International Journal of Advanced Research in Engineering Technology & Science ISSN: 2394-2819

Email: editor@ijarets.org Volume-4, Issue-2 February- 2017 www.ijarets.org

REAL-TIME PROGRAMMING SYSTEMS

A.M. Sallam
Egyptian Armed Forces, Egypt

ABSTRACT: Real-time applications are becoming increasingly important in our daily lives and can be found in diverse environments such as the automatic braking system on an automobile, large communication systems, encryption system, security system, robotic environmental samplers on a space station, and most of the military systems. The use of real-time programming techniques is rapidly becoming a common means for improving the predictability of our technology. The discussion in this paper will be on the meaning of the Real-Time system, its definitions, and the generations of the it from the analog system till we reach the Real-Time programming using Real-Time Operating System (RTOS).
KEYWORDS: Real-Time system, Real-Time programming, RTOS.

INTRODUCTION

Real-Time (RT) system is a very important issue in the world, many systems used for many mission critical applications must be RT it is using in any applications such as our cars, mobile phones, communication system, nuclear station ,ATMs system, space craft and so on, the most important issue in the RT system it has to respond to externally generated input stimuli within a finite and specified [1], A RT system has been described as one which "controls an environment by receiving data, processing them, and returning the results sufficiently quickly to affect the environment at that time."The term "real-time" is also used in simulation to mean that the simulation's clock runs at the same speed as a real clock, and in process control and enterprise systems to mean "without significant delay". This paper is organized as follows. RT definition is presented in section 2. In section 3, the characteristics of RT system is presented, classifications of RT system presented in section 3, while problem of schedule and RT approach is presented in section4 and 5, section 6 presented RT performance, section 7 presented data structure and control logic, section 8 and 9 presented programming language and embedded Linux versus desktop Linux, section 10 presented H/W system construction, while conclusion is presented in section 11.

REAL-TIME DEFINITION

- A Real-Time (RT) system or application is one in which the correctness of depends on the timeliness and predictability of the application as well as the results of computations and logical correctness, but also on the time at which the result is produce. In other words, "A late answer is a wrong answer".
- Reactive system: continuous interaction with the environment (as opposed to information processing)
- Embedded system: computer system encapsulated in its environment, combination of computer hardware and software, dedicated to specific purpose.
- Safety-critical system: a failure may cause injury, loss of lives, significant financial loss.
- Hard RT system missing a deadline: failure of the system aircraft control, nuclear plant control, detection of critical conditions, etc.
- Soft RT system missing a deadline: undesirable for performance reasons multimedia application, booking system, displaying status information, most systems combined from both hard and soft deadlines.
- Firm deadline: missing a deadline makes the task useless (similar to hard deadline), however the deadline may be missed occasionally (similar to soft deadline).

- Generalization: cost function associated with missing each deadline.
- Mixture of hardware and software: use of special purpose hardware and architectures.
- Concurrent control of separate system components: devices operate in parallel in the real-world, better to model this parallelism by concurrent entities in the program.
- Extreme reliability and safety: RT systems are usually safety-critical.
- Deadline: is a given time after a triggering event, by which a response has to be completed (i.e. it is the time which the task should be complete).
- Schedule: that's the objective to find.

CHARACTERISTICS OF REAL-TIME SYSTEM

The real-time system should have the following two properties:

1- The functions of the real time system must be predictable (the same data input will produce the same data output).
2- Timing behavior must be predictable i.e. the system should meet the temporal constraints (e.g. deadline, response time).

Note: Predictable means that we can compute the system temporal behavior before the execution time.

To assist the real-time application the designer should be aware about meeting the three goals:

1- The system should provide the facility of the inter-process communication and synchronization.
2- The system should have fast interrupt response time.
3- The systems must do simultaneous tasks. Figure 1 illustrates the generic component of RT system and the main idea of the it.

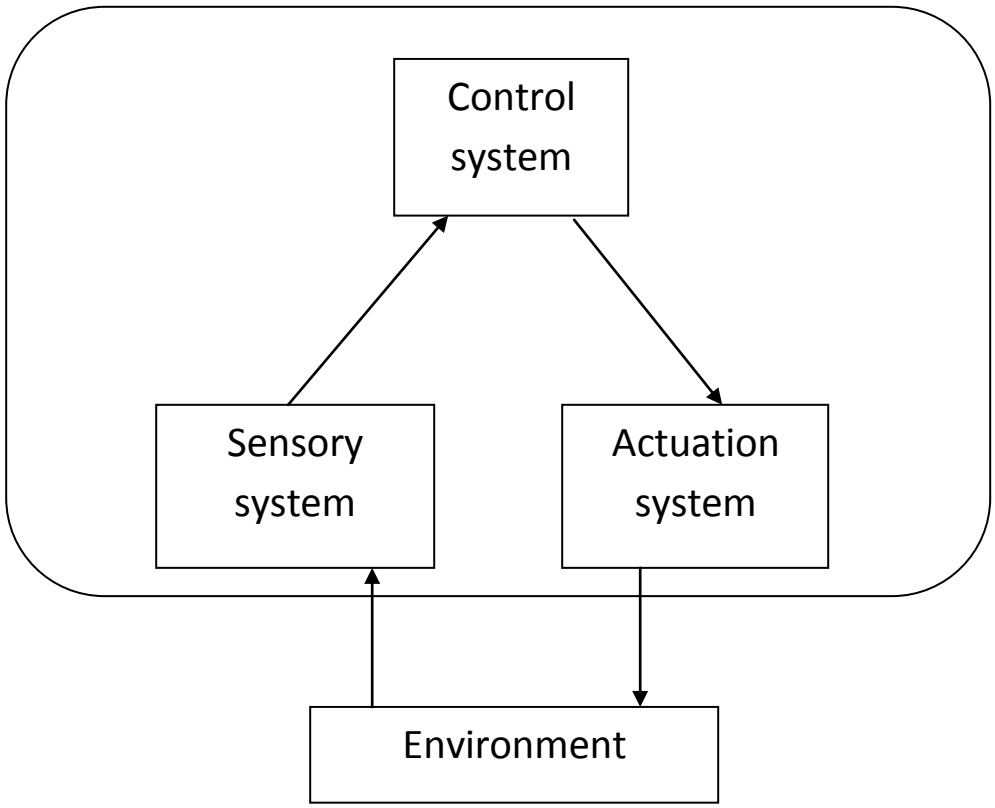


Figure 1 Block diagram of a generic Real-Time system

CLASSIFICATIONS OF REAL-TIME SYSTEMS:

We can classify RT systems as follows and figure 2 will Shows that:

1- Hardware approach:
   a- Analog circuits (vacuum tubes, analog filters, coils, capacitors …).
   b- Digital circuits (logic gates, digital filters, timers …).
2- Combined Hardware and Software (Embedded systems) approach:
   a- Microprocessor/Microcontroller.
   b- Digital Signal Processor (DSP).
   c- Field Programmable Gate Array (FPGA).
3- Parallel Processing Programming approach.
4- Software approach (Embedded Linux, Embedded Windows).
5- Software based Hardware approach by using of Graphical Processor Unit (GPU).

In the following section I will illustrate the definition and method of fourth classification Software approach that uses the Linux operating system.

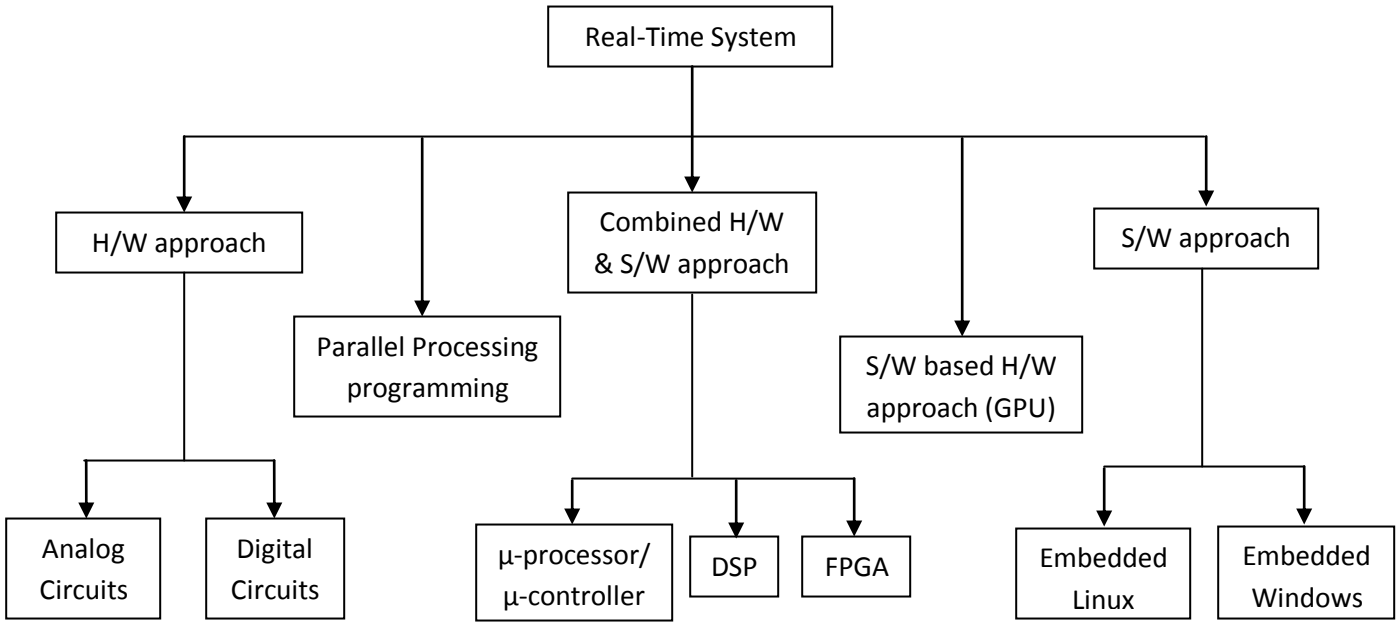


Figure 2 Classification of RT systems

PROBLEM OF SCHEDULING

The definition of schedule: it is the timetable for a project, program or portfolio. It shows how the work will progress over a period of time and takes into account factors such as limited resources and estimating uncertainty; and in RT Operating System (RTOS) defined as:

1- Part of the kernel responsible for deciding which task should be executing at any particular time. The kernel can suspend and later resume a task many times during the task lifetime.
2- The scheduling policy is the algorithm used by the scheduler to decide which task to execute at any point in time. The policy of a (non real time) multi user system will most likely allow each task a "fair" proportion of processor time. The policy used in RT/Embedded systems is more complex, and will describe in following.

SCHEDULING IN RTOS

Given a set of RT tasks and the resources in the system, task assignment and scheduling is the process of determining where and when each task will execute. For example, consider a real-time application with six tasks a, b . . . f with precedence and timing constraints as shown in Figure 3. In this figure, the vertices represent the tasks and the directed arcs represent the precedence relation. For instance, tasks a and b must complete before task d can begin, because there are directed arcs from a to d and from b to d. Each vertex has a weight associated with it which represents the time required to execute the corresponding task. The timing constraints are such that tasks e and f must complete within 31 and 16 time units, respectively, assuming that all tasks are ready to execute at time 0 subject to their precedence constraints.

Figure 4(a) and (b) shows two possible schedules for this application on a system with two processors. In Figure 4 (a), tasks b, c, and f are assigned to one processor and the remaining tasks are assigned to the other processor. On the first processor, task b executes from time 0 to 5, task c executes from time 5 to 15, and so on. Likewise, on the second processor. Since, in this schedule, task f does not complete its execution by its deadline of 16, a key timing constraint of the application is not satisfied. However, the schedule shown in Figure 4(b) satisfies the precedence and timing constraints of all tasks, and therefore, it is better suited for real-time applications. The problem of scheduling is to identify schedules like the one in Figure 4(b) given the application as in Figure 3.

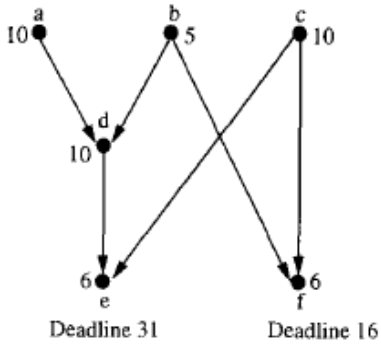
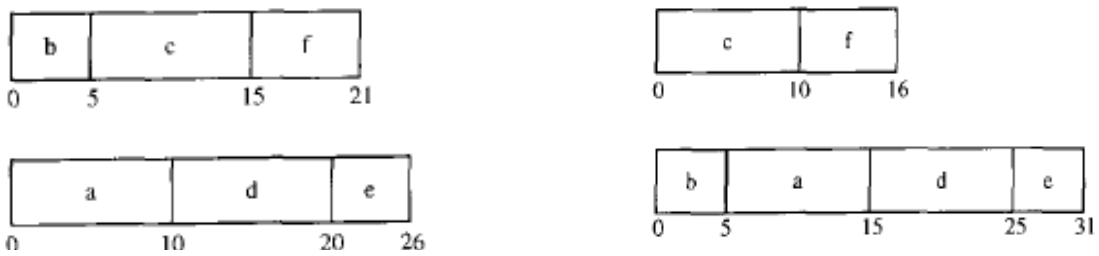


Figure 3 Example of a RT application



(a) Infeasible schedule (b) Feasible schedule

Figure 4 illustrate the difference between RT and Non-RT of a RT application

Scheduling algorithms for real-time applications can be classified along many dimensions; the first is the periodicity [2]:

1- Some scheduling algorithms deal only with periodic tasks.
2- While others are intended only for a-periodic tasks.
3- There are very few algorithms which deals with both types of tasks since the approach needed to deal with them differ considerably. Likewise, some scheduling algorithms can only handle.

The second classification of scheduling algorithms also varies significantly depending on the type of computer system they are intended for [3, 4]:

1- Some algorithms are for uni-processors.
2- While others are for multiprocessor systems. Among multiprocessor systems, the scheduling algorithms can depend on whether it is a shared-memory or a message-passing system. The type of interconnection network can also affect the scheduling algorithm.

REAL-TIME SOFTWARE APPROACH

One of the most and widely used approach is the Real-Time Software approach because it depend upon the uses of Software like C, and Ada. We will illustrate the RTOS (Real-Time Operating System) Linux. RTOS: is an operating system (OS) intended to serve real-time application process data as it comes in, typically without buffering delays. A real-time application (RTA) is an application program that functions within a time frame that the user senses as immediate or current. The latency must be less than a defined value, usually measured in millisecond; the use of RTAs is called real-time computing (RTC). An example is payroll and billing systems. In contrast, real time data processing involves a continual input, process and output of data. Data must be processed in a small time period (or near real time). Radar systems, customer services and bank ATMs are examples.

ADVANTAGES OF LINUX AND OPEN-SOURCE FOR EMBEDDED SYSTEMS

1- **Reusing of the component**:
   a- The key advantage of Linux and open-source in embedded systems is the ability to re-use components.
   b- The open-source ecosystem already provides many components for standard features, from hardware support to network protocols, going through multimedia, graphic, cryptographic libraries, etc.
   c- As soon as a hardware device, or a protocol, or a feature is wide-spread enough, high chance of having open-source components that support it.
   d- Allows quickly designing and developing complicated products, based on existing components.
   e- No-one should re-develop yet another operating system kernel, TCP/IP stack, USB stack or another graphical toolkit library.
   f- Allows to focus on the added value of your product.
2- **Low cost**:
   a- Free software can be duplicated on as many devices as you want, free of charge.
   b- If your embedded system uses only free software, you can reduce the cost of software licenses to zero. Even the development tools are free, unless you choose a commercial embedded Linux edition.
   c- Allows to have a higher budget for the hardware or to increase the company's skills and knowledge.
3- **Full control**:
   a- With open-source, you have the source code for all components in your system.
   b- Allows unlimited modifications, changes, tuning, debugging, optimization, for an unlimited period of time.
   c- Without lock-in or dependency from a third-party vendor.
   d- To be true, non open-source components must be avoided when the system is designed and developed.
   e- Allows having full control over the software part of your system.

4- Quality:

a- Many open-source components are widely used, on millions of systems.
b- Usually higher quality than what an in-house development can produce, or even proprietary vendors.
c- Of course, not all open-source components are of good quality, but most of the widely-used ones are.
d- Allows to design your system with high-quality components at the foundations.

5- **Eases testing of new features**:

a- Open-source being freely available, it is easy to get a piece of software and evaluate it.
b- Allows to easily studying several options while making a choice.
c- Much easier than purchasing and demonstration procedures needed with most proprietary products.
d- Allows to easily exploring new possibilities and solutions.

6- **Taking part into the community**:

a- Possibility of taking part into the development community of some of the components used in the embedded systems: bug reporting, test of new versions or features, patches that fix bugs or add new features, etc.
b- Most of the time the open-source components are not the core value of the product: it's the interest of everybody to contribute back.
c- For the engineers: a very motivating way of being recognized outside the company, communication with others in the same field, opening of new possibilities, etc.
d- For the managers: motivation factor for engineers, allows the company to be recognized in the open-source community and therefore get support more easily and be more attractive to open-source developers.

REAL-TIME PERFORMANCE

Real Time Performance The term "real time" is bandied about a lot, so I will define what it implies before I describe the real time features of Linux [2]. By "real time" I mean the ability to put deterministic bounds on the time the elapses between an interrupt occurring and the corresponding interrupt service routine to execute. Other items, such as context switch time and system clock granularity, are related, but not central, issues. In general, real time performance can be grouped into two broad categories Hard Real-Time and Soft Real-Time [3].

Hard Real-Time: means that a late answer is a wrong answer. If the system does not respond to an interrupt within a fixed, predictable amount of time disastrous things can happen. Some examples are the nuclear system, Intensive Care Unit (ICU) in medical application, most of the defense application, avionics, etc.

Soft Real-Time: on the other hand, doesn't have any dire consequences associated with a late answer but relies on deterministic timing of interrupt handlers to achieve top performance. It's important to remember that although the actual performance numbers are important, real time characteristics are actually more about deterministic behavior than raw speed. A good example is the sound system in your PC you can miss a few bits, no big deal, but miss too many and you are going to eventually degrade the system. Similar would be seismic sensors. If you miss a few data points, no big deal, but you have to catch most of them to make sense the data. More importantly, nobody is going to die if they don't work correctly.

DATA STRUCTURE AND CONTROL LOGIC

Data Structure:

Real-time programmers have to deal with some data structures that are normally hidden from high-level programmers. The task control block is where the CPU stores the state of the last run task so it can be restored. The semaphore (it is hardware or software flag used in multitasking systems and is defined as a protected integer variable that can facilitate and restrict access to shared resources in a multi-processing environment. The two most common kinds of semaphores are counting semaphores and binary semaphores),

is used to coordinate processes and shared resources. There are two types of semaphores: binary and counting. A binary semaphore is used to provide mutual exclusion. A counting semaphore is used when a resource can be used by more than one task at a time. The basic counting type is an integer variable that is accessed only through two basic operations, wait and signal; however, an initialize operation is also usually provided. Modifications to the integer value of the semaphore must be executed without interruption. A macro is a label that replaces a block of instructions that is used more than once, but only coded once. It differs from a subroutine in that the assembler inserts the code where the call is made rather than having a jump-to-it command. It works by text substitution and is usually faster than a subroutine but takes up more memory. A pipe is a stream of data used to connect tasks, or to provide task communication. A buffer, like a first-in-first-out buffer, can implement it. This eliminates the need to use a file to store temporary results. A pipe self-regulates its flow so that it uses less disk space than a temporary file. A script is a file of characters used for input or instructions to a program. The programmer can use it to simulate an interactive user or other I/O device. A script file could be a list of commands for a command interpreter such as a batch file. A communications port consists of a queue to hold messages and two semaphores. One semaphore controls producers, or the process that generates messages, and the other controls consumers, which are the processes that use the messages.

Control Logic (Structure):

Two basic software control structures are [4]:

1- The polling loop system. In a polling loop, the program examines each input in turn to see if an event has occurred. The program structure is a loop, and the inputs to be examined are predetermined. If an event occurs, the polling is stopped, some action is taken, and the polling continues. Controlling refrigerator temperature could be done with a simple polling loop. The temperature would be read as input and the compressor turned on or off based on the reading. If the temperature is within controlled limits, no action is taken. There are three kinds of event-driven systems: foreground/background, multitasking, and multiprocessor.

2- The event-driven system, the program loops (sometimes called a spin loop) until an interrupt occurs, at which time the loop stops and services the interrupt, and then continues. Interrupt latency is the interval of time measured from the instant an interrupt is asserted until the corresponding ISR begins to execute. Remember that an interrupt request is a request. The processor may have some critical processing to finish before it responds and services the request. Context switching time is the time the operating system takes to store the state of the processor or the contents of the registers before it begins to process another task. Because the context switching time and interrupt latency may not be constant times, making the system predictable can be a challenge for the real-time programmer.

Microcontrollers come with a monitor program that allows programmers to develop and execute software. Do not confuse this monitor with the screen monitor. The word monitor is also used for a shared data structure that contains a semaphore. A monitor for a microcontroller is a program that combines a debugger, some device drivers, and a bootstrap loader program. If provided, it is usually part of the read-only memory. The bootstrap program initializes the system by setting the registers to known states, and then it calls in or loads the rest of the required software routines. A monitor may include an assembler, which is a program that translates source code into object code, and can also produce a listing file. A linker combines one or more object code files to produce a hex file. A loader converts the hex file into an executable form called a binary file. The foreground/background system is basically a polling loop with interrupts enabled. The loop runs in the background. Only critical processing is done inside the interrupt.

REAL-TIME PROGRAMMING LANGUAGE

A lot of real-time programming is done in assembly language but the most popular language is:

1- C is popular, as well as C++ [5, 6] (it is a popular language for any programmer).
2- Forth [2] (Forth is an imperative stack-based computer programming language and environment originally designed). Although Forth is an interpreted language, it is efficient because of its stack-oriented design. Java is also being used; or rather a form of Java is being used. A language with automatic garbage collection is not a good choice for real-time because it hinders determinism (The deterministic nature of a law of the Universe does not entail the predictability of its results or their precision [7]), but there is a working group making a real-time version of Java, the Real-Time Specification for Java.
3- Ada was designed for real time and is the most powerful of those mentioned. The Ada language specification is devoted to real-time issues. The strong type checking can be turned off to increase speed by using a pragma, while "Predictability is extremely important in real-time programming, and to get it, you need to keep track of time. Response time is the time it takes the computer to recognize and respond to an external event.
4- You can use MATLAB and Simulink with Texas Instrument DSP's to get Real Time application in easy way [8].

EMBEDDED LINUX VERSUS DESKTOP LINUX

Linux operating system is used in desktop, servers and in embedded system also. In embedded system it is used as Real Time Operating System. There are so many products in the market that use embedded Linux. Embedded system requirements are very much different then requirements of desktop system [9]. The following comparison illustrates the difference between them.

In comparison	Embedded Linux	Desktop Linux
Development Environment	Linux kernel running in the embedded system product / single board computer / development board.	Linux kernel running on Desktop / Laptop.
Real Time response	Real time Linux kernel is used. Kernel response is deterministic.	Linux kernel running in desktop / laptop is not real time. Kernel response is not deterministic for response against events.
Purpose of using	Linux kernel is to perform particular function.	Purpose of using Linux kernel to many works for user.
Example of usage	Used in Video Streamer to do the function of converting MPEG4 and sending video stream on network	Used in desktop to run so many different applications.
Kernel foot print	Embedded Linux kernel footprint is less. Around 1 MBs	Desktop Linux kernel footprint is more around 100 MB.

HARDWARE SYSTEM CONSTRUCTION:

RT applications are becoming increasingly important in our daily lives and can be found in diverse environments such as the automatic braking system on an automobile, a lottery ticket system, or robotic

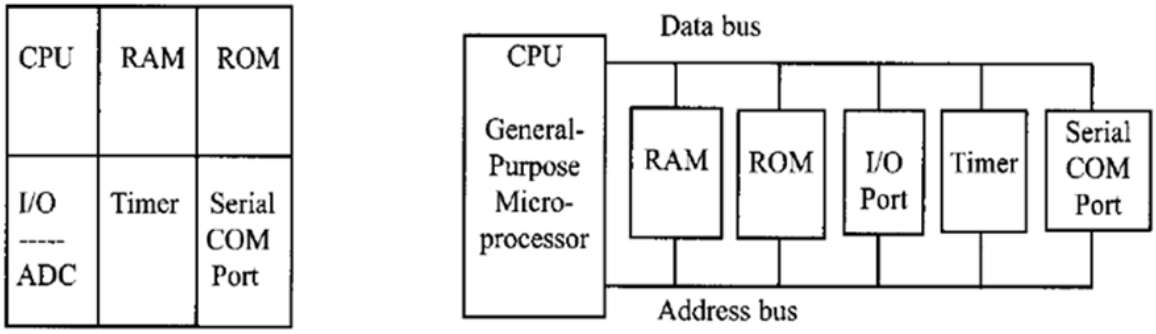
environmental samplers on a space station. The use of real-time programming techniques is rapidly becoming a common means for improving the predictability of our technology.

Real-time programmers must have a intimate relationship with computer hardware, and if there is one, the operating system. Thus, another name for real-time programming is low-level programming, and at this low level, the silicon world looks a lot different from high-level application programming.

There are two types of H/W choices:

1- Inexpensive hardware choices for controlling a real-time project include microcontrollers like the Motorola 68HC11, Intel 8051, or PIC 16F84. Microcontroller is optimized for data acquisition and control purposes. It contains a central processing unit (CPU), random access memory, read only memory, serial and parallel I/O ports, an analog to digital converter, and a timer circuit. All of these systems communicate via a data bus as in figure 5 (a), a microcontroller can be defined as a microprocessor with special hardware support.

2- A personal computer system could be used if the operating system allows access to peripheral ports. The basic requirements are input, some processing capability, and an output as in figure 5 (b).



(a) Microcontroller (b) General-Purpose Microprocessor System

Figure 5 Block diagram of the General-Purpose Microprocessor and Microcontroller

CONCLUSIONS

From the above, I think that embedded Linux has a bright future; because to the ability to run on many different processors, lack of the requirement for an MMU and extremely low cost are huge factors. Also, its popularity seems to be rising rapidly and there is a large installed base of developers rapidly gaining experience.

Of course everything has its drawbacks. The embedded Linux is very fragmented and requires more research than other operating systems to find the configuration that works best for your design. Its requirement of a 32-bit microprocessor and minimal footprint is still too big for some embedded devices. This is true of Windows CE as well and, although the two have a large overlap, embedded Linux seems to be winning the battle in terms of the number of devices that it can be adapted to. The embedded system market has been exploding in the past few years and Linux is certainly in the heart of it.

REFERENCES

1. **Stefan M. Petters**: Real-Time Systems, NICTA 2007/2008.
2. **Andrew Tucker:** An Overview of Embedded Linux, CSE585 (Design and Implementation of Digital Systems), March 2000.
3. **Radek Pelanek:** http: Real Time System Introduction, //www.fi.muni.cz/~xpelanek/IA158/.
4. **K. G. Shin, P. Ramanathan:** Real-Time Computing: A New Discipline of Computer Science and Engineering, Proceedings of the IEEE, vol. 82, No. 1, January 1994.
5. **Dennis Ludwig:** An Introduction to Real-Time Programming, the Journal of Defense Software Engineering, November 2003.
6. **Michael D. McDonnell:** The Tics RTOS Programmer's Guide, www.TicsRealtime.com, 2004.
7. **Daniel Martin:** Determinism Extended to Better Understand and Anticipate, http://www.danielmartin.eu/emailaddress.htm, January 1, 2009.
8. **Jacob Fainguelernt:** From MATLAB and Simulink to Real-Time with TI DSP's, Rice University, Houston, Texas, Connexions 2009.
9. **Embedded Craft Site:** Difference between Embedded Linux and Desktop Linux, embeddedcraft.org.