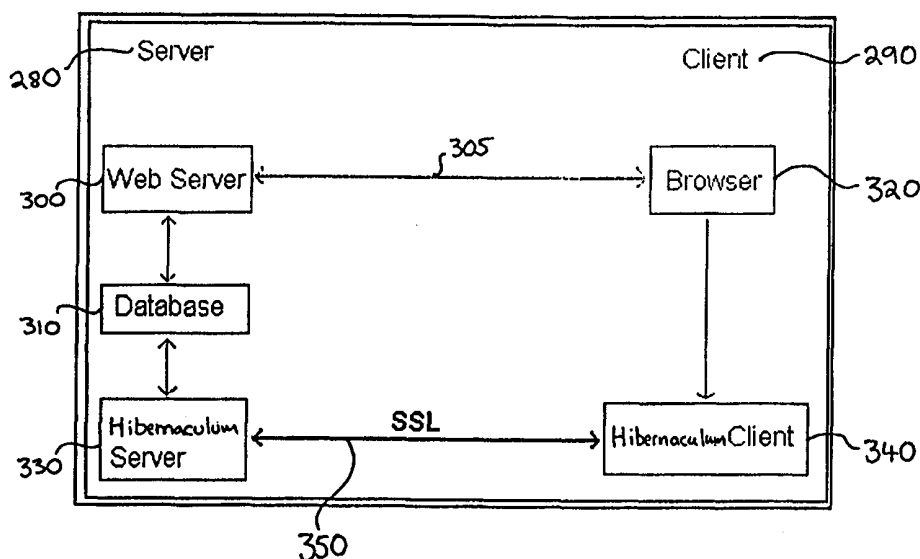




INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification ⁷ : G06F 17/14	A1	(11) International Publication Number: WO 00/36526 (43) International Publication Date: 22 June 2000 (22.06.00)
(21) International Application Number: PCT/SG99/00024 (22) International Filing Date: 26 February 1999 (26.02.99) (30) Priority Data: PCT/SG98/00102 16 December 1998 (16.12.98) SG (71) Applicant (for all designated States except US): KENT RIDGE DIGITAL LABS [SG/SG]; 21 Heng Mui Keng Terrace, Singapore 119613 (SG). (72) Inventors; and (75) Inventors/Applicants (for US only): NGAIR, Teow, Hin [SG/SG]; 334 Kang Ching Road #13-254, Singapore 610334 (SG). PANG, Hwee, Hwa [SG/SG]; 19 Shelford Road #01-42, Singapore 288408 (SG). (74) Agent: GREENE-KELLY, James, Patrick; Lloyd Wise, Tanjong Pagar, P.O. Box 636, Singapore 910816 (SG).		(81) Designated States: JP, SG, US, European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE). Published <i>With international search report.</i>

(54) Title: A METHOD OF PROCESSING DIGITAL MATERIAL**(57) Abstract**

A method of processing digital material comprising program code and data is disclosed, the method comprising the steps of: running the program code on a computing device until a predetermined execution point is reached, and execution state existing at the execution point; and forming a combined representation of the execution state, data and program code at that execution point, the data, code and execution state being restorable from said representation so that execution of the material may subsequently be resumed from the execution point on a remote computing device. Preferably, prior to the execution point, customization information is provided whereby the representation is customized by the information. The information may include information identifying an intended user of the material, information identifying an intended machine on which the material is subsequently to be run, information specifying the number of times the digital material may be run and/or information configuring the digital material. Particular applications for the sale of video and audio files on demand are disclosed.

FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AL	Albania	ES	Spain	LS	Lesotho	SI	Slovenia
AM	Armenia	FI	Finland	LT	Lithuania	SK	Slovakia
AT	Austria	FR	France	LU	Luxembourg	SN	Senegal
AU	Australia	GA	Gabon	LV	Latvia	SZ	Swaziland
AZ	Azerbaijan	GB	United Kingdom	MC	Monaco	TD	Chad
BA	Bosnia and Herzegovina	GE	Georgia	MD	Republic of Moldova	TG	Togo
BB	Barbados	GH	Ghana	MG	Madagascar	TJ	Tajikistan
BE	Belgium	GN	Guinea	MK	The former Yugoslav Republic of Macedonia	TM	Turkmenistan
BF	Burkina Faso	GR	Greece			TR	Turkey
BG	Bulgaria	HU	Hungary	ML	Mali	TT	Trinidad and Tobago
BJ	Benin	IE	Ireland	MN	Mongolia	UA	Ukraine
BR	Brazil	IL	Israel	MR	Mauritania	UG	Uganda
BY	Belarus	IS	Iceland	MW	Malawi	US	United States of America
CA	Canada	IT	Italy	MX	Mexico	UZ	Uzbekistan
CF	Central African Republic	JP	Japan	NE	Niger	VN	Viet Nam
CG	Congo	KE	Kenya	NL	Netherlands	YU	Yugoslavia
CH	Switzerland	KG	Kyrgyzstan	NO	Norway	ZW	Zimbabwe
CI	Côte d'Ivoire	KP	Democratic People's Republic of Korea	NZ	New Zealand		
CM	Cameroon			PL	Poland		
CN	China	KR	Republic of Korea	PT	Portugal		
CU	Cuba	KZ	Kazakstan	RO	Romania		
CZ	Czech Republic	LC	Saint Lucia	RU	Russian Federation		
DE	Germany	LI	Liechtenstein	SD	Sudan		
DK	Denmark	LK	Sri Lanka	SE	Sweden		
EE	Estonia	LR	Liberia	SG	Singapore		

A METHOD OF PROCESSING DIGITAL MATERIALBACKGROUND AND FIELD OF THE INVENTION

5

This invention relates to a method of processing digital material, more particularly but not exclusively to a process which allows the generation and distribution of customized digital material.

10 The recent exponential growth of Internet usage has provided a corresponding opportunity for selling digital goods through this medium. Such digital goods include application programs, images, videos, audio files and other digitized IPR (Intellectual property rights) material. However, due to security issues, large IPR owners are hesitant to embrace the Internet for selling their products. The ease by which software
15 may be duplicated, together with the access to such goods in digital form that transmission over the Internet requires and which sale of legitimate digital goods would initially provide, makes the subsequent distribution of counterfeit digital goods much easier.

20 This problem has given rise to active research in the field of digital watermarking. Such technology embeds licence and owner information into the digital goods which can help to trace the source of counterfeit goods in the event that duplicated copies of, some digital IPR materials are found. Another method is to include access control information within the digital goods to restrict its usage. One particular problem with

such techniques is that the licence information needs somehow to be applied to the digital goods in a cost-effective and secure way. If the information is embedded in the program code of the digital goods, this will require the program to be recompiled every time, so that the new licence and owner information can be included in the machine
5 code, which may be prohibitively costly. Alternatively, if the information is embedded in the data of the digital goods, the information may be vulnerable, since the program code that is used to access the data can be rewritten to ignore that part of the data and thus bypass the access control information.

10 It is an object of the invention to provide a novel method of processing digital material which allows such information to be incorporated into digital goods.

SUMMARY OF THE INVENTION

15 According to the invention in a first aspect there is provided a representation of digital material, the digital material comprising program code and data and the representation comprising a combination of the code, data and an execution state existing at a predetermined execution point when the program code is run, customization
information being provided during running of the program code prior to the execution
20 point being reached so that said representation is customized by said customization information.

According to the invention in a second aspect, there is provided a method of processing digital material, the material comprising program code and data, the method

comprising the steps of: (1) running the program code until a predetermined execution point is reached, an execution state existing at the execution point; and (2) forming a combined representation of the execution state, data and program code at that execution point, the data, code and execution state being restorable from said representation so
5 that execution of the material may subsequently be resumed from the execution point.

Further features of the invention are recited in the claims appendant hereto.

The term "execution state" means the values and contents of transient parameters such
10 as the contents of registers, context frames, counters and the like.

In the described embodiment of the invention, a method is disclosed that provides a flexible mechanism to prevent the possibility of digital material transmitted over a communication network, in particular digital goods to be marketed, from being
15 duplicated. Using such a method, any type of digital material can be transmitted by hiding the information as part of a single representation termed a computing process in which data, program code and execution state are combined. Such computing processes can be structured by the IPR providers to include necessary anti-copying mechanisms and standard security protection like channel privacy, user and machine
20 authentication, etc. Since the computing processes could be made unique to each IPR provider and changed frequently independent of the information included in them, it could render any duplication effort very difficult and costly.

Specific examples of applications of this method disclosed are the protection of video,

audio and other multimedia material supplied over the Internet or electronically.

BRIEF DESCRIPTION OF THE DRAWINGS

5

Some embodiments of the present invention will now be described by way of example and with reference to the accompanying drawings, in which:

Fig.1 is a general schematic model of an operating environment of a computing system ;

10 Fig.2 schematically illustrates a process life-cycle and operations that may be performed on a process,

Fig.3 is a flow-chart illustrating the operation Hibernaculum Construct(Stack s, Module m, Data d),

15 Fig.4 is a flow-chart illustrating operation int Assimilate(Hibernaculum h, Override Flags f),

Fig.5 is a flow-chart illustrating the operation int Usurp(Hibernaculum h, Override Flags f),

Fig.6 is a flow-chart illustrating the operation int Bequeath(Hibernaculum h, Thread threadname, Override Flags f),

20 Fig.7 is a flow-chart illustrating the operation int Inherit(Hibernaculum h, Thread threadname, Override Flags f),

Fig.8 is a flow-chart illustrating the operation int Mutate(Stacks, int sFlag, Module m, int mFlag, Data d, int dFlag),

Fig.9 is a flow-chart illustrating the operation int Checkpoint(Process p1, Target

t),

Fig.10 is a flow-chart illustrating the operation int Migrate(Process p1, Machine m),

Fig. 11 is a schematic diagram of a specific embodiment of the invention for
5 the sale of videos on-line,

Fig. 12 is a view of a login screen of the embodiment of Fig. 11,

Fig. 13 is a view of a video selection screen of the embodiment of Fig. 11, and

Fig. 14 is a view of a download screen of the embodiment of Fig. 11.

10 DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

The embodiments of the invention to be described rely on a representation of digital material being generated, termed a process, which comprises a combination of data, program code and execution state. This general technique will now first described,
15 following which, embodiments of the invention, being specific applications of the technique will be described.

Figure 1 shows the general model of a computing system. An application program 30 comprises data 10 and program modules 20. The operating system 60, also known as
20 the virtual machine, executes the application 30 by carrying out the instructions in the program modules 20, which might cause the data 10 to be changed. The execution is effected by controlling the hardware of the underlying machine 70. The status of the execution, together with the data and results that the operating system 60 maintains for the application 30, form its execution state 40.

Such a model is general to any computing system. It should be noted here that the present invention starts from the realisation that all the information pertaining to the application at any time is completely captured by the data 10, program modules 20 and execution state 40, known collectively as the process 50 of the application 30.

5

The process 50 can have one or more threads of execution at the same time. Each thread executes the code of a single program module at any given time. Associated with the thread is a current context frame, which includes the following components:

- 10 1. A set of registers;
2. A program counter, which contains the address of the next instruction to be executed;
3. Local variables of the module;
4. Input and output parameters of the module;
- 15 5. Temporary results of the module.

In any module A, the thread could encounter an instruction to invoke another module B. In response, the program counter in the current frame is incremented, then a new context frame is created for the thread before it switches to executing module B. Upon
20 completing module B, the new context frame is discarded. Following that, the thread reverts to the previous frame, and resumes execution of the original module A at the instruction indicated by the program counter, i.e., the instruction immediately after the module invocation. Since module B could invoke another module, which in turn could invoke some other module and so on, the number of frames belonging to a thread may

grow and reduce with module invocations and completions. However, the current frame of a thread at any given time is always the one that was created last. For this reason, the context frames of a thread are typically stored in a stack with new frames being pushed on and popped from the top. The context frames of a thread form its execution state, and the state of all the threads within the process 50 constitute its execution state 40 in Fig.1.

The data 10 and program modules 20 are shared among all threads. The data area is preferably implemented as a heap, though this is not essential. The locations of the data 10 and program modules 20 are summarized in a symbol table. Each entry in the 10 table gives the name of a datum or a program module, its starting location in the address space, its size, and possibly other descriptors. Instead of having a single symbol table, each process may alternatively maintain two symbol tables, one for data alone and the other for program modules only, or the process could maintain no symbol table at all.

15

In a preferred embodiment of the present invention, the data and program code of a process are stored in a heap and a program area respectively and are shared by all the threads within the process. In addition the execution state of the process comprises a stack for each thread, each stack holding context frames, in turn each frame containing 20 the registers, local variables and temporary results of a program module, as well as addresses for further module invocations and returns. Before describing an embodiment of the invention in more detail, however, it is first necessary to introduce some definitions of data types and functions that are used in the embodiment and which will be referred to further below.

In addition to conventional data types such as integers and pointer, four new data types Data, Module, Stack and Hibernaculum are defined in the present invention:

- 5 **Data:** A variable of this data type holds a set of data references. Members are added to and removed from the set by means of the following functions;

 Int AddDatum(Data d, String dataname) inserts the data item dataname in the heap of the process as a member of d.

- 10 Int DelDatum(Data d, String dataname) removes the data item dataname from d

Module: A variable of this data type holds a set of references to program modules. Members are added to and removed from the set with the following functions;

- 15 Int AddModule(Module d, String modulename) inserts the program module modulename in the program area of the process as a member of d.

 Int DelModule(Module d, Stringname modulename) removes the program module modulename from d.

20

Stack: A variable of this data type holds a list of ranges of execution frames from the stack of the threads. The list may contain frame ranges from multiple threads, however no thread can have more than one range. Variables of this type are manipulated by the following functions:

Int OpenFrame(Stack d, Thread threadname) inserts into d a new range for the thread threadname, beginning with the thread s current execution frame. This function has no effect if the thread already has a range in d.

- 5 Int CloseFrame(Stack d, Thread threadname) ends the open-ended range in d that belongs to the thread threadname. This function has no effect if the thread does not currently have an open-ended range in d.

- Int PopRange(Stack d, Thread threadname) removes from d the range belonging
10 to the thread threadname.

Hibernaculum: A variable of this data type is used to hold a suspended process.

- As will be explained in more detail below a process may be suspended and
15 stored in a construct prior to being transferred from one operating environment to another operating environment and/or may be subject to evolutionary operations:

- Hibernaculum Construct(Stack s, Module m, Data d): This operation creates a new process with the execution state, program table and data heap specified as input
20 parameters. The process is immediately suspended and then returned in a hibernaculum. The hibernaculum may be signed by the originating process as indication of its authenticity. Fig.3 is a flow-chart showing the hibernaculum construct operation.

A hibernaculum may be sent between operating environments by the following send and receive functions:

Int Send(Hibernaculum h, Target t) transmits the process contained within h to the
5 specified target.

Hibernaculum Receive(Source s) receives from the specified source a hibernaculum containing a process.

10 A hibernaculum may be subject to the following evolutionary functions:

Int Assimilate(Hibernaculum h, OverrideFlags f) activates the threads of the process stored within h and runs them as threads within a calling process s operating environment. Where there is a conflict between the data and/or program modules of
15 the hibernaculum and the operating environment, the override flags specify which to preserve. Fig.4 is a flow-chart illustrating the steps of the assimilate operation.

Int Usurp(Hibernaculum h, OverrideFlags f) copies the data and program modules of the process within h into the calling process s operating environment. Where there is
20 a conflict between the data and/or program modules of the hibernaculum and the operating environment, the override flags specify which to preserve. Fig.5 is a flow-chart illustrating the steps of the usurp operation.

Int Bequeath(Hibernaculum h, Thread threadname, OverrideFlags f). upon threadname

s termination, activates the threads of the process stored within h and runs them as threads within the environment of the process containing threadname. Where there is a conflict between the data and/or program modules of the hibernaculum and the calling process, the override flags specify which is to be preserved. Fig.6 is a
5 flow-chart illustrating the steps of the bequeath operation.

Int Inherit(Hibernaculum h, Thread threadname, OverrideFlags f) suspends threadname and activates the process within h. When the process within h terminates its data and program modules are added to the process containing threadname before threadname
10 is reactivated. Where there is a conflict between the data and/or program modules of the hibernaculum and the process containing threadname, the override flags specify which is to be preserved. Fig.7 is a flow-chart illustrating the steps of the inherit operation.

15 Int Mutate(Stack s, int sFlag, Module m, int mflag, Data d, int dflag) modifies the execution state, program table and data heap of the calling process. If a thread has an entry in s, only the range of execution frames specified by this entry is preserved, the other frames are discarded. Execution stacks belonging to threads without an entry in s are left untouched. In addition, program modules listed in m and data items listed in
20 d are kept or discarded depending on the flag status. Fig.8 is a flow-chart illustrating the steps of the mutate operation.

Int Checkpoint(Process p, Target t) creates a snapshot of p including all of its data, program modules and execution state. The snapshot is then sent to the specified target.

Int Migrate(Process p, Machine m) transfers the process p to the specified machine for p to continue execution there. All of the data (including file handles and established sockets), program modules and execution state are preserved in the transfer.

- 5 The typical life cycle of a process is depicted in Figure 2. First, an application 210 or a suspended process 220 stored in a hibernaculum is loaded by the operating system, then starts running as a process 230. The application 210 may be a program that is designed to perform some tasks, or it may be a simple loader that is used to start up other processes, while the suspended process 220 could be either produced locally
10 earlier or generated on and transferred from another machine. As the process 230 goes through the application program logic, the process may create new processes, absorb other processes, mutate, or migrate as explained below.

In particular, the process may be stored in a construct hibernaculum in a suspended
15 state. In this condition the process may be subject to a number of operations. To begin with the hibernaculum may be sent to another operating environment. Alternatively the process may be modified within its own operating environment by the selective deletion of elements from within the process, or by the selective reloading of elements into the process. A still further possibility is that the process may receive a suspended
20 process from another operating environment, and either all or part of this suspended process may be incorporated into the first process. Alternatively all or part of the first process may be incorporated into the second process. It will also be understood that all these operations may be combined in many different ways. For example, a process may be sent from one operating environment to another and then may mutate by

dropping certain elements and reloading other elements when in the second environment. In the following description the construction of a hibernaculum, send and receive functions, and exemplary evolutionary operations will all be described in greater detail.

5

New processes are created with the Construct 110 operation. Fig.3 is a flow-chart showing the steps of this Construct operation. Each invocation of this operation starts up a controller thread in the process 230. The controller thread freezes all other active threads in the process 230, then creates a new process with some or all of the
10 execution state, program modules and/or data of the process 230 except for those belonging to the controller thread, before resuming the frozen threads. Therefore, the new process contains no trace of the controller thread. The new process is suspended immediately and returned in a hibernaculum in the data area of the process 230. As explained earlier, a hibernaculum is a special data type that serves the sole purpose of
15 providing a container for a suspended process. Since a process may have several hibernacula in its data area, it could create a new hibernaculum that contains those hibernacula, each of which in turn could contain more hibernacula, and so on. When the new process is activated subsequently, only those threads that were active just before the Construct 110 operation will begin to execute initially: threads that were
20 suspended at that time will remain suspended. At the end of the Construct 110 operation, the controller thread resumes those threads that were frozen by it before terminating itself.

To specify what execution state should go into the new process, the Construct 110

operation is passed a list of ranges of context frames. The list may include frames from the state of several threads. No thread is allowed to have more than one range in this list. A thread can specify that all of its frames at the time of the Construct 110 operation are to be included in the list, by calling the AllFrame function beforehand.

5 Alternatively, the thread can call the OpenFrame function to register its current frame, so that all the frames from the registered frame to the current frame at the time of the Construct 110 operation are included in the list. The thread can also call the CloseFrame function subsequently, to indicate that only frames from that registered with OpenFrame to the current frame at the time of calling CloseFrame are to be

10 included in the list for the Construct 110 operation. An AllFrame or OpenFrame request for a thread erases any previous AllFrame, OpenFrame and CloseFrame requests for that thread. A CloseFrame request overrides any earlier CloseFrame request for the same thread, but the request is invalid if the thread has not already had an OpenFrame request. A thread can also make AllFrame, OpenFrame and/or

15 CloseFrame requests on behalf of another thread by providing the identity of that thread in the requests; the effect is as if that thread is making those requests itself.

The Construct 110 operation can also be passed a list of program modules that should go into the newly created process. A thread can specify that all modules of the process

20 230 are to be included in the list, by calling the AllModules function prior to the Construct 110 operation. Alternatively, the thread can call the AddModule function to add a specific module to the list, and the DelModule function to remove a specific module from the list. The effect of the AllModules, AddModule and DelModule requests, possibly made by different threads, are cumulative. Hence a DelModule

request after an AllModules request would leave every module in the list except for the one removed explicitly, and a DelModule can be negated with an AddModule or AllModules request. As there could be multiple AddModule requests for the same module and AllModules could be called multiple times, a program module may be
5 referenced several times in the list. However, the Construct 110 operation consolidates the entries in the list, so no program module gets duplicated in the new process.

To copy some or all of the data of the process 230 to the new process, the Construct 110 operation can be passed a data list. This list contains only the name of, or
10 reference to data that should be copied. The actual data content or values that get copied to the new process are taken at the time of the Construct 110 operation, not at the time that each datum is added to the list. To ensure consistency among data that could be related to each other, all the threads in the process 230 are frozen during the Construct 110 operation. A thread can specify that all data of the process 230 are to
15 be included in the list, by calling the AllData function prior to the Construct 110 operation. Alternatively, the thread can call the AddDatum function to add a specific datum to the list, and the DelDatum function to remove a specific datum from the list. The effect of the AllData, AddDatum and DelDatum requests, possibly made by different threads, are cumulative. Hence a DelDatum request after an AllData request
20 would leave all of the data in the list except for the one removed explicitly, and a DelDatum can be negated with an AddDatum or AllData request. As there could be multiple AddDatum requests for the same datum and AllData could be called multiple times, a datum may be referenced several times in the list. However, the Construct 110 operation consolidates the entries in the list, so no datum gets duplicated in the new

process.

Since the lists passed to the Construct 110 operation are constructed from the execution state, program modules and data of the process 230, the new process initially does not
5 contain any component that is not found in the process 230. Consequently, the symbol table in the new process is a subset of the symbol table of the process 230. Threads in the process 230 that do not have any frame in the new process are effectively dropped from it. For those threads that have frames in the new process, when activated later, each will begin execution at the instruction indicated by the program counter in
10 the most recent frame amongst its frames that are copied. By excluding one or more of the most recent frames from the new process, the associated thread can be forced to return from the most recent module invocations. An exception is raised to alert the thread that those modules are not completed normally. Alternatively, the thread can be made to redo those modules upon activation, by decrementing the program counter in
15 the most recent frame amongst those frames belonging to that thread that are copied. Similarly, by excluding one or more of its oldest frames from the new process, a thread can be forced to terminate directly after completing the frames that are included.

Hibernacula, produced by the Construct 110 operation, can be exchanged between
20 processes via the Send 120 and Receive 130 operations. The process 230 can send a hibernaculum in its data area out on an output stream, by invoking the Send 120 operation with the hibernaculum and output stream as parameters. The output stream could lead to either a disk file, a memory stream, a device stream, or the input stream of another process, possibly on a different machine. In the former case, the process in

the hibernaculum is stored away for the time being, whereas in the latter case the process in the hibernaculum is received directly into the data area of another process.

The process 230 can also acquire other processes via the Receive 130 operation, which
5 receives from an input stream a hibernaculum containing a suspended process. The source of the input stream could be a disk file, a memory stream, a device stream, or the output stream of another process, possibly from a different machine.

The process 230 can absorb the suspended process within a hibernaculum through the
10 Assimilate 140, Usurp 150, Bequeath 160 and Inherit 170 operations. These operations can be invoked by any thread in the process 230, or by another process that has appropriate permissions. The hibernaculum could have been created by the process 230 itself earlier, or received from another process via a disk file or an input stream.

15 The Assimilate 140 operation accepts a hibernaculum as input. Fig.4 is a flow-chart showing this operation in detail. The operation starts up a controller thread in the process 230. The controller thread freezes all other active threads in the process 230, adds the program modules and data of the process in the hibernaculum to the program modules and data of the process 230, respectively, and updates its symbol table
20 accordingly. In case of a name conflict, the program module or data from the hibernaculum is discarded in favor of the one from the process 230 by default. However, a flag can be supplied to give preference to the hibernaculum. Moreover, the context frames of the threads within the hibernaculum are added to the execution state of the process 230, enabling those threads to run within the process. Only those threads

that were active just before the hibernaculum was created are activated initially; threads that were suspended at that time remain suspended. Each newly acquired thread begins execution at the instruction indicated by the program counter in the most recent frame amongst the context frames that belong to that thread. Finally, all the original threads
5 in the process 230 that were frozen by the controller thread are resumed before it terminates itself.

The Usurp 150 operation, which also accepts a hibernaculum as input, enables the process 230 to take in only program modules and data from a hibernaculum, without
10 acquiring its threads. Fig.5 is a flow-chart showing this operation in detail. The operation starts up a controller thread in the process 230. The controller thread freezes all other active threads in the process 230, adds the program modules and data of the process in the hibernaculum to the program modules and data of the process 230, respectively, and updates its symbol table accordingly. In case of a name conflict, the
15 program module or data from the hibernaculum is discarded in favor of the one from the process 230 by default. However, a flag can be supplied to give preference to the hibernaculum. Finally, all the original threads in the process 230 that were frozen by the controller thread are resumed before it terminates itself.

20 The Bequeath 160 operation accepts a hibernaculum and a thread reference as input. Fig.6 is a flow-chart showing this operation in detail. It starts up a bequeath-thread in the process 230. The bequeath-thread registers the referenced thread and any existing bequeath-threads on that thread, then allows them to carry on execution without change. After all those threads and threads that they in turn activate have terminated,

the bequeath-thread loads in the context frames, program modules and data in the hibernaculum. In case of a name conflict, the program module or data from the hibernaculum is discarded in favor of the one from the process 230 by default. However, a flag can be supplied to give preference to the hibernaculum. Subsequently,

5 threads within the hibernaculum are activated to run in the process 230 before the bequeath-thread terminates itself. Only those threads that were active just before the hibernaculum was created are activated initially; threads that were suspended at that time remain suspended. Each newly acquired thread begins execution at the instruction indicated by the program counter in the most recent frame amongst the context frames

10 that belong to that thread. If multiple Bequeath 160 requests are issued for the same thread, there will be several bequeath-threads in the process 230. Each bequeath-thread will wait for all the existing bequeath-threads on the same thread, together with all the threads that they activate, to terminate before performing its function. As a result, the Bequeath 160 requests are queued up and serviced in chronological order.

15

The reverse of Bequeath 160 is the Inherit 170 operation, which also accepts a hibernaculum and a thread reference as input. The flow-chart for this operation is shown in Fig.7. This operation starts up an inherit-thread in the process 230. The inherit-thread freezes the referenced thread in the process 230 together with the

20 bequeath-threads and any inherit-thread for the referenced thread, adds the program modules and data in the hibernaculum to the program modules and data of the process 230, respectively, and updates its symbol table accordingly. In case of a name conflict, the program module or data from the hibernaculum is discarded in favor of the one from the process 230 by default. However, a flag can be supplied to give preference

to the hibernaculum. Moreover, the context frames of the threads within the hibernaculum are added to the execution state of the process 230, enabling those threads to run within the process. Only those threads that were active just before the hibernaculum was created are activated initially; threads that were suspended at that
5 time remain suspended. Each newly acquired thread begins execution at the instruction indicated by the program counter in the most recent frame amongst the context frames that belong to that thread. After all the acquired threads and threads that they in turn activate have terminated, the inherit-thread resumes those threads that were frozen by it earlier, before terminating itself. If one of the acquired threads issues an Inherit 170
10 request, the acquired threads and the current inherit-thread would in turn be suspended, pending the completion of the latest Inherit operation. If an acquired thread does a Bequeath 160, the inherit-thread would wait for the Bequeath request to be satisfied before resuming the threads that were frozen by it.

15 Besides constructing and absorbing new processes, the process 230 can also modify any of its components directly by calling the Mutate 180 operation from any thread. Fig.8 shows the flow-chart for the Mutate operation. The operation starts up a controller thread in the process 230. The controller thread first freezes all other active threads in the process 230, then selectively retains or discards its execution state,
20 program modules and data. A list of context frames, together with a flag, can be passed to the Mutate 180 operation to retain or discard the frames in the list. Similarly, a program module list and/or a data list can be provided to indicate program modules and data of the process 230 that should be retained or discarded. The generation of the execution state, program module and data lists are the same as for the Construct 110

operation. After mutation, the controller thread resumes the threads that were frozen by it before terminating itself. Threads that no longer have a context frame in the process 230 are terminated. Each of the remaining threads resumes execution at the instruction indicated by the program counter in the most recent frame amongst the retained frames belonging to that thread. By discarding one or more of its most recent frames, a thread can be forced to return from the most recent module invocations. An exception is raised to alert the thread that those modules are not completed normally. Similarly, by discarding one or more of its oldest frames, a thread can be forced to terminate directly after completing the frames that are retained. Space freed up from the discarded context frames, program modules and data is automatically reclaimed by a garbage collector.

The process 230 is also capable of suspending and migrating itself. The Checkpoint 190 operation starts up a controller thread in the process 230. Fig.9 is a flow-chart showing the Checkpoint operation in detail. The controller thread freezes all other active threads in the process 230, then sends it in the form of a hibernaculum on a specified output stream that it established earlier. The hibernaculum contains all of the execution state, program modules, and data of the process 230, except for those belonging to the controller thread and the output stream used by the Checkpoint 190 operation. The output stream could lead to either a disk file or the input stream of another process, possibly on a different machine. The controller thread ends by terminating the process 230. When the process in the hibernaculum is activated subsequently, only those threads that were active just before the Checkpoint 190 operation will begin to execute initially; threads that were suspended at that time will

remain suspended. The Checkpoint 190 operation could be invoked by one of the threads of the process 230, or by another process with appropriate permissions.

The Migrate 200 operation is used to move the process 230 to another machine. Fig. 10
5 is a flow-chart showing the Migrate operation in detail. The operation can be invoked by one of the threads of the process 230, or by another process with appropriate permissions. The Migrate 200 operation starts up a first controller thread in the process 230, which carries out the following steps: (a) It freezes all other active threads in the process 230. (b) It initiates a receiver process on the target machine, which runs a
10 second controller thread. (c) The two controller threads establish an output stream in the process 230 that leads to an input stream in the receiver process. (d) The second controller thread in the receiver process performs a Receive 130 operation on its input stream. (e) The first controller thread sends the process 230 in the form of a hibernaculum on the output stream. The hibernaculum contains all of the execution
15 state, program modules, and data of the process 230 except for those belonging to the controller thread and the output stream used for the migration. (f) The first controller thread ends by terminating the process 230. (g) After receiving the hibernaculum, the second controller thread activates the process in the hibernaculum before terminating itself. Only those threads that were active just before the migration are activated
20 initially; threads that were suspended at that time remain suspended.

To implement the process migration and adaptation system of the present invention in a Java environment, a package called snapshot is introduced. This package contains the following classes, each of which defines a data structure that is used in the migration

and adaptation operations:

```
public class Hibernaculum {  
    ...  
5    }  
  
public class State {  
    ...  
    }  
10  
public class Module {  
    ...  
    }  
  
15 public class Data {  
    ...  
    }  
  
public class Machine {  
20    ...  
    }
```

In addition, the package contains a Snapshot class that defines the migration and adaptation operations:

```
public class Snapshot {  
    private static native void registerNatives();  
    static {  
        registerNatives();  
5    }  
  
    public static native Hibernaculum Construct(State s, Module m, Data d);  
    public static native int Send(Hibernaculum h, OutputStream o);  
    public static native Hibernaculum Receive(InputStream i);  
10    public static native int Assimilate(Hibernaculum h, int f);  
    public static native int Usurp(Hibernaculum h, int f);  
    public static native int Bequeath(Hibernaculum h, int f);  
    public static native int Inherit(Hibernaculum h, int f);  
    public static native int Mutate(State s, Module m, int mflag, Data d, int dflag);  
15    public static native int Checkpoint(OutputStream o);  
    public static native int Migrate(Machine m);  
  
    // This class is not to be instantiated  
    private Snapshot() {  
20    }  
}
```

The methods in the Snapshot class can be invoked from application code. For example:

```

    try {
        if (snapshot.Snapshot.Construct(s, m, d) != null) {
            // hibernaculum has been created
        } else {
5           // failed to create hibernaculum
        }
        catch(snapshot.SnapshotException e) {
            // Failed to create hibernaculum
        }
10

```

The migration and adaptation operations are implemented as native codes that are added to the Java virtual machine itself, using the Java Native Interface (JNI). To do that, a Java-to-native table is first defined:

```

15  #define KSH  Ljava/snapshot/Hibernaculum;
    #define KSS  Ljava/snapshot/State;
    #define KSM  Ljava/snapshot/Module;
    #define KSD  Ljava/snapshot/Data;

20  static JNINativeMethod snapshot_Snapshot_native_methods[] = {
    {
        Construct ,
        ( KSSKSMKSD ) KSH,
        (void*)Impl_Snapshot_Construct
    }

```

```
    },  
    {  
        Send ,  
        ( KSH Ljava/io/OutputStream;)I ,  
5        (void*)Impl_Snapshot_Send  
    },  
    {  
        Receive ,  
        (Ljava/io/InputStream;) KSH,  
10        (void*)Impl_Snapshot_Receive  
    },  
    {  
        Assimilate ,  
        ( KSH I)I ,  
15        (void*)Impl_Snapshot_Assimilate  
    },  
    {  
        Usurp ,  
        ( KSH I)I ,  
20        (void*)Impl_Snapshot_Usurp  
    },  
    {  
        Bequeath ,  
        ( KSH I)I ,
```

```
(void*)Impl_Snapshot_Bequeath
    },
    {
        Inherit ,
5        ( KSH I)I ,
        (void*)Impl_Snapshot_Inherit
    },
    {
        Mutate ,
10        ( KSSKSM I KSD I)I ,
        (void*)Impl_Snapshot_Mutate
    },
    {
        Checkpoint ,
15        (Ljava/io/OutputStream;)I ,
        (void*)Impl_Snapshot_Checkpoint
    },
    {
        Migrate ,
20        (Ljava/snapshot/Machine;)I ,
        (void*)Impl_Snapshot_Migrate
    },
};
```

After that, the native implementations are registered via the following function:

```
JNIEXPORT void JNICALL
Java_snapshot_Snapshot_registerNatives(JNIEnv *env, jclass cls) {
5      (*env)->RegisterNatives(    env,
                                   cls,
                                   snapshot_Snapshot_native_methods,
                                   sizeof(snapshot_Snapshot_native_methods) /
                                   sizeof(JNINativeMethod)    );
10 }
```

Besides the above native codes, several functions are added to the Java virtual machine implementation, each of which realizes one of the migration and adaptation operations:

```
15
void* Impl_Snapshot_Construct(..) {
    // follow flowchart in Figure 3

}

20
void* Impl_Snapshot_Send(..) {
    // send given hibernaculum to specified target

}
```

```
void* Impl_Snapshot_Receive(..) {  
    // receive a hibernaculum from a specified source  
  
}
```

5

```
void* Impl_Snapshot_Assimilate(..) {  
    // follow flowchart in Figure 4  
  
}
```

10

```
void* Impl_Snapshot_Usurp(..) {  
    // follow flowchart in Figure 5  
  
}
```

15

```
void* Impl_Snapshot_Bequeath(..) {  
    // follow flowchart in Figure 6  
  
}
```

20

```
void* Impl_Snapshot_Inherit(..) {  
    // follow flowchart in Figure 7  
  
}
```

```
void* Impl_Snapshot_Mutate(..) {  
    // follow flowchart in Figure 8  
  
}
```

5

```
void* Impl_Snapshot_Checkpoint(..) {  
    // follow flowchart in Figure 9  
  
}
```

10

```
void* Impl_Snapshot_Migrate(..) {  
    // follow flowchart in Figure 10  
  
}
```

15

As will be apparent from the above, using the computer environment described, it is possible using the Construct 110 operation to create a hibernaculum of a process in which the data, program modules and execution state at a particular execution point when running on a first machine are suspended and stored. That process can then be transferred as an output stream to another location using the Send 120 operation, for example a disk file or can be transmitted to a second machine, for example at a remote location, for execution to be resumed either as a self-standing process or as input to another process running on the second machine using any one of the Assimilate 140, Usurp 150, Bequeath 160 or Inherit 170 operations..

This feature allows a customized version of software and other digital material to be provided by running the software to a particular execution point, prior to which customized features are incorporated, generating the suspended process, which incorporates the customization, using Construct 110 or Checkpoint 190 and then
5 supplying the process to a customer as a bespoke product for the customer to use, as will now be explained.

Essentially, software is written to run in two phases, a customization phase and an execution phase.

10

Customization Phase: In the customization phase, the software initializes, loads the program modules into memory and then takes in a configuration specification, either through user input or from a configuration file. The specification can include use control information, watermarks, user's machine identification, a user's
15 cryptographic token, licence number, software features to enable/disable features, functions or commands (for example to permit saving of data into files, to allow the software to run only a limited number of times or to run only on a specific machine) and content to be supplied (eg an abridged or expanded version of the software). The software determines from the specification which data files and any further program
20 modules that might be needed for the configuration are required and loads these into memory. Similarly, any program modules and data not required can be discarded using the Mutate 180 operation. A hibernaculum is then generated of the process at that execution point. The hibernaculum will be a binary code representation of the process which contains program code, data and execution state. It is preferred that

configuration information is stored as execution state (e.g. as stack values or in registers) to make it more difficult for potential hacker to access such information. These elements can be interleaved or encrypted by any conventional means.

- 5 Execution Phase: The customer is provided with a utility which is capable of resuming execution of the process. Once the representation has been transferred to the customer, this is loaded into memory, the interleaving and/or encryption is reversed using the utility and the process resumed allowing the software to run. In order to prevent potential compromise of the digital goods, a series of representations may be
10 sent to the customer. The earlier representation can take on the role of reversing the encryption and interleaving of subsequent representations, thus complicating the task of an attacker in accessing the digital goods.

It will be appreciated that the software as received by the customer is not the same as
15 the software prior to the customization step, since it has been modified by the configuration input and may thus be made unique to that customer. The software resumes execution to perform all its intended functions. In some applications, the software can be saved by suspending the process at a subsequent execution point (to save data, for example) by use of the Checkpoint function by the customer, but the
20 program will then always resume from that subsequent execution point, so the customization phase is never re-entered, thus preventing tampering with the configuration of the software as set by the manufacturer/distributor.

It will further be appreciated that since the representation as passed to the customer as

a single binary file, embedded data is protected and cannot be easily accessed from outside the customization process.

A first specific embodiment of protecting videos transmitted over the Internet will now
5 be explained. The embodiment is implemented using the techniques described above, in particular using the computing language Java and using Web technology for user interface. Specifically, using the snapshot package described, the Java language is extended in conjunction with an extended Java Virtual Machine (JVM). This extended JVM provides the primitive commands described above to allow the creation of a
10 hibernaculum of a software process that can be transmitted over a communication network and subsequently be resumed in another computing environment.

In this embodiment, two machines, server 280 and client 290, are configured as shown in Fig. 11. The server 280 contains a Web server 300 offering digital video files for
15 sales over the Internet 305. This machine implements a typical electronic commerce system that allows users to browse, select and pay for the video files which are stored on a database 310. The server further includes an extended JVM, snapshot package and a Server program. The client 290 runs a standard Web browser 320, together with an extended JVM, snapshot package and a Client program to be used by customers
20 who wish to purchase/view the video files. The server 280 and client 290 further include a "hibernaculum server" 330 and "hibernaculum client" 340 which are arranged to communicate over a secure link 350 as described below.

The user interface is shown in Figs 12-14.

In order to alleviate the problem of use of any video transmitted by those other than the user who has paid for the video, various means may be employed. One is to assign or request from each user a userid and password for example using the input screen shown in Figure 12. Subsequently, some or all of this information is embedded in the
5 video viewing software prior to the hibernaculum being created to verify subsequently the authenticity of the user.

After verifying the user information, the browser will show a video-list page in which the user can select his or her favorite video and the number of video reviews required,
10 from the screen shown in Figure 13. The latter information is used to configure the viewing software, prior to the hibernaculum being created, to allow replaying of the video for a maximum number of times specified.

Once the user confirms the order by clicking the Order! button, the client machine
15 will launch the Client program. The server then sends two hibernacula to the client, one being of a process for loading the other, which is a hibernaculum of the video viewing process. This two-stage process is used for security, the other hibernaculum being encrypted and the one including a decryption key for the other. The screen display when the second hibernaculum is being sent is shown in Figure 14. The video
20 can then be played on the client machine by resuming the process of the second hibernaculum.

The operation of the server and client machines to provide these functions is as follows:

1. After the user has accessed the server's site over the World Wide Web, the server, under command of the Server program, runs the extended JVM and loads in a video viewing program having no video file saving capability or having such capability discarded using the Mutate 180 operation. This program then displays the screen of
5 Fig. 12 to the user to prompt for userid, password and any other information, such as client machine serial number, for access control.
2. The server loads the digital video (data) file selected by the user (Fig. 13) into the video viewing program and sets the userid and password parameters. The viewing
10 program may optionally apply a video watermarking process to embed user and licensing information directly into the video file.
3. The server sets the licensing parameters in the video viewing program including the number of video reviews allowed.
15
4. The server runs Construct or Checkpoint to create a first hibernaculum of the video viewing program process with all the parameters set and video files loaded. During the construction of the hibernaculum, the server may selectively discard data, code and execution state that are not needed for the intended use of the hibernaculum.
20 Specifically, data, code and executing state can be purposely made incomplete to complicate the task of others intending to misuse the hibernaculum.
5. The server selects a one-time secret key and encrypts the video viewing process with it.

6. The server embeds the same secret key into a hibernaculum loading process. This process may be created on demand or preloaded. The purpose for introducing the hibernaculum loading process is to make it more difficult to obtain access to the video through a possibly compromised JVM on the client machine.

5

7. The server then creates a hibernaculum out of the hibernaculum loading process and sends it to the Client using either the Migrate operations or a combination of Construct and Send operations, preceded by a command to the client machine instructing the user's browser to launch the Client program. The client program is a
10 simple routine which runs on the JVM of the client machine and uses the snapshot package to first Receive and then Assimilate an incoming hibernaculum.

8. Upon receiving the hibernaculum, the Client resumes the hibernaculum loading process with the Assimilate operation.

15

9. The hibernaculum loading process on the client machine makes a connection to the server machine and establishes a secure connection (e.g. using SSL) to protect the privacy and integrity of the network channel.

20 10. The server sends over the encrypted video viewing hibernaculum (Fig. 14). The hibernaculum loading process decrypts the video viewing hibernaculum and resumes the process.

11. The video viewing process prompts the user for password authentication. After

proper authentication, the user can proceed to play the video file.

12. After the maximum number of reviews allowed, the video viewing process stops the viewing capability and terminates itself automatically or awaits further
5 instruction from the user.

In another specific embodiment of the invention, the first specific embodiment described above is used for providing audio on-line. The second specific embodiment is practically the same as the first except that the files selectable by the user are audio
10 files rather than video files, and instead of a video viewing program, an audio playing program is used.

Preferably, the hibernaculum loading process is made vendor-specific by being independently written and by the inclusion of registration and other information. The
15 essential requirements of the program must be maintained, namely that the program must be able to decrypt the video viewing hibernaculum and resume the process but it is preferable for the vendor's hibernaculum loading process to only be able to load hibernacula from that vendor. Similarly, to prevent simple access to the video via the system routines for video image display, the video viewing process can use vendor-
20 specific video playing routine to make such video capturing difficult.

In the specific described embodiment, two hibernacula are generated, one of which is arranged to resume running of the other. However, any number of hibernacula may be linked in this way.

The embodiments described above may also be easily adapted to protect other digital material provided over open networks and can be combined with each other, so that a single server can transfer video, audio or programs, for example, depending upon user selection.

5

The embodiments of the invention described are not be construed as limitative, the scope of the invention being defined by the scope of the appendant claims.

CLAIMS

1. A representation of digital material, the digital material comprising program code and data and the representation comprising a combination of the code, data and
5 an execution state existing at a predetermined execution point when the program code is run, customization information being provided during running of the program code prior to the execution point being reached so that said representation is customized by said customization information.
- 10 2. A representation as claimed in Claim 1 wherein the information comprises information for the purpose of use control.
3. A representation as claimed in Claim 1 or Claim 2 wherein the customization information includes information relating to at least one of:
15 information identifying the material
information identifying an intended user of the material;
information identifying an intended machine on which the material is subsequently to be run;
information specifying the number of times the digital material may be run;
20 information configuring the digital material.
4. A representation as claimed in Claim 2 wherein the information identifying the material includes a watermark.

5. A representation as claimed in Claim 3 or Claim 4 wherein the information identifying an intended user of the material includes user authentication information.

6. A representation as claimed in Claim 5 wherein the authentication information
5 includes a cryptographic token.

7. A representation as claimed in any one of Claims 3 to 6 wherein the information identifying an intended machine on which the material is subsequently to be run comprises a serial number of the machine.

10

8. A representation as claimed in any one of Claims 3 to 7 wherein the information configuring the digital material comprises information specifying program code and/or data and/or execution state to be selected or discarded prior to the execution point.

15

9. A representation as claimed in any one of the preceding Claims wherein said digital material includes video data.

10. A representation as claimed in Claim 9 wherein the digital material includes a
20 program to play the video data.

11. A representation as claimed in Claim 9 or Claim 10 wherein the data included in the representation comprises at least one video recording specified by said customization information.

12. A representation as claimed in any one of Claims 1 to 8 wherein said digital material includes audio data.

13. A representation as claimed in Claim 12 wherein the digital material includes
5 a program to play the audio data.

14. A representation as claimed in Claim 12 or Claim 13 wherein the data included in the representation comprises at least one audio recording specified by said customization information.

10

15. A representation as claimed in any one of the preceding claims wherein the representation is encrypted.

16. In combination, a representation as claimed in any one of the preceding claims
15 and software arranged to use the representation as input for resuming the running of the program code from the execution point.

17. In combination, a representation of software arranged to use the representation of Claim 15, as input for resuming the running of the program code from the execution
20 point, the representation of software comprising program code, data and an execution state existing at a chosen execution point in running of the software, decryption information for decrypting the representation of Claim 15 being provided during running of the software prior to the chosen execution point being reached.

18. In combination, a plurality of representations as claimed in any one of claims 1 to 15, each representation but one being arranged to resume running of another of the representations.

5 19. A method of processing digital material, the material comprising program code and data, the method comprising the steps of:

(1) running the program code until a predetermined execution point is reached, an execution state existing at the execution point;and

(2) forming a combined representation of the execution state, data and program code
10 at that execution point, the data, code and execution state being restorable from said representation so that execution of the material may subsequently be resumed from the execution point.

20. A method as claimed in Claim 19 wherein, prior to the execution point,
15 customization information is provided whereby the representation is customized by the information.

21. A method as claimed in Claim 20 wherein the information comprises information for the purpose of use control.

20

22. A method as claimed in Claim 20 or Claim 21 wherein the customization information includes information relating to at least one of:

information identifying the material

information identifying an intended user of the material;

information identifying an intended machine on which the material is subsequently to be run;

information specifying the number of times the digital material may be run;

information configuring the digital material.

5

23. A method as claimed in Claim 22 wherein the information identifying the material includes a watermark.

24. A method as claimed in Claim 22 or Claim 23 wherein the information
10 identifying an intended user of the material includes user authentication information.

25. A method as claimed in Claim 24 wherein the authentication information includes a cryptographic token.

15 26. A method as claimed in any one of claims 22 to 25 wherein the information identifying an intended machine on which the material is subsequently to be run comprises a serial number of the machine.

27. A method as claimed in any one of Claims 22 to 26 wherein the information
20 configuring the digital material comprises information specifying program code and/or data and/or execution state to be selected or discarded prior to the representation being formed.

28. A method as claimed in any one of Claims 20 to 27 further comprising the step

of encrypting the representation.

29. A method as claimed in any one of Claims 20 to 28 further comprising the step of storing the representation.

5

30. A method as claimed in any one of Claims 20 to 29 further comprising the step of transferring the representation to a remote computing device.

31. A method as claimed in any one of Claims 20 to 30 further comprising the steps of restoring the data, code and execution state from said representation and resuming execution of the program code from the execution point.

10

32. A method as claimed in Claim 31 where dependent on Claim 28 further comprising the step of decrypting the encrypted representation prior to the restoring step.

15

33. A method as claimed in any one of Claims 20 to 32 further comprising the steps of generating a plurality of representations, each representation but one being arranged to resume execution of the program code from the execution point of another of the representations.

20

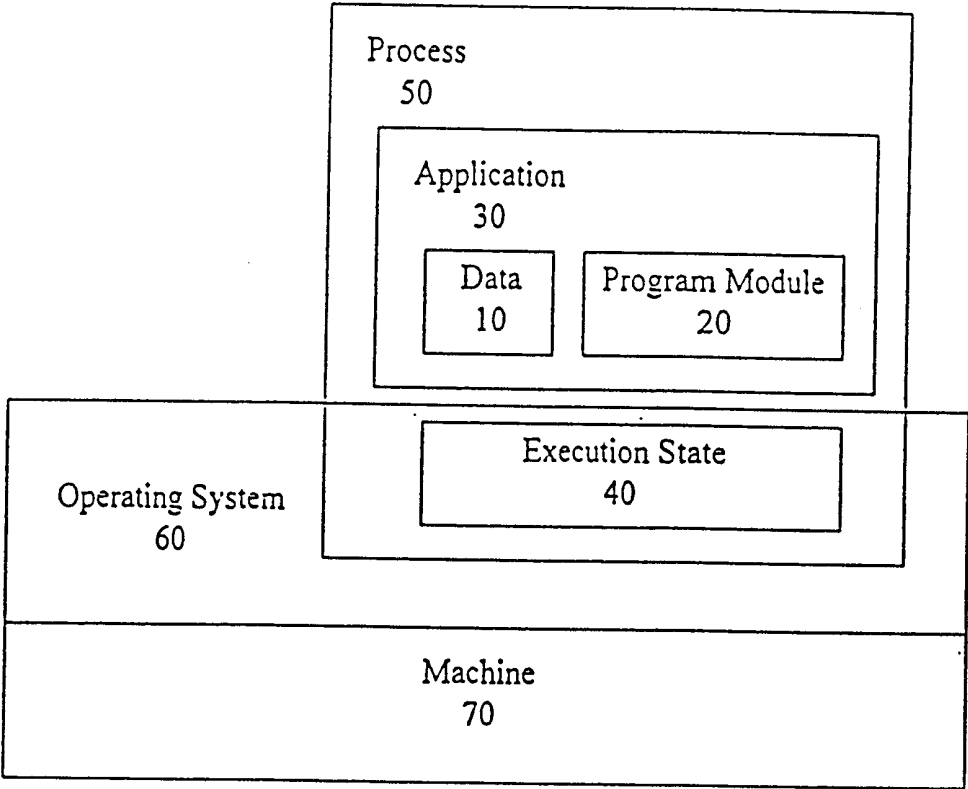


Fig. 1

2/12

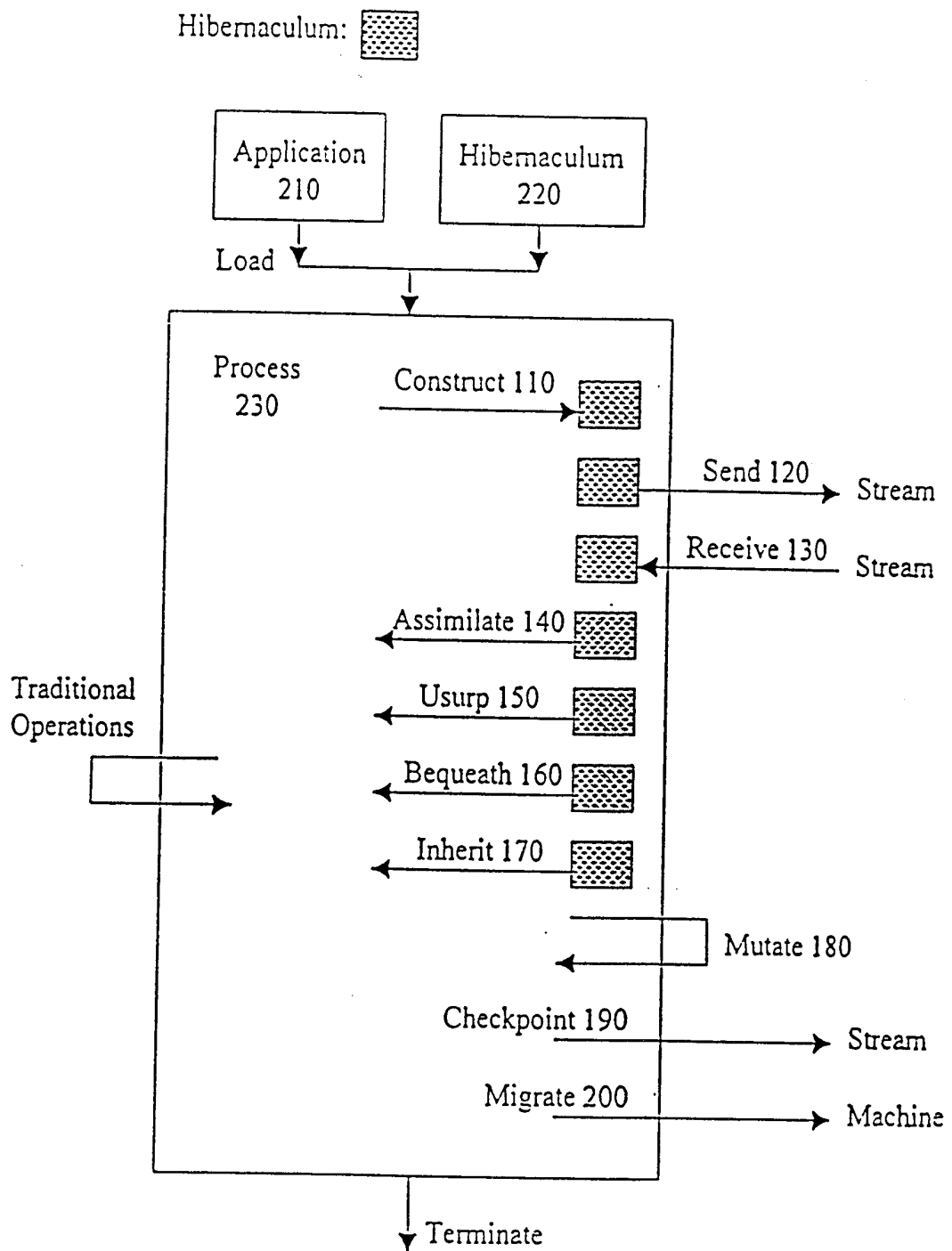


Fig. 2

3/12

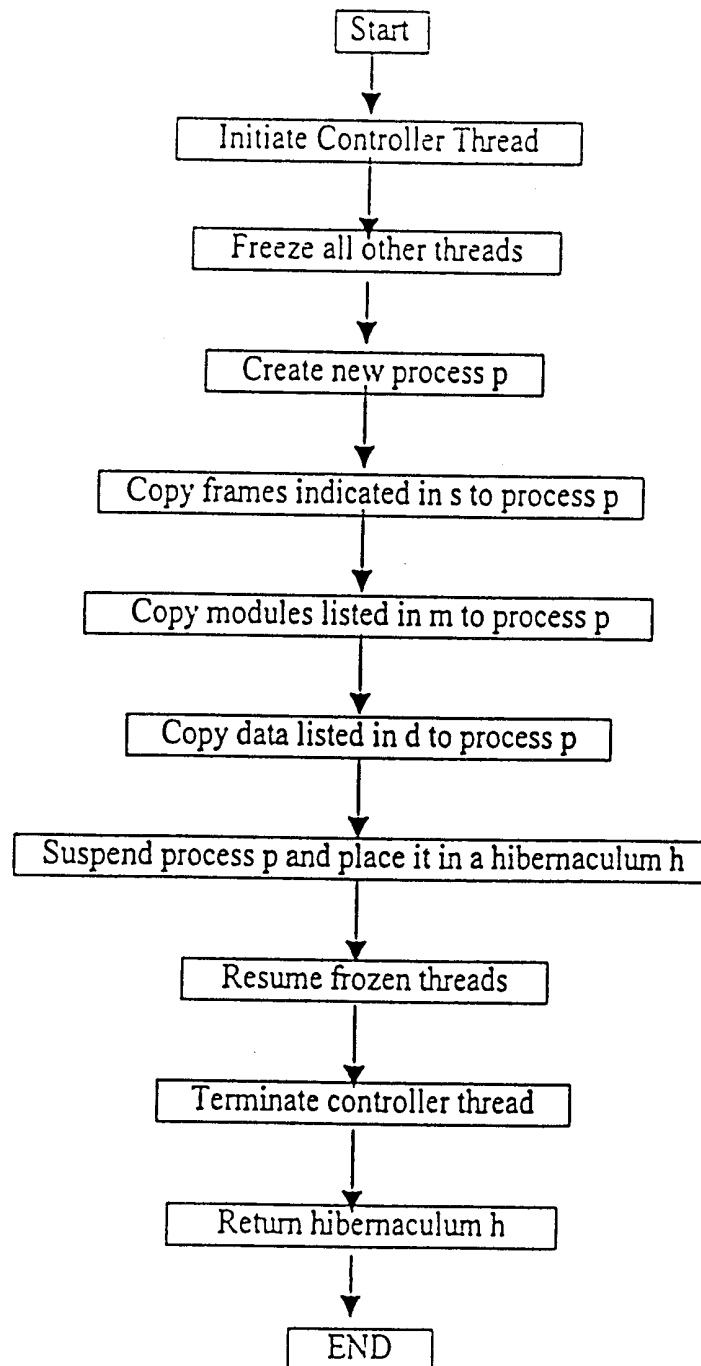


Fig. 3

4 / 12

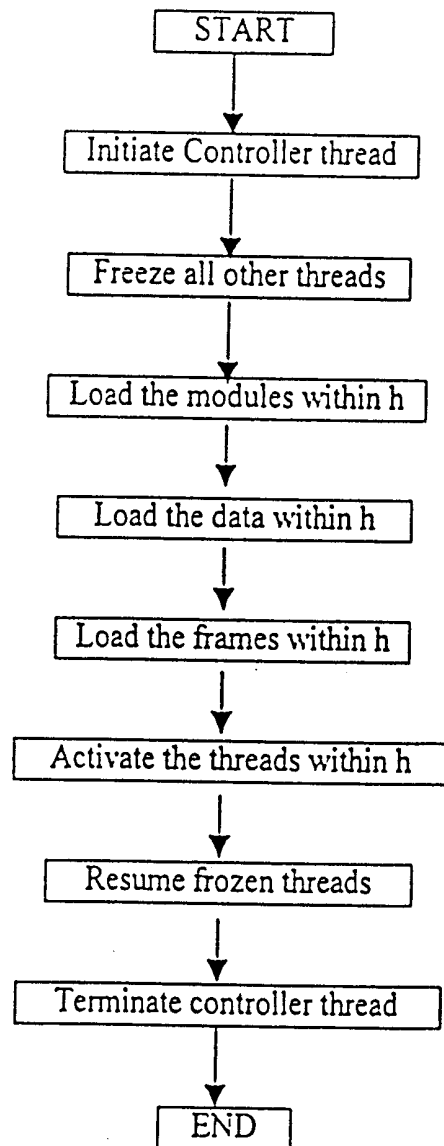


Fig. 4

5/12

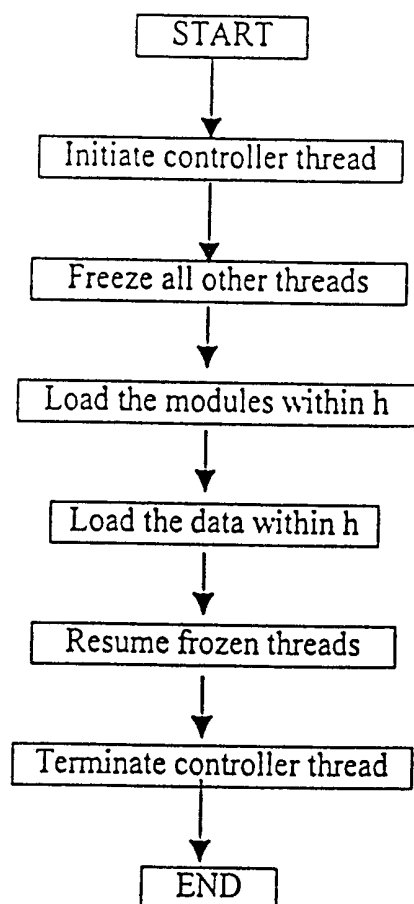


Fig. .5

6/12

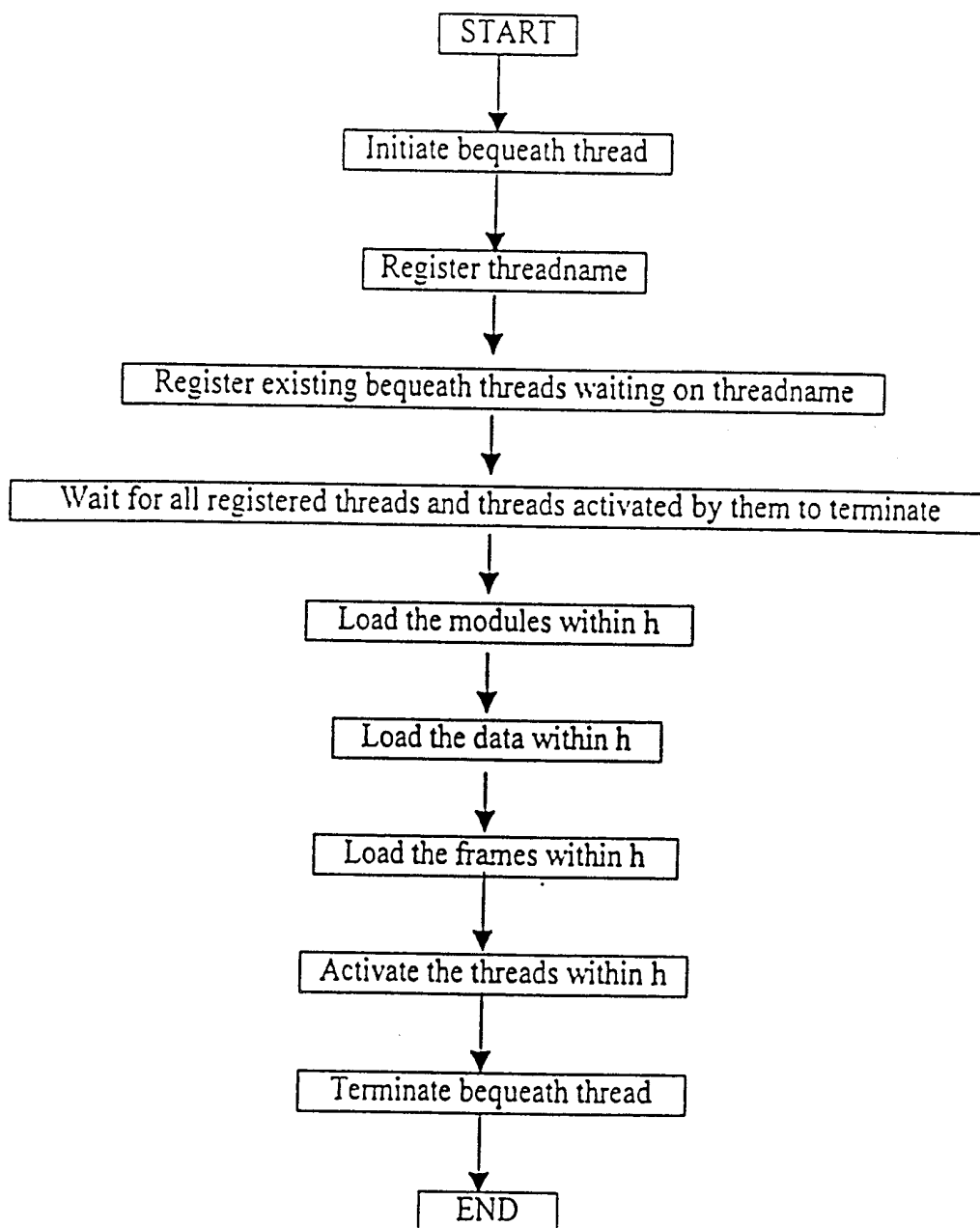


Fig. 6

7/12

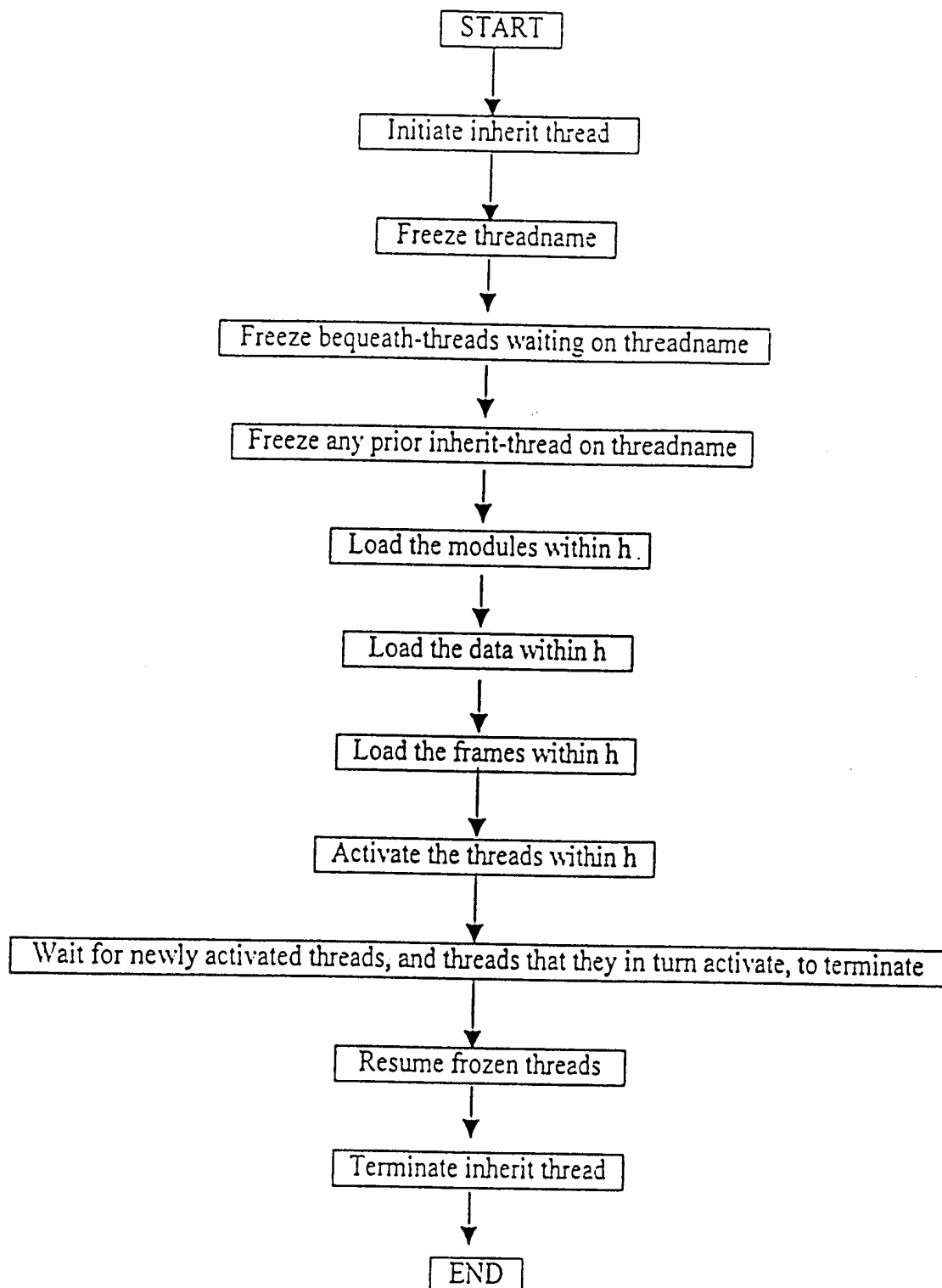


Fig. 7

8/12

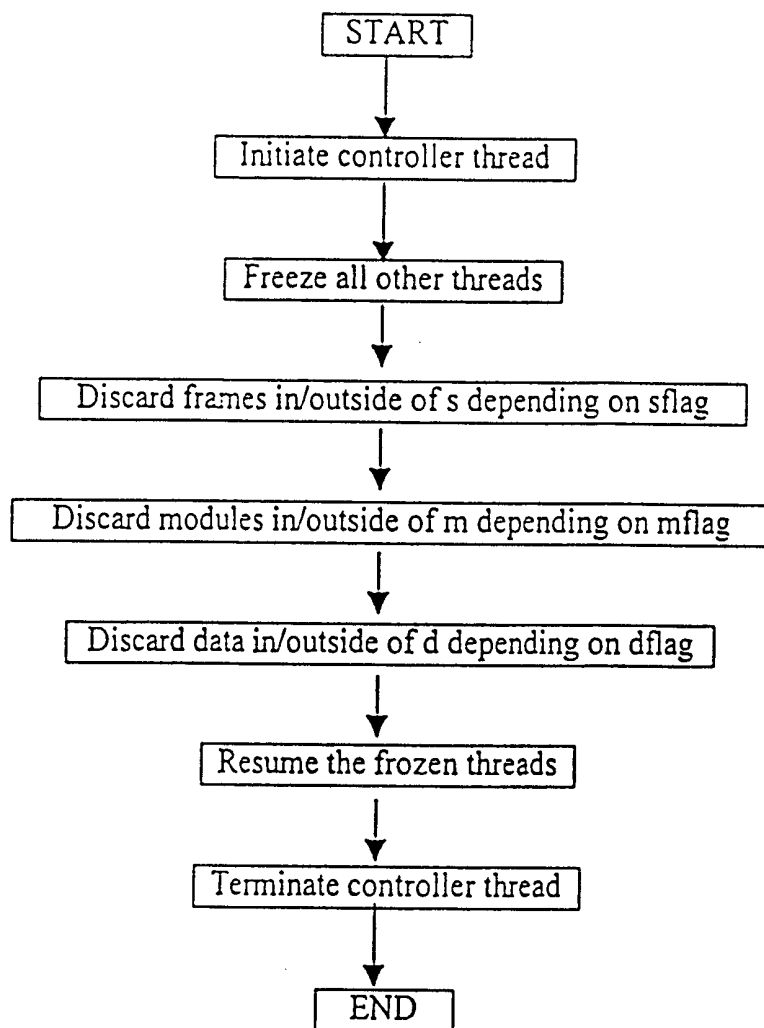


Fig. 8

9/ 12

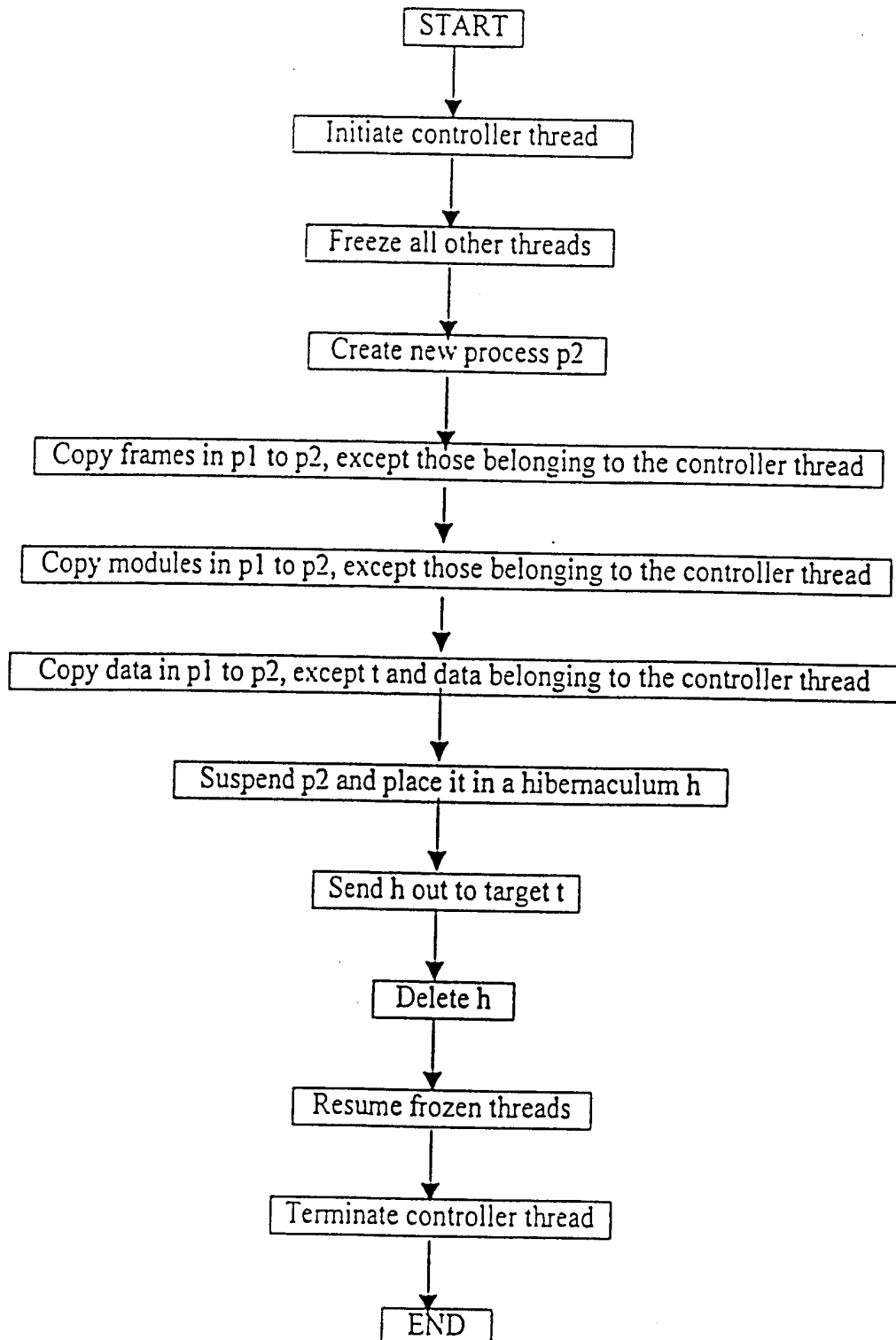


Fig. 9

10/12

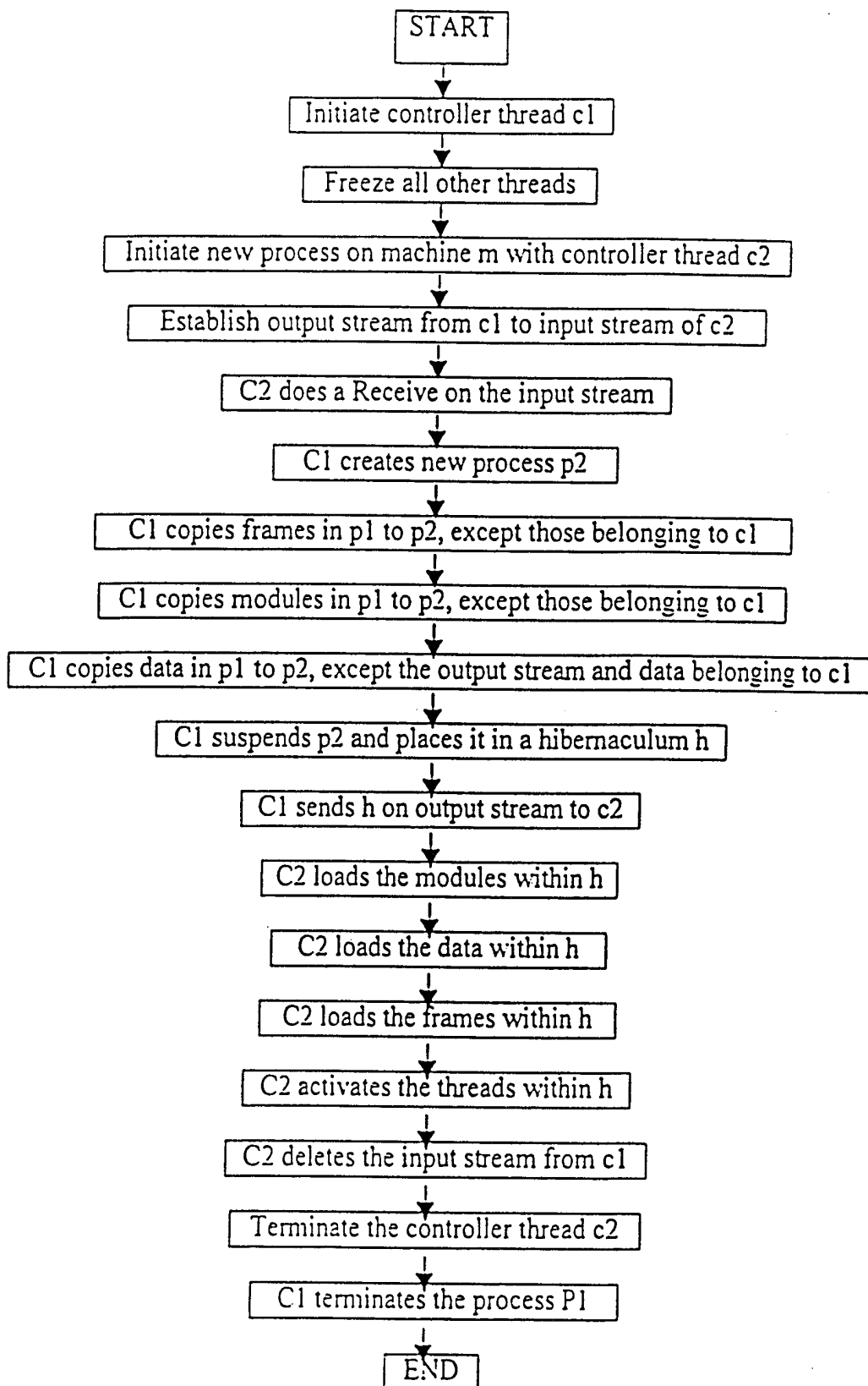
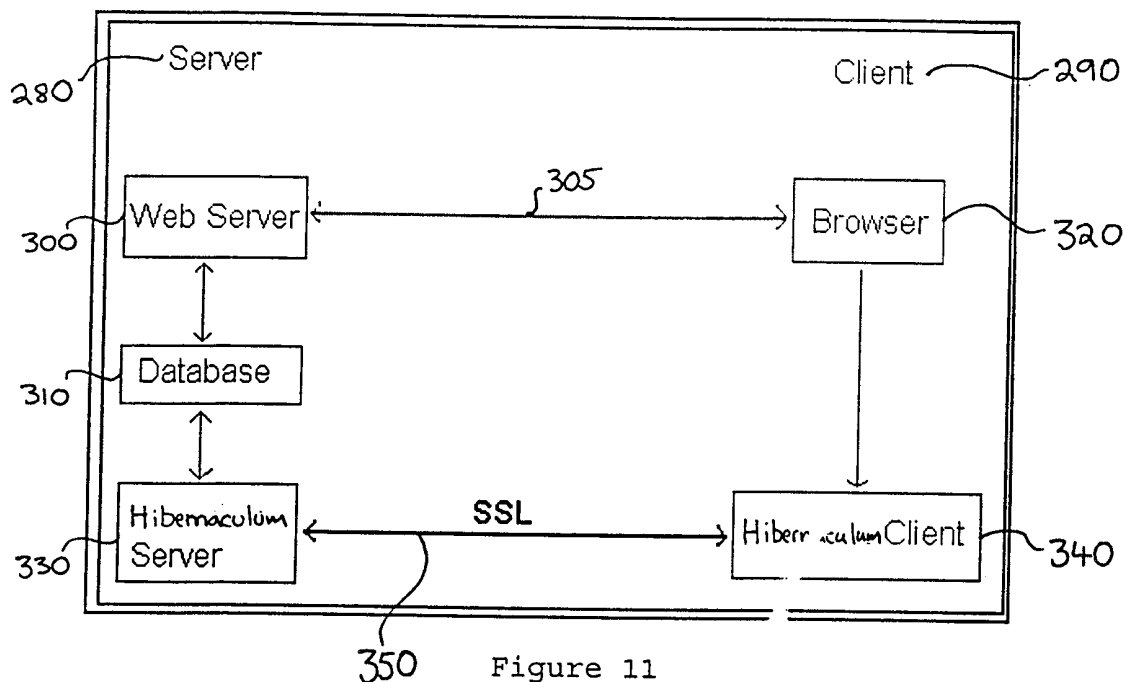


Fig. 10

11/12



Welcome to the Internet Video Purchase System

Check In

- Check in a member session.

MemberId

Password

- Check in a non-member session.
- Discard the existing session.

Membership Services

- New membership application.
- Change member particulars.
- Change member password.

Figure 12

12/12

Video On The Net




	<p>Title : An Early Summer Tour Actors : Good Boys Producer : Kent Ridge Digital Labs Price : S\$0.95 per view</p> <p>Time(s) : 1 <input type="button" value="Order"/></p>
	<p>Title : Happy Riding! Actors : UPS Members Producer : KRDL UPS Price : S\$1.99 per view</p> <p>Time(s) : 1 <input type="button" value="Order"/></p>
	<p>Title : Confession Actors : Javamaniacs Producer : Java Correctness Watch Group Price : S\$0.49 per view</p> <p>Time(s) : 1 <input type="button" value="Order"/></p>

Figure 13

My Site
KRDL

You have to pay S\$0.50 for playing 2 time(s) of video "bus.mpg"

Snapshot read in ...

65%

Received: 456KB/707KB

Figure 14

INTERNATIONAL SEARCH REPORT

International application No.
PCT/SG 99/00024

A. CLASSIFICATION OF SUBJECT MATTER

IPC⁷: G06F 17/14

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)

G06F

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

EPODOC, WPI, PAJ

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X A	US 5745569 A (MOSKOWITZ) 28 April 1998 (28.04.98), totality.	1-17,19-32 18,33
X	JP 10-232918 A (CANON) 1998-09-02 (abstract). [online] [retrieved on 1999-12-17]. Retrieved from EPO PAJ Database.	1-3,22
A	WO 9618951 A1 (DUNN) 20 June 1996 (20.06.96), abstract.	1-33
A	US 4558176 A (ARNOLD) 10 December 1985 (10.12.85), abstract.	1-33

☐ Further documents are listed in the continuation of Box C.

☒ See patent family annex.

* Special categories of cited documents:

„A“ document defining the general state of the art which is not considered to be of particular relevance

„E“ earlier application or patent but published on or after the international filing date

„L“ document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

„O“ document referring to an oral disclosure, use, exhibition or other means

„P“ document published prior to the international filing date but later than the priority date claimed

„T“ later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

„X“ document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

„Y“ document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art

„&“ document member of the same patent family

Date of the actual completion of the international search

18 January 2000 (18.01.00)

Date of mailing of the international search report

27 January 2000 (27.01.00)

Name and mailing address of the ISA/AT
Austrian Patent Office
Kohlmarkt 8-10; A-1014 Vienna
Facsimile No. 1/53424/200

Authorized officer

Fastenbauer

Telephone No. 1/53424/447

INTERNATIONAL SEARCH REPORT

International application No.
PCT/SG 99/00024

Box I Observations where certain claims were found unsearchable (Continuation of item 1 of first sheet)

This international search report has not been established in respect of certain claims under Article 17(2)(a) for the following reasons:

1. ☐ Claims Nos.:
because they relate to subject matter not required to be searched by this Authority, namely:
2. ☐ Claims Nos.:
because they relate to parts of the international application that do not comply with the prescribed requirements to such an extent that no meaningful international search can be carried out, specifically:
3. ☐ Claims Nos.:
because they are dependent claims and are not drafted in accordance with the second and third sentences of Rule 6.4(a).

Box II Observations where unity of invention is lacking (Continuation of item 2 of first sheet)

This International Searching Authority found multiple inventions in this international application, as follows:

See extra sheet

1. ☐ As all required additional search fees were timely paid by the applicant, this international search report covers all searchable claims.
2. ☒ As all searchable claims could be searched without effort justifying an additional fee, this Authority did not invite payment of any additional fee.
3. ☐ As only some of the required additional search fees were timely paid by the applicant, this international search report covers only those claims for which fees were paid, specifically claims Nos.:
4. ☐ No required additional search fees were timely paid by the applicant. Consequently, this international search report is restricted to the invention first mentioned in the claims: it is covered by claims Nos.:

Remark on Protest

- ☐ The additional search fees were accompanied by the applicant's protest.
☐ No protest accompanied the payment of additional search fees.

INTERNATIONAL SEARCH REPORT

International application No.

PCT/SG 99/00024

1	cl. 1,2,4			A representation of digital material ... information for the purpose of control ... watermark
2			cl.19-21, 23	A method of processing digital material running ... until a predetermined execution point forming a combined representation ... being restorable ... so that execution ... may be resumed ... representation customized by the information information for the purpose of control watermark
3	cl. 3		cl. 22	... customization information
4	cl. 5-6		cl. 24-25	... authentication information ... cryptographic token
5	cl. 7		cl. 26	... serial number of the machine
6	cl. 8		cl. 27	... program code ... to be selected or discarded ...
7	cl. 9-11			... video material ... program to play video material ... video recording specified ...
8	cl. 12-14			... audio data ... program to play audio data ... audio recording specified ...
9	cl. 15,17		cl. 28	... encrypted ... decryption information
10	cl.16			... to use the representation as an input for resuming running the program
11	cl.18			... to resume running of another of the representations
12			cl. 29	.. storing the representation ...
			cl. 30	... transferring the representation to a remote device ...
			cl. 31,32	... restoring the data, code and execution state further decrypting ...
13			cl. 33	... further generating a plurality of representations ...

INTERNATIONAL SEARCH REPORT

Information on patent family members

International application No.

PCT/SG 99/00024

Patent document cited in search report			Publication date	Patent family member(s)			Publication date
JP	A2	10232918	02-09-1998	none			
WO	A1	9618951	20-06-1996	AU	A0	25/94	12-01-1995
				AU	A1	42497/96	03-07-1996
				AU	B2	695468	13-08-1998
US	A	4558176	10-12-1985	none			
US	A	5745569	28-04-1998	AU	A1	18294/97	11-08-1997
				WO	A1	9726732	24-07-1997