
18 the computer system, each child process having a
19 corresponding child process database structure, the
20 executing application having a corresponding application
21 process database structure the virtual device driver
22 comprising:

23 a determining part in communication with the
24 system process creation function which determines, upon
25 receiving the Create_Thread signal, that the new process
26 has been created;

27 a calling part which:

28 (i) calls the Get_Current_Proccess_Handle
29 function to retrieve the new process database structure,



- 57 -

30 and
31 (ii) retrieves, using a pointer in the new
32 process database structure, a parent process database
33 structure corresponding to a parent process of the new
34 process;
35 a comparing part which compares the parent
36 process database structure with the application process
37 database structure;
38 an identifier part which identifies, if the
39 parent process database structure is the same as the
40 application process database structure, the new process as
41 an existing child process; and
42 an adder part which adds, if the new process is
43 identified as the existing child process, the new process
44 database structure to the list of existing child process
45 database structures.

1 65. The computer system of claim 64, the operating
2 system further having a system process termination
3 function for terminating a process and thereupon providing
4 a Terminate_Thread signal, wherein the
5 Get_Current_Process_Handle function further provides, when
6 called, a terminated process database structure
7 corresponding to the terminated process, the virtual
8 device driver further comprising:
9 a determining part which determines, upon
10 receiving the Terminate_Thread signal, that the process
11 has been terminated;
12 a calling part which calls the

- 58 -

13 Get_Current_Process Handle function to retrieve the
14 terminated process database structure;
15 a comparing part which compares the terminated
16 process database structure with the existing child process
17 database structures in the list;
18 an identifier part which identifies, if the
19 terminated process database structure is the same as one
20 of the existing child process database structures, the
21 terminated process as a terminated child process; and
22 a removing part which removes, if the terminated
23 process is identified as the terminated child process, the
24 terminated process database structure from the list of
25 existing child process database structures.

