(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
14 June 2001 (14.06.2001)

PCT

(10) International Publication Number
WO 01/42932 A2

(51) International Patent Classification⁷: G06F 13/00

(21) International Application Number: PCT/US00/33419

(22) International Filing Date: 8 December 2000 (08.12.2000)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
60/170,200 10 December 1999 (10.12.1999) US
09/648,985 28 August 2000 (28.08.2000) US

(71) Applicant: HONEYWELL INTERNATIONAL INC. [US/US]; 101 Columbia Avenue, P.O. Box 2245, Morristown, NJ 07960 (US).

(72) Inventors: ABOUTABL, Mohamed, Said; 1402 Spring lane, East Stroudsburg, PA 18301 (US). MOHAMED, Younis; 5029 Columbia Road #301, Columbia, MD 21044 (US).

(74) Agents: HOIRIIS, David et al.; Honeywell International Inc., 101 Columbia Avenue, P.O. Box 2245, Morristown, NJ 07960 (US).

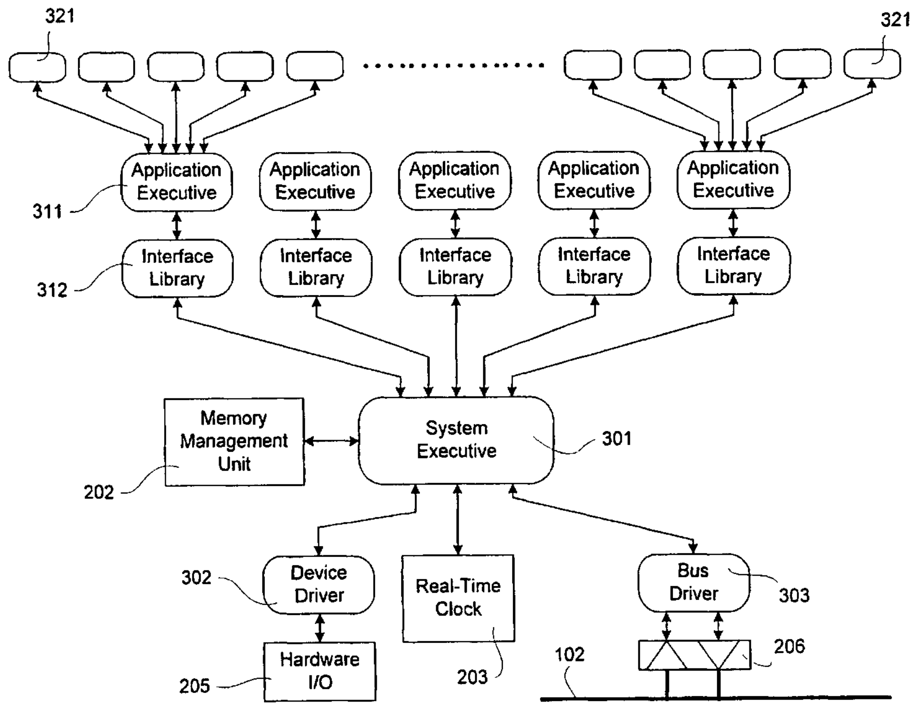
(81) Designated States (national): AE, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, CA, CH, CN, CR, CU, CZ, DE, DK, DM, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, TZ, UA, UG, UZ, VN, YU, ZA, ZW.

(84) Designated States (regional): ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE, TR), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).

Published:
— *Without international search report and to be republished upon receipt of that report.*

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

(54) Title: TWO LAYER OPERATING SYSTEM AND METHOD FOR AVIONICS SOFTWARE APPLICATIONS



(57) Abstract: Integrated Modular Avionics (101) can take advantage of faster modern processors and can run multiple existing avionics software applications (321) on a single processor (201) by using a two-layer operating system consisting of a system executive (301) and multiple application executives (311).

UNITED STATES PATENT APPLICATION FOR:

TWO LAYER OPERATING SYSTEM AND METHOD FOR AVIONICS SOFTWARE APPLICATIONS

FIELD OF INVENTION

This invention relates to a circuit module and method for administrating process or job execution over a digital data processing system, especially for a an Integrated Modular Avionics circuit card programmed to provide a two-layer operating system.

BACKGROUND OF THE INVENTION

Several newly designed aircraft and new retrofits for older aircraft will use an integrated avionics architecture that is referred to in the industry as Integrated Modular Avionics (IMA). Prior to the development of integrated avionics architectures, aircraft typically used many line replaceable units (LRU), such as radios, where each LRU performed a dedicated function, such as VHF communication or VOR navigation. IMA uses a few multifunction LRUs to perform the various avionics functions, typically performed by several dedicated LRUs. Each IMA LRU contains several modules that perform processing, inputs, and outputs to other aircraft hardware. The IMA approach uses a common chassis and power supply for the various modules within each IMA LRU. This allows for an overall avionics system that is lighter and lower in cost than the typical 'federated' avionics architecture found on older generation aircraft.

A large number of software applications have been developed for and certified on federated avionics systems as described above and it is advantageous to reuse this previously developed software. However, the existing avionics software applications run on several different real-time operating systems. There also exists a need to support integration of multiple software applications that have traditionally been implemented in federated LRUs. It is known that one of the side effects of merging multiple software applications using existing software development tools is unwanted dependencies between the software applications, such as the propagation of faults from one failing software application to others.

For FAA aircraft certification, it is necessary to be able to show, with a very high level of assurance, that a problem or failure in a software application cannot have an adverse impact on any other software application.

A prior art avionics operating system environment is described in ARINC Specification 653, "Avionics Application Software Standard Interface", dated. January 1997, and RTCA SC-182/EUROCAE WG-48, "Minimum Operational Performance Standards for Avionics Computer Resource", dated November 1998.

SUMMARY OF THE INVENTION

The invention integrates multiple software applications to run on one central processing unit (CPU), wherein technology has made available CPUs that are powerful enough to meet the combined computation demands of several avionics software applications. The invention further provides a software architecture in which the operating system portion is split into two distinct layers, consisting of a system executive layer and multiple application executive layers. Advantageously, this software architecture allows various real-time operating systems to run concurrently on the same CPU. System-level functions such as software and configuration database loading, application health monitoring, problem logging, basic operating system support and high-level handling of input/output can be shared among the several applications running on an integrated modular avionics (IMA) module.

Advantageously, the two-layer architecture provides the ability to integrate software applications developed by various software vendors. The invention also eliminates the need to re-test the whole set of software applications running on an IMA module when only a single software application is added, upgraded, or removed from the system.

The first, system executive, layer of the two-layer architecture provides each software application with a protected partition, within which each distinct software application can execute together with an appropriate application executive. The second, application executive, layer of the two-layer architecture is a modified version of a real-time operating system that provides each software application with a virtual machine and a set of interface library (IL) functions. These IL functions facilitate communication between the application executive and the system executive. An embodiment of the invention includes one system executive and multiple application executives.

Advantageously, each application executive and its associated software applications are spatially isolated from other application executives and their associated software applications by enforcing access restrictions on memory address space. In addition, each application executive and its associated software applications are temporally isolated from other application executives and their associated software applications by enforcing usage restrictions on the CPU and other system resources based on a pre-computed static execution timetable.

The system executive initializes, monitors, and terminates each software application modules and maintains a real-time clock to strictly implement the execution timetable from which each software application is assigned well-defined time slices. The system executive also handles context switching and communication between the various application executives, manages all IMA module hardware input and output resources, and enforces strict isolation of protected memory regions. Advantageously, the system executive prevents the propagation of faults and extraneous data across the various application executives and their associated software applications by assigning and enforcing protected memory partitions for each application executive.

Each application executive is a customized version of an available real-time operating system. Each application executive provides services, for its associated software applications tasks, including communication, synchronization and dynamic memory management. Each application executive also provides, for its associated software applications, access to system level resources, including IMA module hardware input and output (I/O) devices, via interface library functions. In one embodiment, an application executive implements its own strategy for scheduling tasks contained in an associated software application.

Customization of an available real-time operating system into an application executive comprises redirecting functions related to communication, memory management, and access to I/O devices through the interface library functions to the system executive. It is an aspect of the invention that no application executive may perform any boot sequence procedure such as probing for or initialization of hardware devices, initializing interrupt tables, or setting up registers of a memory management unit (MMU). These boot sequence procedures, of the available real-time operating systems, are replaced by a set of initialization data structures located within the protected memory partition of the application executive. As part of the system initialization procedure, the system executive initializes the initialization data structures of all application executives.

BRIEF DESCRIPTION OF THE DRAWINGS

Figure 1 illustrates a prior art line replaceable unit (LRU), commonly referred to as a 'black box' that is intended for aircraft installation. The particular LRU shown is representative of an integrated modular avionics (IMA) cabinet.

Figure 2 illustrates a module or circuit card assembly (CCA), in accordance with one illustrative embodiment of our invention that is suitable for installation in an IMA cabinet.

Figure 3 illustrates the software architecture of the software that runs on each IMA module according to our invention.

Figure 4 illustrates a flow chart of the system executive layer of our invention.

Figure 5 illustrates the interaction between the system executive layer and each application executive layer as described in the present invention.

Figure 6 illustrates a timeline showing the execution sequence of the system executive, and the application executives, the software applications associated with each of the several application executives.

DETAILED DESCRIPTION OF THE INVENTION

Referring first to Figure 1, an integrated modular avionics (IMA) chassis 101, as presently used, is shown. The IMA chassis 101 contains a multiplicity of IMA modules 200, one or more connectors 103, and a motherboard 102 to interconnect the IMA modules 200 with each other and with the connectors 103. The connectors 103 interface aircraft electrical signals with the circuitry contained on the IMA modules. In other embodiments, optical and radio frequency signals are communicated between the IMA modules and other aircraft equipment.

Referring to Figure 2, in an embodiment of our invention, an IMA module 200 is provided with a memory management unit 202. The memory management unit 202 splits system memory 204 into protected partitions and controls read and write access to the partitions according to context instructions sent from a system executive 301 (seen in Figure 3), which executes on a central processing unit 201. A clock 203 generates periodic timer interrupts 505, as shown in Figure 5, to the central processing unit 201. The clock 203 is a real-time clock running independent of the rest of the IMA module 200 hardware. Advantageously, the system executive 301 is able to read the current time from the clock 203 without disrupting its operation. The IMA module 200 of our invention also includes the input/output hardware device 205, input/output bus device 206 and connectors 211 to interface electrical signals with the IMA motherboard 102.

Figure 3 illustrates an architectural layout of a complement of software that is programmed to run on the central processing unit 201. As shown in Figure 3, it is an aspect of our invention that the software applications 321 do not directly communicate with the input/output hardware devices 205 or the input/output bus devices 206. A software application 321 that had been previously linked with an associated real-time operating system during a previous build is linked with an associated application executive 311 and an associated set of interface library functions 312, according to our invention. The software applications 321 communicate with the input/output hardware devices 205 and the input/output bus devices 206 by calling interface library functions 312 associated with each application executive 311. The interface library functions 312 access the input/output hardware device 205 by calling a hardware device driver 302 software application via the system executive 301. All functions of each hardware device 205 are controlled via device driver 302 software applications. The interface library functions 312 access the input/output bus device 206 by calling a bus driver 303 software application via the system executive 301. All functions of each bus device 206 are controlled via device driver 303 software applications.

According to our invention, the system executive 301 consists of a set of instructions that are stored in the memory 204 and are executed on a central processing unit 201. This system executive 301 is responsible for the operation of all devices mounted on IMA circuit card 200, which in turn is installed in the IMA chassis 101. Each application executive 311 consists of a set of instructions that are stored in a memory 204 within an application partition 500 (shown in Figure 5) address space where read/write access in controlled by the memory management unit 202, and the instructions are executed on the central processing unit 201. The software applications 321, interface library 312, and timer interrupt services routines 501 (shown in Figure 5) associated with each application executive 311 also consist of instructions that are executed on central processing unit 201 and are stored in the same application partition 500 (shown in Figure 5) as the associated application executive 311.

Figure 4 shows a flow chart that illustrates the instruction steps of the system executive 301 as it executes on the central processing unit 201. The system executive 301 comprises a set of start-up steps that are executed once followed by a main loop 410 comprising steps that are executed in a pre-determined sequence that repeats indefinitely. In another embodiment of our invention, that is not shown, the system executive 301 further comprises a sequence of shut-down steps that are executed once after the main loop is terminated.

The start-up steps of the system executive 301 comprise the following sequence of software steps that are executed by the central processing unit 201. First, the input/output hardware devices 205 and the input/output bus devices 206 are initialized at step 401. Second, the data memory 204 is partitioned into virtual application memory partitions 500 (shown in Figure 5) by causing the central processing unit 201 to issue a command sequence to the memory management unit 202 at step 402.

The indefinite main loop 410 of the system executive 301 comprises the following sequence of software steps that are executed by the central processor 201. First, the system executive 301 reads (step 415) the application time slice 601 (shown in Figure 6) and the application executive 311 clock tick length 602 (shown in Figure 6), associated with the next scheduled application partition 500, from the static time table schedule (not shown). Next, the system executive 301 calculates (step 415) the number of full-length ticks, 'Nticks', and the length of the remaining partial tick, 'parTick', using the following equations.

```
Eq. 1   Nticks := TimeSlice / TickLength;     /* Division - integer result */

Eq. 2   parTick := TimeSlice % TickLength;  /* Modulo Division - remainder */
```

The application full length clock ticks, 'Nticks', and remaining partial tick 'parTick' are used to update the application partition 500 (shown in Figure 5) local time structure (not shown).

Next, the central processing unit 201 is instructed to busy-wait for a starting time of a next application executive 311 at step 411. The starting time of each application executive 311 is stored in a predefined scheduling timetable (not shown) that resides in memory 204. Next, at step 412, the memory management unit 202 is instructed to use an associated virtual memory partition 500 (shown in Figure 5) to resolve memory address references and control of central processing unit 201 is passed to application executive 311. During step 413, the application executive 311 provides instructions to central processing unit 201 except when periodic timer interrupts cause associated timer service routines 501 to be run, as shown in Figure 5. These timer interrupt service routines 501 comprise instructions that are run at the same privilege level as the system executive 301. When the time slice 601 (as shown in Figure 6) allocated to the application executive 311 expires, control of the central processing unit 201 is returned to the system executive 301 and the main loop 410 continues indefinitely. Strict enforcement of a time-based dispatching of application partitions, as described above, is based on a static timetable schedule (not shown), which is created during overall software build.

Referring again to Figure 3, the application executive 311 is a modified version of a real-time operating system for which associated application software 321 was originally developed. According to our invention, it is essential that the system executive 301 be the first to respond to any exception raised by software applications 321; therefore an interrupt service routine 501 (shown in Figure 5) is provided to intercept 'exception interrupts' that are raised by the central processing unit 201. Re-mapping exception handling functions from existing real-time operating systems requires functionality in both the system executive 301 and the application executive 311. According to our invention, both the software application 321 and the application executive 311 run low 'user' privilege level instructions on central processing unit 201 as contrasted with the system executive 301 and the application interrupt service routine 501 (shown in Figure 5), which both run high 'operating-system' privilege level instructions on central processing unit 201.

The interface library 312 comprises a set of functions that are created to replace some of the services provided by a specific real-time operating system; advantageously, a minimum number of real-time operating system services are replaced. Further, the device driver 302 may directly incorporate low-level code of a prior art real-time operating system.

Each application executive 311 maintains its own data structures to keep track of the progress of real-time. In one particular embodiment, the time duration between periodic timer interrupts 505 (shown in Figure 6) is 10 milliseconds.

Figure 5 illustrates the details of the transfer of control between the system executive 301, an application executive 311, and a software application 321. The system executive passes control of the central processing unit 201 at step 412 using a timer interrupt service routine 501. The timer interrupt service routine 501, associated with the application executive 311, executes instructions on central processing unit 201 at the same privilege level as the system executive 301. The timer interrupt service routine 501 comprises instructions that are stored in the data memory 204 within the application partition 500 address space. The application executive 311 always acts as the entry point of every application partition 500. The application memory partition

500 cooperates with the timer interrupt service routine 501, associated with the system executive 301, that services initial entry into the application partition 500 and periodic timer interrupts 505 that occur during the application partition's time slice 601 (as shown in Figure 6). The application executive 311 schedules the software application 321 tasks within the time slice 601. When the time slice 601 expires, the timer interrupt service routine 501 stores the current time and current state of the partition and returns controls of the central processor unit 201 to the system executive 301.

Upon reentry into partition 500, timer interrupt 501 adjusts partition local time data structure and passes control to the application executive 311. The application executive 321 will determine which task from associated software applications is due and will dispatch said task. According to our invention, it is essential that control of the central processing unit 201 is not given directly to a software application 321 task, but is rather first handed to the associated application executive 311, even if the task was interrupted before completion. The memory space allocated for each application partition 500 includes a predetermined amount of 'heap' memory for use as a dynamic memory pool. Each application executive 311 manages the dynamic memory 'heap' within the application's own partition 500.

Referring to Figure 6, the central processing unit 201 executes all instructions that are included in the system executive 301, the application executives 311, and the software applications 321. Timer interrupts 505 are periodically applied to the central processing unit 201 from the clock 203. These timer interrupts 505 are used to initiate the execution of the system executive 301 and the application executive 311 as defined in a static schedule (not shown). This timeline provides temporal isolation between the software applications 321 associated with different application executives 311. Each application partition 500 is allocated an application time slice 601 and a clock tick length 602 in a predefined static timetable (not shown). Advantageously, this temporal isolation prevents the software applications from interfering with one another.

Our invention accordingly comprises a two-layer operating system for use with multiple avionics software applications 321 that run various aircraft subsystems. The processing throughput and hardware interfaces for these multiple applications are contained on a single IMA card 200 which is installed with other similar cards in an aircraft mounted cabinet 101. Advantageously, our invention takes advantage of the higher speed processors that are currently available, while still allowing reuse of previously developed avionics software applications. The multiple software applications are temporally (time) and spatially (memory) isolated from each other.

WHAT IS CLAIMED IS:

1. A circuit module for an Integrated Modular Avionics arrangement, said module comprising:

    input/output hardware,

    an input/output bus,

    a real-time clock for generating a sequence of periodic timer interrupts,

    a memory management unit,

    a memory partitioned into a plurality of application partitions by said memory management unit; and

    a central processing unit, said central processing unit being programmed to provide a two-level software architecture including

    a system executive comprising an initialization sequence, an indefinite loop and a multiplicity of timer interrupt service routines responsive to said real-time clock,

    a plurality of application executives, each comprising a real-time operating system and associated with one of said timer interrupt service routines; and

    control of said memory management unit.

2. The circuit module according to claim 1 wherein the application executive further comprises an interface library of software functions that provide access to a set of hardware input/ output devices.

3. The circuit module according to claim 1 wherein the application executive maintains its own data structures to keep track of the progress of real-time.

4. A method for executing multiple existing avionics software applications, wherein the applications have been separated from their previous operating environment and have been linked with an application executive running concurrently with a system executive on a single central processing unit that is mounted on a circuit card assembly along with a memory management unit, a clock, data memory, input/output hardware devices and input/output bus devices, said method comprising the steps of:

initializing the input/output hardware devices,
initializing the input/output bus devices,
partitioning the data memory using the memory management unit, and
entering into and remaining in an indefinite loop comprising the steps of:

waiting for a starting time of an application executive,

instructing the memory management unit to resolve addresses in accordance with an application partition associated with said application executive,

passing control of the central processing unit to said application executive,

executing commands in accordance with said application executive,

waiting for a time slice associated with said application executive to expire, and

returning control of the central processing unit to the system executive.

5. The method of claim 4, wherein said indefinite loop further comprises the steps of:

waiting for a starting time of a second application executive,

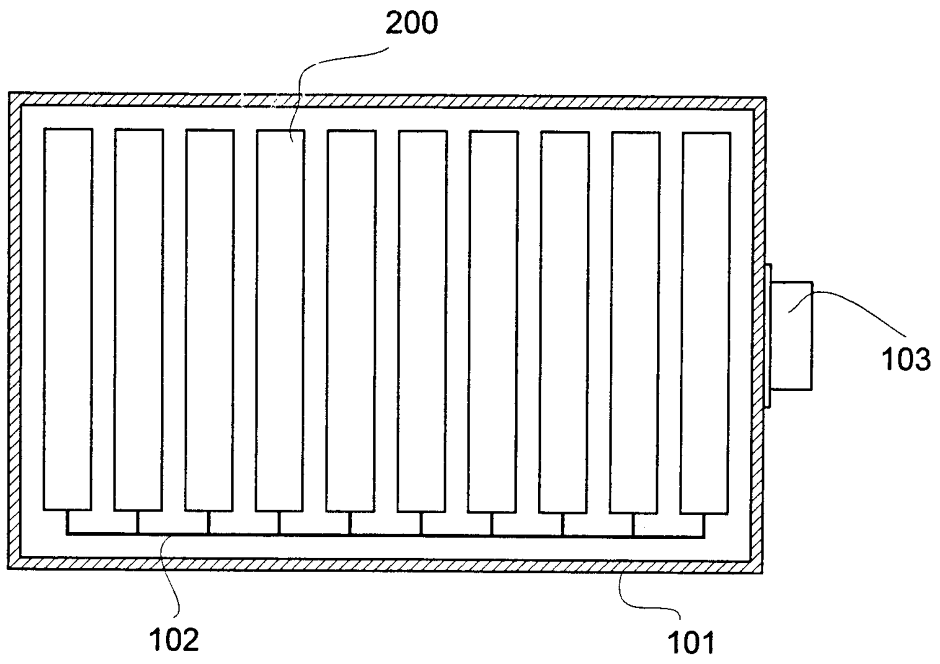
instructing the memory management unit to resolve addresses in accordance with a second application partition associated with said second application executive,

passing control of the central processing unit to said second application executive,

executing commands in accordance with said second application executive,

waiting for a time slice associated with said second application executive to expire, and

returning control of the central processing unit to the system executive.



PRIOR ART

Figure 1

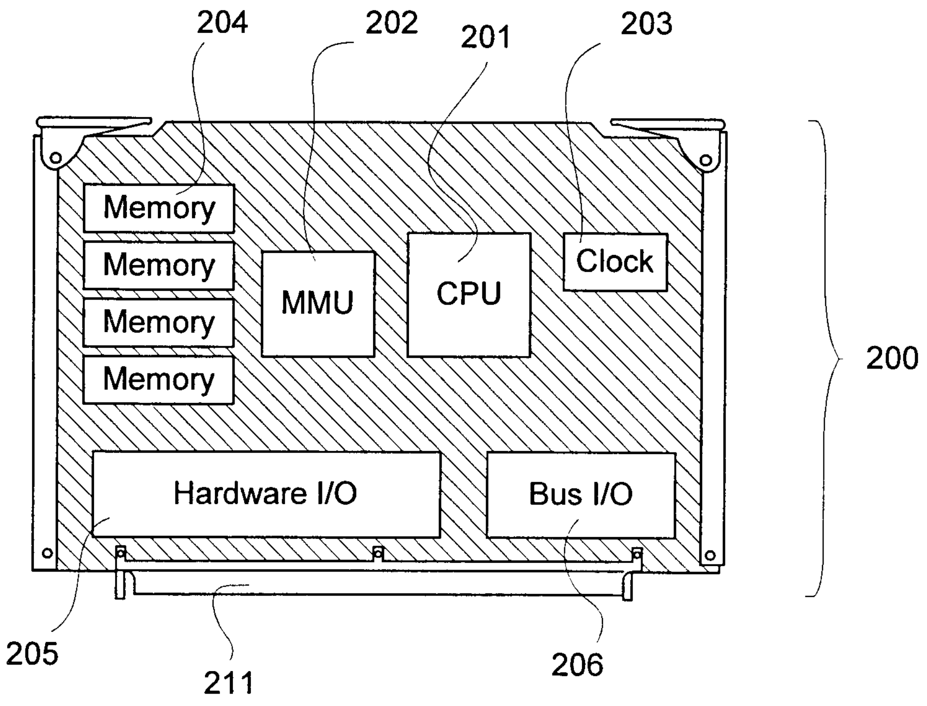

204
202
201
203
Memory
Memory
Memory
Memory
MMU
CPU
Clock
200
Hardware I/O
Bus I/O
205
211
206

Figure 2

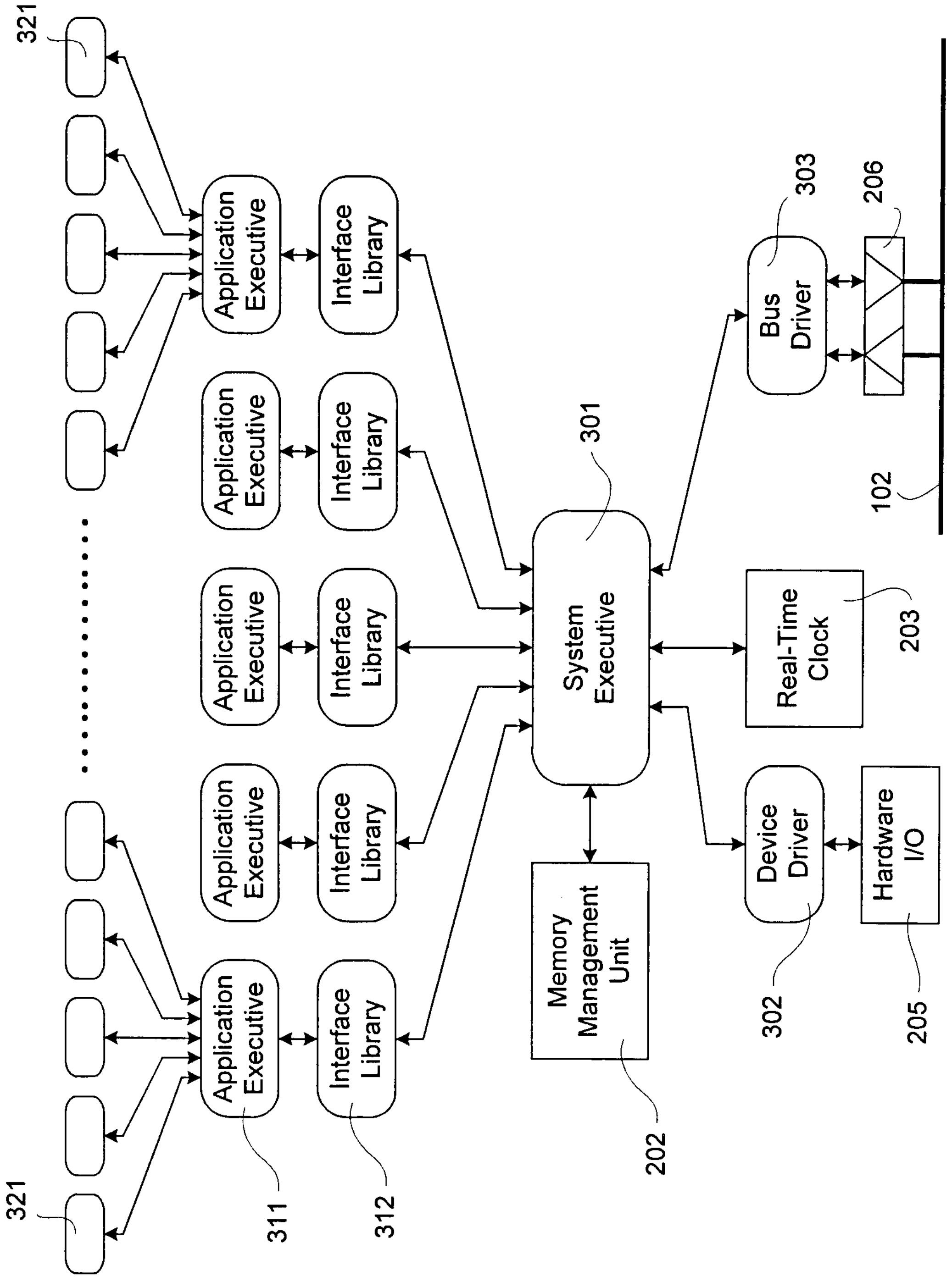

321
Application Executive
Interface Library
311
312
System Executive
301
Memory Management Unit
202
Device Driver
302
Hardware I/O
205
Real-Time Clock
203
Bus Driver
303
206
102

Figure 3

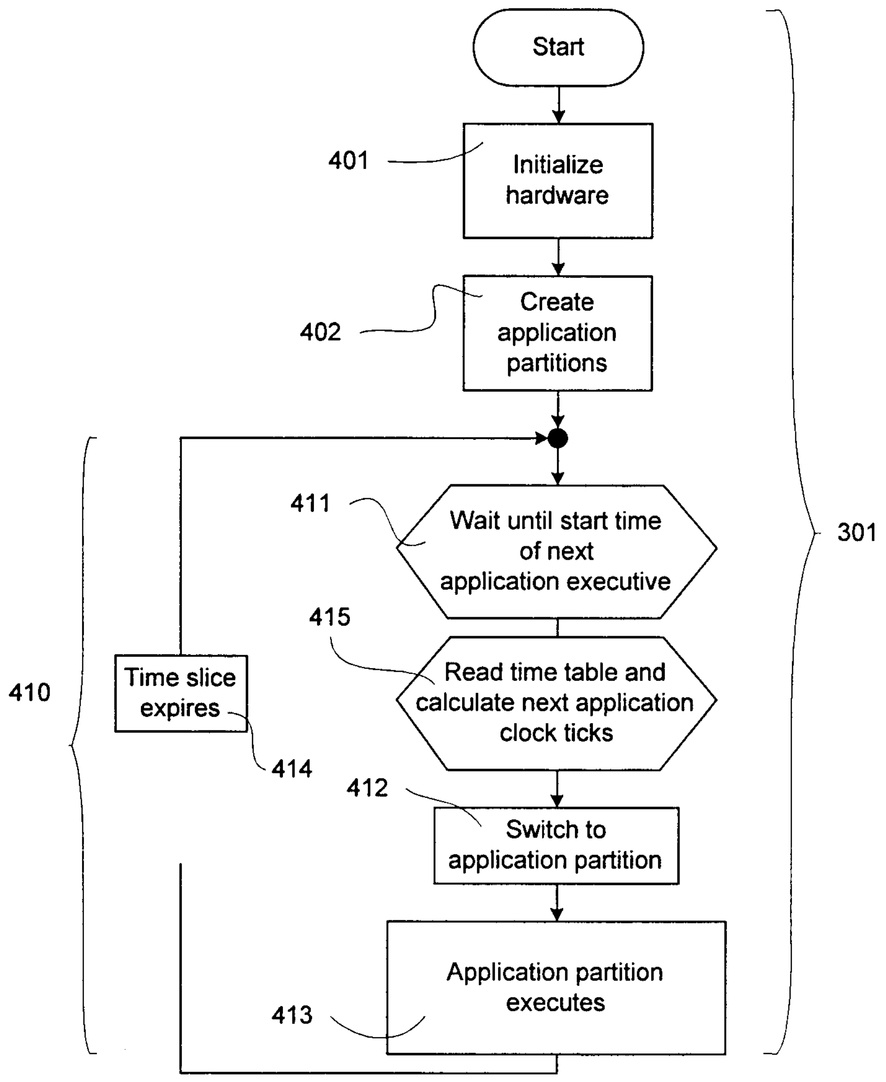

Start
401
Initialize hardware
402
Create application partitions
411
Wait until start time of next application executive
415
Read time table and calculate next application clock ticks
410
Time slice expires
414
412
Switch to application partition
413
Application partition executes
301

Figure 4

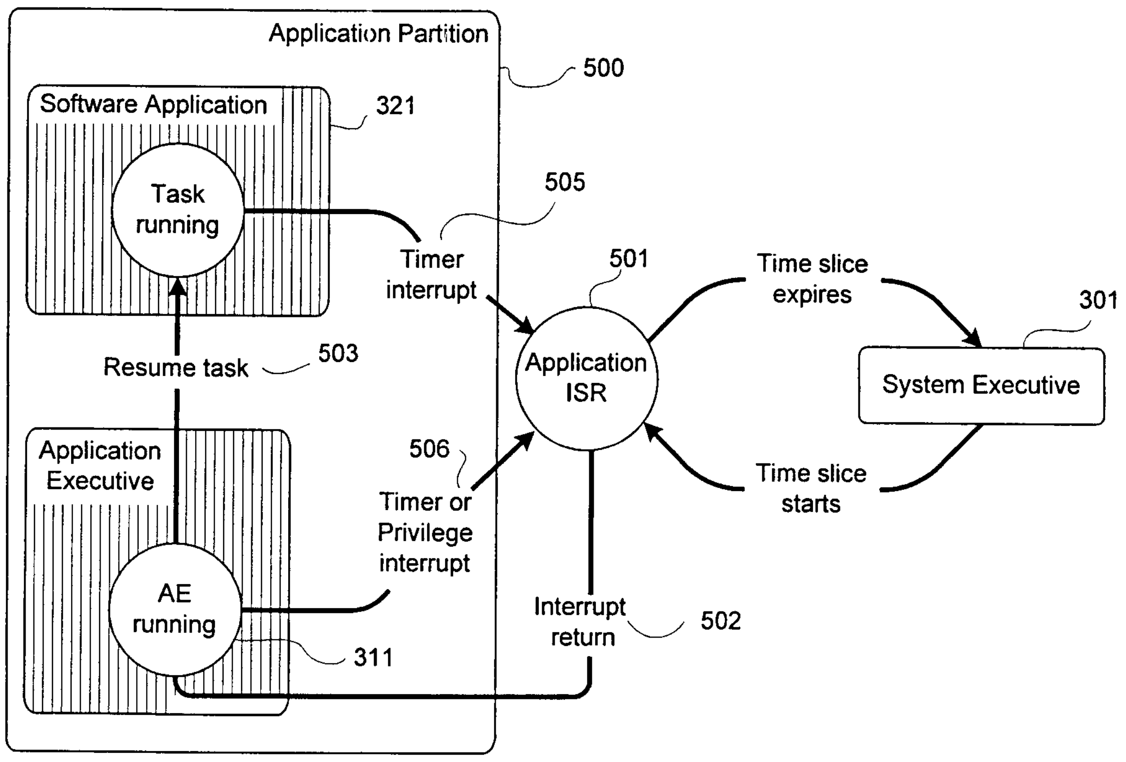

Application Partition
500
Software Application
321
Task running
505
Timer interrupt
501
Time slice expires
301
Application ISR
System Executive
Resume task
503
Application Executive
506
Timer or Privilege interrupt
Time slice starts
AE running
Interrupt return
502
311

Figure 5

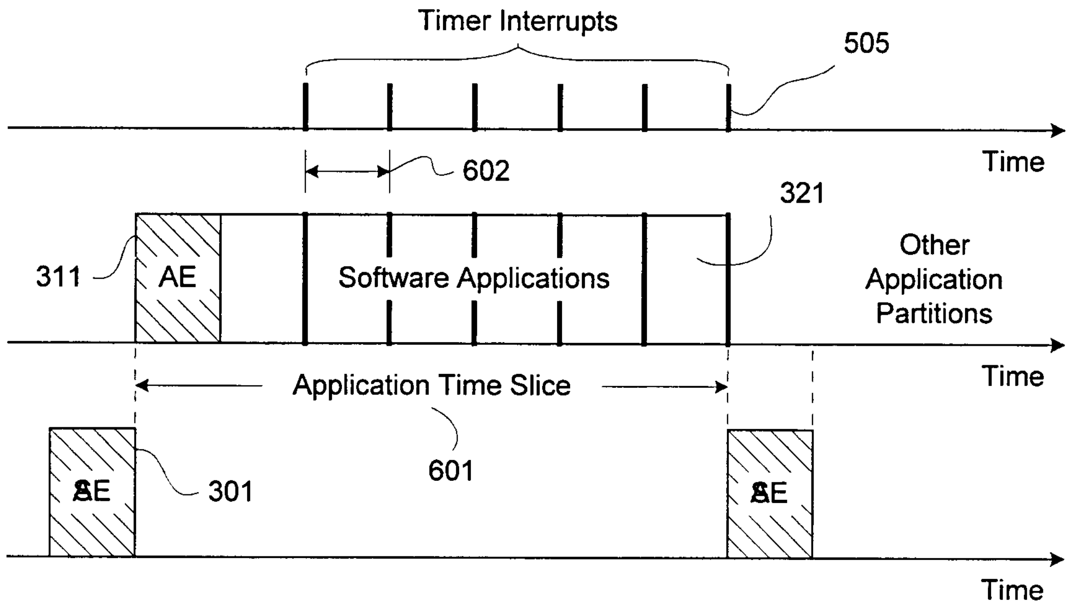

Timer Interrupts
505
Time
602
321
311
AE
Software Applications
Other Application Partitions
Time
Application Time Slice
601
SE
301
SE
Time

Figure 6