

Description

This invention relates to a method for introducing changes into a stored program-controlled system without requiring partial or total system shutdown and subsequent initialization.

Background of the Invention

Program-controlled systems have been applied to a wide range of data processing and control applications. The functional flexibility of these systems comes from the program, stored in memory of such systems, which controls their operation.

In program-controlled systems, it is frequently necessary to introduce program changes after initial installation to accommodate new services and/or new hardware. The most common way of introducing such changes is by shutting down the system and completely reinitializing it. Unfortunately, this renders the system incapable of providing service for the initialization interval, usually, a period of several minutes or even hours. Moreover, initialization frequently destroys some portions of longer term data which may affect subsequent system performance; it may, for example, clear data provided by a telephone customer defining some aspect of his service. Initialization is especially undesirable if the program-controlled system controls continuous processes, such as chemical processes; careful plans must be made to introduce such initialization at a time when the controlled chemical processes have all been stopped or have reached some equilibrium.

The problems of inserting changes into the programs of working systems without interrupting their operation are considered in R.S. Fabry: "To Design A System In Which Modules Can Be Changed On The Fly", 2nd International Conference on Software Engineering, 13-15 October 1976, San Francisco, California, pp.470-476 (Fabry), and R.E. Novak: "Dynamic Linking In a Real-Time Environment", GTE Automatic Electric World-Wide Communications Journal, March-April, 1981, pp. 57-60 (Novak). Fabry relates to techniques using indirect addressing to change the address of a desired sub-routine from its original address to the address where a revised version of the sub-routine is stored. Novak relates to techniques for changing from an original version of a sub-routine to a revised version through the use of a dynamic linking arrangement.

Another method of introducing program changes with partial initialization is accomplished, for example, in telephone electronic switching systems, such as the No. 1 Electronic Switching System described in the Bell System Technical Journal, September, 1964 issue, and in U.S. Patent 3,570,008. The special characteristics of these systems have made it possible to introduce changes without disturbing those customers who are already connected to other customers. All data dealing with completed connections and with customer services is isolated and specially

treated by initialization programs. These systems have duplicate processors, one of which is on-line and the other of which is a standby unit ready to take over control. A modified version of a program is loaded into one of the processors while the processor is in the standby mode. It is then switched on-line and executes an initialization program which does not modify any of the data associated with completed connections or customer services. The modified version of the program is then also loaded into the other processor and the system is in a stable state. Customers who were in the middle of dialing will not have their calls completed and therefore this kind of change is made during hours of very low system activity to minimize disturbance.

This method of introducing changes is not applicable to more sophisticated modern program-controlled systems. These systems are normally run under the control of a complex program called an operating system which initiates other programs called processes. The actual execution of system tasks is normally accomplished by the processes through execution of a series of program segments referred to as program functions. Execution of the program functions may be initiated by a function call from any process in the system. Program functions access a common data base, large portions of which are controlled by many processes, in which the linkages among various associated data items are manifold and complex. Such complex data structures drastically complicate the kind of special treatment of a selected portion of data used in the above-noted prior art systems. Thus, initializing some processes while retaining others, while maintaining a consistent data base, is very difficult. Furthermore, it is impossible to predict the nature of program changes and to plan for these changes in the design of the data base. For all of the above reasons, it has been a problem to introduce all but the most minor program changes into the more sophisticated modern program-controlled systems without massive initialization.

The problem has been solved in accordance with this invention (claims 1 and 6) in which the method comprises the steps of generating a system sequence number which is changed only monotonically in a given direction, assigning the system sequence number to a process as a process sequence number at the time the process is initiated, storing a modified version of a selected program function into the program memory while retaining a present version of the selected program function, recording a decision function sequence number, the decision function sequence number being beyond the system sequence number in the given direction, changing the system sequence number in the given direction so that it reaches a value at least as far in the given direction as the decision function sequence number, and executing the present version if the process sequence number is before the decision function sequence number in the

given direction, and executing the modified version if the process sequence number is at least as far as the decision function sequence number in the given direction.

Summary of the Invention

It is an object of this invention to allow program changes to be introduced into program-controlled systems without interfering with ongoing processes, even while the system is operating under normal load.

In accordance with this invention, changes are introduced into the program of a program-controlled system by storing a modified version of a program function while retaining the present version, then causing all processes initiated prior to the change to execute the present version of that function, and causing all processes initiated after the change to execute the modified version of that function. This is accomplished with the help of a system sequence number, maintained by the system. A system sequence number is changed only monotonically in a given direction and is changed at least every time that a modified program function is introduced into the system. The present value of the system sequence number is recorded with a process whenever the process is initiated. Whenever a modified version of a program function is stored in the system, a decision function sequence number is assigned to the modified function having a value which is beyond, in the given direction, the value of the process sequence number of any presently active process which uses that program function. The system sequence number is modified thereafter, so that it is at least as far in the given direction as the decision function sequence number. Subsequently, whenever a process calls the function which has been modified, the decision function sequence number is compared with the process number. If the result of the comparison indicates that the modified version was stored in the system before the calling process was initiated, the modified version of the program function is executed; otherwise, the present version is executed.

In one specific embodiment of the invention, program functions can be called by a process through the use of a vector in a stored transfer vector table, which normally points directly to the desired program function. If a modified program function is introduced into the system, the vector will, instead, point to a decision function. The decision function then makes the decision of whether the present or modified version of the program function is to be executed. If the present version is to be executed, a program branch to the present version is executed; otherwise, a program branch to the modified version is executed. Advantageously, the decision function may directly precede the modified version thereby simplifying the process of reclaiming storage space when old versions of program functions are no longer needed.

Periodically, an audit program may be run to

determine if any active processes which deal with a program function, which has been modified, have a sequence number, that is, in the given direction, before that of the modified version of the program function. If there are no such processes, the present version of the program function may be deleted, the vector may point directly to the modified version of the function instead of pointing to the decision function, and the memory block occupied by the present version of the program function becomes available space for other new or modified program functions.

Advantageously, sequence numbers may be any set of numbers which change monotonically, either by being incremented, or by being decremented, so that it is possible to tell whether a given sequence number was assigned before another sequence number. Sequence numbers must change at least as often as program changes are introduced, and may change much more frequently. For example, it is possible to use a quantity representing date and time as a sequence number, since this quantity keeps changing monotonically by being positively incremented. This offers the further advantage of using a sequence number which may have added use, for example, to detect processes which have been in an active state for an excessive period of time. Also, if it is desired to load a modification some time before the modification is to be activated, the decision function sequence number can be preset to a quantity representing the activation time; the modification will then not be used before that activation time.

Brief Description of the Drawing

The invention will be better understood from the following detailed description when read with reference to the drawing in which:

FIG. 1 is a memory layout of an illustrative embodiment of the invention, showing the addressing relationships between present and modified versions of a program function;

FIG. 2 is a block diagram of a processor of a program-controlled system; and

FIG. 3 is a flow chart of an audit program for reclaiming storage of an old version of a program when no longer needed.

Detailed Description

A program-controlled system has two types of information stored in its memory: programs and data. A program comprises a collection of basic instructions of a processor, each of which instructions directs some elemental step, so that the collection, when executed, accomplishes some broader system task or series of tasks. Data comprises information about the present status of a task, information derived from external sources and previously stored in the system, and information generated to accomplish the functional tasks of the system. Programs carry out their tasks by processing data and by controlling input/output systems in accordance with the present values of data.

In modern program-controlled systems, the basic control of the system resides in a master program, which, along with its associated data is called the operating system. The operating system directs the carrying out of system tasks by creating and putting into effect processes. A process is a program plus associated control data storage, called a process control block. The process control block keeps track of the context of each process including such information as the process identification, present process state, process priority, system time at process initiation, etc. Processes, in turn, accomplish their tasks by calling for the execution of a series of program functions.

Operating normally, a program-controlled system will usually have some processes that are active and some that are quiescent. A process is considered to be active during the period that is actively executing a task, waiting for a block of time to continue its execution of a task, or waiting for input or output devices. Upon completion of a task, or set of tasks, it assumes the quiescent state and remains in that state until it is again initiated to execute another task or set of tasks. When a process enters a quiescent state, a minimum of data is carried over for subsequent use when the process is reinitiated. No direct program references are retained when a process becomes quiescent.

At any stage in the life of a process, a program change may be introduced by storing a modified program function in some available space in memory. However, incorrect results can be generated if a modified program function is used by a process that was already active prior to the introduction of the change. In order to ensure that each process uses the same program function throughout the time it remains active, decision functions are associated with each modified version of a program function. Decision functions are brief programs which determine when the process which called the program function was initiated — before or after the modified version of the program function was loaded into the system. If the modified version was loaded before the process was started, the modified version is used; if the modified version was loaded after the process was started, the present version is used.

A system sequence number is used to keep track of the order of events, i.e., whether a process was initiated before or after a modified version of a program function was loaded into the system. A system sequence number may be any quantity which is changed monotonically so that the ordering in time between two different sequence numbers can be established. For example, if a system sequence number is always incremented positively, then a greater system sequence number is associated with a later point in time than a lesser system sequence number. Time, comprising date and time-of-day, is used as the system sequence number in this embodiment.

In FIG. 1, block 100 contains the system

sequence number, the time, automatically updated by the operating system using the system clock. The system clock is part of processor 200. It is well-known in the art and is not shown in the drawing. The mechanism for updating a time entry in memory is also known and is not described further herein.

A process sequence number representing time is stored in the process control block of a process whenever it is initiated. Process control block 110 contains the process control information associated with a typical process. As stated earlier herein, the process control block includes several items of information associated with the process. Location 111 of block 110, contains the process sequence number which indicates the time when the associated process was initiated. Whenever a process becomes active, the present value of the system sequence number is placed in location 111. A decision function sequence number, representing present time incremented by one second, is assigned to the decision function at the time a modified program function is loaded into memory.

Function blocks 130, 140, and 150 of FIG. 1 each represent an area of memory for storing a program. Each has an area for storing a decision function (131, 141, and 151) and an area for storing a program function (132, 142, and 152). The program functions represent a present version, F, (132), a modified version, F', (142), and a second modification, F'', (152). Decision function block 141 comprises decision function program 144 and decision function sequence number 143 identified as m. The decision function program may be stated as follows: if the process sequence number, n(pc), is less than the decision function sequence number, m, go to f(2) (block 132); else, go to f' (2) (block 142). In other words, if the process was started before the change was introduced, use the present version of program function, F; otherwise, use the modified version of the program function, F'.

Programs are assigned to any available segment of memory. Individual program functions may be used by many processes, but stored only once to save valuable memory space. Since many programs may reference a program function, a list of the present addresses of all program functions is maintained and made available to all calling programs. This list is referred to as a transfer vector table. When a change is loaded into the system, the transfer vector table address is updated to point to the decision function associated with the modified version. When a process calls the program function which was modified, the decision function pointed to by the transfer vector table will determine whether the present or modified version of the program function is to be used. In transfer vector table 120, entry 121 points to the address of the example program function or decision function. If only one version of a program is loaded in the system, entry 121 is the address of the program function; if more than one version is loaded in the system,

entry 121 is the address of a decision function that will permit the correct version of the program to be selected. When the modified version of the program function, F' , is loaded in system memory, the vector table entry at 121 is changed from $f(2)$, the address of the present version of the program function, to $f'(1)$, the address of the decision function associated with F' . As a result, before either version of the program function is executed, the decision function compares process and decision function sequence numbers to decide whether the process was initiated before or after the change was introduced. If the process was initiated before the change, a branch to the present version F occurs; otherwise, the modified version F' is executed.

If there is some need for an additional modification, a second modification represented by F'' in block 150 must be introduced. When F'' is loaded in the system, entry 121 in the transfer vector table is changed to point to $f''(1)$, the decision function associated with F'' . Block 150 comprises decision function, block 151, and program function F'' , block 152. The decision function block comprises decision function sequence number 153, represented by p , and decision function program 154. Decision function program 154 may be stated as follows: if the process sequence number, $n(pc)$, is less than decision function sequence number, p , go to $f'(1)$; else, go to $f''(2)$. In other words, if the process was initiated before F'' was loaded, a jump is made to decision function program 144 associated with version F' ; otherwise F'' is executed. The outcome of decision function program 144 is a jump to the original function, F , if the process sequence number, $n(pc)$, is less than the decision function sequence number m ; otherwise, program function F' is executed. In this example, processes use program functions F , F' , or F'' . The oldest processes use F , intermediate processes use F' , and the most recently initiated processes use F'' .

If it is necessary or desirable to modify a group of functions simultaneously, a decision function is inserted for each such program function, and the same sequence number assigned to all the decision functions associated with that group. Delayed activation of a program modification is also possible. The decision function sequence number associated with such a modification would be the delayed activation time. The modification would then not take effect until activation time.

This description has been in terms of a modification of a single program function. Clearly, many program functions can be modified using the techniques of this invention. Program modifications for many different program functions, loaded at different times, each modified program function having its own associated decision function, may exist in the system at the same time.

FIG. 2 illustrates three major components of a program-controlled system, the central processor 202, the main memory 201, and input/output equipment 203. The central processing unit and its

associated memory may be any of a number of modern processors well-known in the art. The input/output equipment may include supplementary memory devices such as magnetic tape units, and fixed or moving head disks. The central processing unit executes processes and program functions which are stored in memory. In executing such processes and program functions, the central processing unit accesses and modifies data stored in the memory. The central processing unit executes simple steps such as the comparison of two numbers, performing a conditional branch based on such a comparison, incrementing a quantity in memory, moving quantities stored in memory from one location to another, branching unconditionally to a directly or indirectly indicated address. The central processing unit is also capable of loading a modified version of a program function from input/output equipment 203 into main memory 201.

Periodically, audit programs are run both to reclaim memory space occupied by old versions of program functions that are no longer used and to update the transfer vector table with addresses of modified versions of program functions, in order to by-pass the no-longer needed decision function. FIG. 3 is a flowchart of an audit program that examines processes which might be using either the old program function F or the modified version F' . In this example, it is assumed that the audit program is run after modification F' is introduced into the system and before modification F'' occurs.

The audit program may be considered in three parts: initial steps (300—302), looping (303—306), and reclaiming memory (307—308). The audit program is initiated by the operating system. The audit program must read the decision function sequence number, m , which is to be compared with the process sequence number of all active processes which might use the present or modified version of the program function. The first process is then accessed. The looping series of steps starts with a check to see if the process is quiescent. If the process is not quiescent, the process sequence number is compared with the decision function sequence number. If the process sequence number is not at least as great as the decision function sequence number, the audit program is terminated; this means that some active process is still using the present version of the program function. Otherwise, the looping process continues until all appropriate processes have been examined. If the audit program has not been terminated after all appropriate processes have been examined, the audit program can proceed to reclaim memory. It does so by changing the vector table entry to point directly to the modified version, and by making the memory space occupied by the present version available to the operating system for reassignment for other purposes.

As depicted in block 300 of the flowchart in FIG. 3, the audit program starts to execute after having been started by the operating system. The decision function sequence number, m , asso-

14

ciated with F', is read by the audit program in 301. At 302, the audit program accesses the first process that uses F or F'. Then, at decision points 303 and 304, respectively, the audit program determines whether the process is quiescent, and whether its sequence number, n(pc), is greater than or equal to the decision function sequence number associated with F'. If both of these decisions are negative, the audit program concludes that the process is active and uses F, the present version of the program function. Since F is still being used by an active process, it cannot be deleted. Consequently, the audit program terminates at 309. If either of the decisions is positive, the audit program concludes that the process is either quiescent and prepared to use the modified version of the program function F' when it is initiated, or is already using F'.

The audit program proceeds to decision point 305 where it determines if there are any more processes that use F or F'. If there are more such processes, the audit program accesses the next process at block 306 and cycles through decision points 303, 304, and 305 again.

The audit program repeats the cycle until all the processes that use functions F or F' have been examined. If no active processes using F have been discovered, there is no reason to retain the address of the decision function in the transfer vector table. Therefore, in block 307, the audit program changes the transfer table entry 121 to point directly to address F' (2) of the new program function F'. Because program function F is no longer referenced by any process and no longer can be addressed from the decision function, it is wastefully occupying space in memory. Since its space can be used for some other purpose, the audit program deletes program function F, releases block 130 as free memory space at 308, and terminates at 309. Note that block 130 also includes the space formerly occupied by a decision function when F was introduced as a program modification. By keeping the modification and its associated decision function together, the entire block of memory may be made available when F is no longer needed.

In some systems, it may be difficult for the audit program to determine whether a process is active or quiescent. In such a case, the check of block 303 may be by-passed. It may also be more convenient to examine all processes whether or not they use F or F'. The only adverse results of such simplifications are that the decision function remains active for a longer period than is necessary, and that the storage space occupied by the present version of the program function cannot be reclaimed as quickly.

A large class of program modifications can be introduced using the techniques of this invention, including modifications in programs which initiate and execute processes. Broadly, such programs are program functions called by the operating system which is also a group of program functions. Any program which is called indirectly may, in principle, be changed using the tech-

niques of this invention. However, certain restrictions arise in making modifications to program functions which use data interpreted differently by the present and modified versions of a program function, particularly, where such data is retained during a quiescent interval of a process, or where such data is shared by two or more processes. Even when accessing such data, careful use of decision functions for data, in conjunction with decision functions for program functions, can allow modifications to be introduced to such data using the techniques of this invention. Note that an implied prerequisite is a single access point, such as a transfer vector for program functions, used whenever a decision function is used for accessing such data.

Claims

1. A method for introducing changes into a stored program-controlled system comprising the steps of: storing a present version of a selected program function in a program memory; storing a modified version of the selected program function in the program memory; generating a system sequence number; assigning the present value of the system sequence number to each process as a process sequence number at the time of initiation of said process; recording a decision function sequence number for the modified version of the selected program function, the value of the decision function sequence number being beyond the present value of any process sequence number in a given direction at the time when the said modified version is stored; changing the system sequence number in the given direction so that it reaches a value at least as far in the given direction as the decision function sequence number; interrogating the program memory in response to a process calling the selected program function; executing said present version of the selected program function if the process sequence number of the calling process is before the decision function sequence number in the given direction, and executing said modified version of the selected program function if the process sequence number of the calling process is at least as far as the decision function sequence number in the given direction.

2. The method in accordance with claim 1 wherein the executing step further comprises the steps of:

comparing the process sequence number of the calling process with the decision function sequence number;

branching to the present version if the result of the comparison indicates that the process sequence number of the calling process is before the decision function number in the given direction; and

branching to the modified version if result of the comparison indicates that the process sequence number of the calling process is at least as far as the decision function sequence number in the given direction.

(74)

ciated with F', is read by the audit program in 301. At 302, the audit program accesses the first process that uses F or F'. Then, at decision points 303 and 304, respectively, the audit program determines whether the process is quiescent, and whether its sequence number, n(pc), is greater than or equal to the decision function sequence number associated with F'. If both of these decisions are negative, the audit program concludes that the process is active and uses F, the present version of the program function. Since F is still being used by an active process, it cannot be deleted. Consequently, the audit program terminates at 309. If either of the decisions is positive, the audit program concludes that the process is either quiescent and prepared to use the modified version of the program function F' when it is initiated, or is already using F'.

The audit program proceeds to decision point 305 where it determines if there are any more processes that use F or F'. If there are more such processes, the audit program accesses the next process at block 306 and cycles through decision points 303, 304, and 305 again.

The audit program repeats the cycle until all the processes that use functions F or F' have been examined. If no active processes using F have been discovered, there is no reason to retain the address of the decision function in the transfer vector table. Therefore, in block 307, the audit program changes the transfer table entry 121 to point directly to address f' (2) of the new program function F'. Because program function F is no longer referenced by any process and no longer can be addressed from the decision function, it is wastefully occupying space in memory. Since its space can be used for some other purpose, the audit program deletes program function F, releases block 130 as free memory space at 308, and terminates at 309. Note that block 130 also includes the space formerly occupied by a decision function when F was introduced as a program modification. By keeping the modification and its associated decision function together, the entire block of memory may be made available when F is no longer needed.

In some systems, it may be difficult for the audit program to determine whether a process is active or quiescent. In such a case, the check of block 303 may be by-passed. It may also be more convenient to examine all processes whether or not they use F or F'. The only adverse results of such simplifications are that the decision function remains active for a longer period than is necessary, and that the storage space occupied by the present version of the program function cannot be reclaimed as quickly.

A large class of program modifications can be introduced using the techniques of this invention, including modifications in programs which initiate and execute processes. Broadly, such programs are program functions called by the operating system which is also a group of program functions. Any program which is called indirectly may, in principle, be changed using the tech-

niques of this invention. However, certain restrictions arise in making modifications to program functions which use data interpreted differently by the present and modified versions of a program function, particularly, where such data is retained during a quiescent interval of a process, or where such data is shared by two or more processes. Even when accessing such data, careful use of decision functions for data, in conjunction with decision functions for program functions, can allow modifications to be introduced to such data using the techniques of this invention. Note that an implied prerequisite is a single access point, such as a transfer vector for program functions, used whenever a decision function is used for accessing such data.

Claims

1. A method for introducing changes into a stored program-controlled system comprising the steps of: storing a present version of a selected program function in a program memory; storing a modified version of the selected program function in the program memory; generating a system sequence number; assigning the present value of the system sequence number to each process as a process sequence number at the time of initiation of said process; recording a decision function sequence number for the modified version of the selected program function, the value of the decision function sequence number being beyond the present value of any process sequence number in a given direction at the time when the said modified version is stored; changing the system sequence number in the given direction so that it reaches a value at least as far in the given direction as the decision function sequence number; interrogating the program memory in response to a process calling the selected program function; executing said present version of the selected program function if the process sequence number of the calling process is before the decision function sequence number in the given direction, and executing said modified version of the selected program function if the process sequence number of the calling process is at least as far as the decision function sequence number in the given direction.

2. The method in accordance with claim 1 wherein the executing step further comprises the steps of:

comparing the process sequence number of the calling process with the decision function sequence number;

branching to the present version if the result of the comparison indicates that the process sequence number of the calling process is before the decision function number in the given direction; and

branching to the modified version if result of the comparison indicates that the process sequence number of the calling process is at least as far as the decision function sequence number in the given direction.



15

3. The method in accordance with claim 1 wherein the step of assigning comprises the step of:

storing the system sequence number in memory locations associated with each of the processes.

4. The method in accordance with claim 2 in which the present version of the selected program function comprises an intermediate version and an older version of the selected program function, and in which the step of branching to the present version if the result of the comparison indicates that the process sequence number of the calling process is before the decision function sequence number in the given direction further comprises the steps of:

making a second comparison between the process sequence number of the calling process and a second decision function sequence number associated with the intermediate version of the selected program function;

branching to the older version if the second comparison indicates that the process sequence number of the calling process is before the second decision function sequence number in the given direction; and

branching to the intermediate version if the second comparison indicates that the process sequence number of the calling process is at least as far as the second decision function sequence number in the given direction.

5. The method in accordance with claim 1, 2, 3 or 4 further comprising the steps of:

periodically, making third comparisons between the process sequence number of at least all active processes which make function calls to the selected program function and the decision function sequence number; and

deleting the present version if none of the process sequence numbers of at least all active processes which make function calls to the selected program function is before the decision function sequence number in the given direction.

6. A program-controlled processor system comprising:

a central processing unit (202); and
a program memory (201);

wherein processes are initiated and executed by function calls to program functions; characterized in that

the program memory (201) stores a present version of a selected program function (132);

stores a modified version of the selected program function (142);

stores a system sequence number (100);

stores a system sequence number as a process sequence number (111) at the time any of the processes is initiated; and

stores a decision function sequence number (143) for the modified version at the time of storing the modified version, the value of the decision function sequence number being beyond the present value of any process sequence number in a given direction;

when a decision function sequence number is

stored, the central processing unit (202) modifies the system sequence number so that it reaches a value at least as far in the given direction as the decision function sequence number;

the central processing unit (202) interrogates the program memory (201, 110, 111, 121, 141, 132, 142) and executes the present version (132) of the selected function in response to a function call by a calling process to the selected function if the process sequence number of the calling process is before the decision function sequence number in the given direction and executes the modified version (142) of the selected function in response to a function call by the calling process to the selected function if the process sequence number of the calling process is at least as far as the decision function number in the given direction.

7. A program-controlled processor system in accordance with claim 6 characterized in that

the central processing unit (202) arranges to compare the process sequence number of the calling process with the decision function sequence number and branches to the present version if the comparison indicates that the process sequence number of the calling process is before the decision function sequence number in the given direction and branches to the modified version if the comparison indicates that the process sequence number of the calling process is at least as far as the decision function sequence number in the given direction.

8. A program-controlled processor system in accordance with claim 7 characterized in that

the program memory (201) stores an intermediate version of the selected program function and an older version in place of the present version;

the central processing unit (202) branches to the present version if the comparison indicates that the process sequence number of the calling process is before the decision function sequence number in the given direction and makes a second comparison between the process sequence number of the calling process and a second decision function sequence number associated with the intermediate version of the selected program function and branches to the older version if the second comparison indicates that the process sequence number of the calling process is before the second decision function sequence number in the given direction and branches to the intermediate version if the second comparison indicates that the process sequence number is at least as far as the second decision function number in the given direction.

Patentansprüche

1. Verfahren zur Einführung von Änderungen in ein Speicherprogramm-gesteuertes System mit den Verfahrensschritten:

Speichern einer augenblicklichen Version einer gewählten Programmfunktion in einem Programmspeicher; Speichern einer modifizierten



Version der gewählten Programmfunktion im Programmspeicher; Erzeugen einer Systemfolgennummer; Zuordnen des augenblicklichen Wertes der Systemfolgennummer zu jedem Prozeß als Prozeßfolgennummer zum Zeitpunkt der Einleitung des Prozesses; Aufzeichnen einer Entscheidungsfunktionsfolgennummer für die modifizierte Version der gewählten Programmfunktion, wobei der Wert der Entscheidungsfunktionsfolgennummer in einer gegebenen Richtung vor bzw. hinter dem augenblicklichen Wert jeder Prozeßfolgennummer zu dem Zeitpunkt liegt, wenn die modifizierte Version gespeichert wird; Ändern der Systemfolgennummer in einer gegebenen Richtung, so daß sie einen Wert erreicht, der wenigstens so weit in der gegebenen Richtung liegt, wie die Entscheidungsfunktionsfolgennummer; Abfragen des Programmspeichers unter Ansprechen auf einen Prozeß, der die gewählte Programmfunktion aufruft; Ausführen der augenblicklichen Version der gewählten Programmfunktion, wenn die Prozeßfolgennummer des aufrufenden Prozesses in der gewählten Richtung vor der Entscheidungsfunktionsfolgennummer liegt, und Ausführen der modifizierten Version der gewählten Programmfunktion, wenn die Prozeßfolgennummer des aufrufenden Prozesses wenigstens so weit in der gewählten Richtung liegt wie die Entscheidungsfunktionsfolgennummer.

2. Verfahren nach Anspruch 1, bei dem der Entscheidungsschritt ferner die Schritte aufweist: Vergleichen der Prozeßfolgennummer des aufrufenden Prozesses mit der Entscheidungsfunktionsfolgennummer; Abzweigen zu der augenblicklichen Version, wenn das Ergebnis des Vergleiches angibt, daß die Prozeßfolgennummer des aufrufenden Prozesses in der gegebenen Richtung vor der Entscheidungsfunktionsfolgennummer liegt; und Abzweigen zu der modifizierten Version, wenn das Ergebnis des Vergleiches anzeigt, daß die Prozeßfolgennummer des aufrufenden Prozesses wenigstens so weit in der gegebenen Richtung liegt wie die Entscheidungsfunktionsfolgennummer.

Verfahren nach Anspruch 1, bei dem der Zuordnungsschritt den Schritt umfaßt: Speichern der Systemfolgennummer in Speicherstellen, die jedem der Prozesse zugeordnet sind.

4. Verfahren nach Anspruch 2, bei dem die augenblickliche Version der gewählten Programmfunktion eine Zwischenversion und eine ältere Version der gewählten Programmfunktion umfaßt, und bei dem der Schritt einer Abzweigung zu der augenblicklichen Version, wenn das Ergebnis des Vergleiches anzeigt, daß die Prozeßfolgennummer des aufrufenden Prozesses in der gegebenen Richtung vor der Entscheidungsfunktionsfolgennummer liegt, ferner die Schritte umfaßt: Durchführen eines zweiten Vergleiches zwischen der Prozeßfolgennummer des aufrufenden Prozesses und einer zweiten Entscheidungsfunktionsfolgennummer, die der Zwischenversion der gewählten Programmfunktion zugeordnet ist; Abzweigen zu der älteren Version, wenn der zweite Vergleich angibt, daß die Prozeßfolgen-

mer des aufrufenden Prozesses in der gegebenen Richtung vor der zweiten Entscheidungsfunktionsfolgennummer liegt; und Abzweigen zu der Zwischenversion, wenn der zweite Vergleich anzeigt, daß die Prozeßfolgennummer des aufrufenden Prozesses wenigstens so weit in der gegebenen Richtung liegt wie die zweite Entscheidungsfunktionsfolgennummer.

5. Verfahren nach Anspruch 1, 2, 3 oder 4 mit ferner den Schritten: periodisches Durchführen dritter Vergleiche zwischen der Prozeßfolgennummer wenigstens aller aktiver Prozesse, die Funktionsaufrufe der gewählten Programmfunktion durchführen, und der Entscheidungsfunktionsfolgennummer; und Fallenlassen der augenblicklichen Version, wenn keine der Prozeßfolgennummern wenigstens aller aktiven Prozesse, die Funktionsaufrufe der gewählten Programmfunktion durchführen, in der gegebenen Richtung vor der Entscheidungsfunktionsfolgennummer liegen.

6. Programmgesteuertes Prozessorsystem mit einer zentralen Prozessoreinheit (202) und einem Programmspeicher (201), wobei Prozesse durch Funktionsaufrufe von Programmfunktionen eingeleitet und ausgeführt werden, dadurch gekennzeichnet, daß der Programmspeicher (201) speichert: eine augenblickliche Version einer gewählten Programmfunktion (132), eine modifizierte Version der gewählten Programmfunktion (142), eine Systemfolgennummer (100), die Systemfolgennummer als Prozeßfolgennummer (111) zu dem Zeitpunkt, zu dem jeder der Prozesse eingeleitet wird, und eine Entscheidungsfunktionsfolgennummer (143) für die modifizierte Version zum Zeitpunkt der Speicherung der modifizierten Version, wobei der Wert der Entscheidungsfunktionsfolgennummer in einer gegebenen Richtung vor bzw. hinter dem augenblicklichen Wert jeder Prozeßfolgennummer liegt, daß, wenn eine Entscheidungsfunktionsfolgennummer gespeichert wird, die zentrale Prozessoreinheit (202) die Systemfolgennummer so ändert, daß sie einen Wert erreicht, der wenigstens so weit in der gegebenen Richtung liegt, wie die Entscheidungsfunktionsfolgennummer, und daß die zentrale Prozessoreinheit (202) den Programmspeicher (201, 110, 111, 121, 141, 132, 142) abfragt und die augenblickliche Version (132) der gewählten Programmfunktion unter Ansprechen auf einen Funktionsaufruf der gewählten Funktion durch einen aufrufenden Prozeß ausführt, wenn die Prozeßfolgennummer des aufrufenden Prozesses in der gegebenen Richtung vor der Entscheidungsfunktionsfolgennummer liegt, und die modifizierte Version (142) der gewählten Programmfunktion unter Ansprechen auf einen Funktionsaufruf der gewählten Programmfunktion durch den aufrufenden Prozeß ausführt, wenn die Prozeßfolgennummer des aufrufenden Prozesses wenigstens so weit in der gegebenen Richtung liegt wie die Entscheidungsfunktionsfolgennummer.

7. Programmgesteuertes Prozessorsystem nach Anspruch 6, dadurch gekennzeichnet, daß die zentrale Prozessoreinheit (202) die Prozeßfolgennummer des aufrufenden Prozesses mit der Ent-



(19)

scheidungsfunktionsfolgennummer vergleicht und zu der augenblicklichen Version abzweigt, wenn der Vergleich angibt, daß die Prozeßfolgennummer des aufrufenden Prozesses in der gegebenen Richtung vor der Entscheidungsfunktionsfolgennummer liegt, und zu der modifizierten Version abzweigt, wenn der Vergleich anzeigt, daß die Prozeßfolgennummer des aufrufenden Prozesses wenigstens so weit in der gegebenen Richtung liegt, wie die Entscheidungsfunktionsfolgennummer.

8. Programmgesteuertes Prozessorsystem nach Anspruch 7, dadurch gekennzeichnet, daß der Programmspeicher (201) eine Zwischenversion der gewählten Programmfunktion und eine ältere Version anstelle der augenblicklichen Version speichert und daß die zentrale Prozessoreinheit (202) zu der augenblicklichen Version abzweigt, wenn der Vergleich anzeigt, daß die Prozeßfolgennummer des aufrufenden Prozesses in der gegebenen Richtung vor der Entscheidungsfunktionsfolgennummer liegt, einen zweiten Vergleich zwischen der Prozeßfolgennummer des aufrufenden Prozesses und einer zweiten, der Zwischenversion der gewählten programmfunktion zugeordneten Entscheidungsfunktionsfolgennummer durchführt und zu der älteren Version abzweigt, wenn der zweite Vergleich anzeigt, daß die Prozeßfolgennummer des aufrufenden Prozesses in der gegebenen Richtung vor der zweiten Entscheidungsfunktionsfolgennummer liegt, und zu der Zwischenversion abzweigt, wenn der zweite Vergleich anzeigt, daß die Prozeßfolgennummer wenigstens so weit in der gegebenen Richtung liegt wie die zweite Entscheidungsfunktionsfolgennummer.

Revendications

1. Un procédé pour introduire des changements dans un système commandé par programme enregistré, comprenant les opérations suivantes: on enregistre dans une mémoire de programme une version présente d'une fonction de programme sélectionnée; on enregistre dans la mémoire de programme une version modifiée de la fonction de programme sélectionnée; on génère un numéro de séquence de système; on affecte la valeur présente du numéro de séquence de système à chaque processus, en tant que numéro de séquence de processus, au moment du lancement de ce processus; on enregistre un numéro de séquence de fonction de décision pour la version modifiée de la fonction de programme sélectionnée, la valeur du numéro de séquence de fonction de décision étant au-delà de la valeur présente de tout numéro de séquence de processus, dans une direction donnée, au moment où la version modifiée est enregistrée; on change le numéro de séquence de système dans la direction donnée, de façon qu'il atteigne une valeur au moins aussi éloignée, dans la direction donnée, que le numéro de séquence de fonction de décision; on interroge la mémoire de programme lorsqu'un processus appelle la fonction de pro-

gramme sélectionnée; on exécute la version présente de la fonction de programme sélectionnée si le numéro de séquence de processus du processus appelant est en deçà du numéro de séquence de fonction de décision, dans la direction donnée, et on exécute la version modifiée de la fonction de programme sélectionnée si le numéro de séquence de processus du processus appelant est au moins aussi éloigné que le numéro de séquence de fonction de décision, dans la direction donnée.

2. Le procédé selon la revendication 1, dans lequel l'opération d'exécution comprend en outre les opérations suivantes: on compare le numéro de séquence de processus du processus appelant avec le numéro de séquence de fonction de décision; on effectue un branchement vers la version présente si le résultat de la comparaison indique que le numéro de séquence de processus du processus appelant est en deçà du numéro de fonction de décision dans la direction donnée; et on effectue un branchement vers la version modifiée si le résultat de la comparaison indique que le numéro de séquence de processus du processus appelant est au moins aussi éloigné que le numéro de séquence de fonction de décision, dans la direction donnée.

3. Le procédé selon la revendication 1, dans lequel l'opération d'affectation comprend l'opération suivante:

on enregistre le numéro de séquence de système dans des positions de mémoire qui sont associées à chacun des processus.

4. Le procédé selon la revendication 2, dans lequel la version présente de la fonction de programme sélectionnée comprend une version intermédiaire et une version plus ancienne de la fonction de programme sélectionnée, et dans lequel l'opération de branchement vers la version présente si le résultat de la comparaison indique que le numéro de séquence de processus du processus appelant est en deçà du numéro de séquence de fonction de décision dans la direction donnée, comprend en outre les opérations suivantes:

on effectue une seconde comparaison entre le numéro de séquence de processus du processus appelant et un second numéro de séquence de fonction de décision qui est associé à la version intermédiaire de la fonction de programme sélectionnée;

on effectue un branchement vers la version plus ancienne si la seconde comparaison indique que le numéro de séquence de processus du processus appelant est en deçà du second numéro de séquence de fonction de décision, dans la direction donnée;

et on effectue un branchement vers la version intermédiaire si la seconde comparaison indique que le numéro de séquence de processus du processus appelant est au moins aussi éloigné que le second numéro de séquence de fonction de décision dans la direction donnée.

5. Le procédé selon les revendications 1, 2, 3 ou 4, comprenant en outre les opérations suivantes:



périodiquement, on effectue des troisièmes comparaisons entre le numéro de séquence de processus d'au moins tous les processus actifs qui effectuent des appels de fonction dirigés vers la fonction de programme sélectionnée, et le numéro de séquence de fonction de décision, et on supprime la version présente si aucun des numéros de séquence de processus d'au moins tous les processus actifs qui effectuent des appels de fonction dirigés vers la fonction de programme sélectionnée, n'est en deçà du numéro de séquence de fonction de décision dans la direction donnée.

6. Un système de processeur commandé par programme comprenant:
une unité centrale de traitement (202); et
une mémoire de programme (201);
dans lequel des processus sont lancés et exécutés par des appels de fonction dirigés vers des fonctions de programme;

caractérisé en ce que la mémoire de programme (201) enregistre une version présente d'une fonction de programme sélectionnée (132);
enregistre une version modifiée de la fonction de programme sélectionnée (142);

enregistre un numéro de séquence de système (100);

enregistre le numéro de séquence de système en tant que numéro de séquence de processus (111) au moment du lancement de tout processus; et

enregistre un numéro de séquence de fonction de décision (143) pour la version modifiée au moment de l'enregistrement de la version modifiée, la valeur du numéro de séquence de fonction de décision étant au-delà de la valeur présente de tout numéro de séquence de processus, dans une direction donnée;

lorsqu'un numéro de séquence de fonction de décision est enregistré, l'unité centrale de traitement (202) modifie le numéro de séquence de système de façon qu'il atteigne une valeur au moins aussi éloignée dans la direction donnée que le numéro de séquence de fonction de décision;

l'unité centrale de traitement (202) interroge la mémoire de programme (201, 110, 111, 121, 141, 132, 142) et exécute la version présente (132) de la fonction sélectionnée, en réponse à un appel de fonction qu'un processus appelant émet vers la fonction sélectionnée, si le numéro de séquence de processus du processus appelant est en deçà

du numéro de séquence de fonction de décision dans la direction donnée, et elle exécute la version modifiée (142) de la fonction sélectionnée en réponse à un appel de fonction que le processus appelant émet vers la fonction sélectionnée, si le numéro de séquence de processus du processus appelant est au moins aussi éloigné que le numéro de fonction de décision, dans la direction donnée.

7. Un système de processeur commandé par programme selon la revendication 6, caractérisé en ce que l'unité centrale de traitement (202) compare le numéro de séquence de processus du processus appelant avec le numéro de séquence de fonction de décision et elle effectue un branchement vers la version présente, si la comparaison indique que le numéro de séquence de processus du processus appelant est en deçà du numéro de séquence de fonction de décision dans la direction donnée, et elle effectue un branchement vers la version modifiée si la comparaison indique que le numéro de séquence de processus du processus appelant est au moins aussi éloigné que le numéro de séquence de fonction de décision, dans la direction donnée.

8. Un système de processeur commandé par programme selon la revendication 7, caractérisé en ce que la mémoire de programme (201) enregistre une version intermédiaire de la fonction de programme sélectionnée et une version plus ancienne, à la place de la version présente, l'unité centrale de traitement (200) se branche vers la version présente si la comparaison indique que le numéro de séquence de processus du processus appelant est en deçà du numéro de séquence de fonction de décision dans la direction donnée, et elle effectue une seconde comparaison entre le numéro de séquence de processus du processus appelant et un second numéro de séquence de fonction de décision qui est associé à la version intermédiaire de la fonction de programme sélectionnée, et elle se branche vers la version plus ancienne si la seconde comparaison indique que le numéro de séquence de processus du processus appelant est en deçà du second numéro de séquence de fonction de décision dans la direction donnée, tandis qu'elle se branche vers la version intermédiaire si la seconde comparaison indique que le numéro de séquence de processus est au moins aussi éloigné que le second numéro de fonction de décision dans la direction donnée

