IMPLEMENTATION OF MEMORY PROTECTION IN RTEMS RTOS

Shema Mathew, Tessin Baby
PG Scholars, Electronics & Communication Engineering,
St. Joseph’s College of Engineering & Technology,
Palai, India

Ms. Juney M George
Asst. Professor, Electronics & Communication Engineering,
St. Joseph’s College of Engineering & Technology,
Palai, India

ABSTRACT

A real-time operating system (RTOS) is a key to many embedded systems and provides a software platform upon which to build applications. When we add more applications on a single computing platform, complexity of hardware & software are getting increased & analysis of adequate protection is challenging. This Project design is made with the intention of development of a Commercial-Off-The-Shelf (COTS) RTOS software in aviation systems. At present high cost RTOSs such as Lynx OS-178, HeartOS etc. are used. Instead of these, this work defines memory protection in RTEMS (Real Time Executive for Multiprocessor Systems - an open source RTOS) by the use of rtems_semaphore for i386 processor in Linux platform. It is to produce a RTEMS kernel version with a minimal set of functionalities and managers that satisfy the software requirements for various space & avionics applications.

Keywords: Kernel, Memory Protection, RTEMS, RTOS.

1. INTRODUCTION

Embedded systems research [1, 2] is to develop techniques that allow several applications or subsystems to share a common hardware platform in a safe and reliable [3] manner. In avionics, space industries & defense fields, the safety concept is of paramount importance. There is a general demand for the use and re-utilization of Commercial Off-The-Shelf (COTS) components in the design &
development of complex & hard embedded systems, such as found in space, defense & aerospace applications.

Programming parallel processing applications raises questions in the field of sharing data & cooperation among processes on different processors. In the case of distributed memory management, both data sharing and process cooperation is realized by message-passing. But the shared memory management offers processors one single address space to share and exchange data. Shared memory systems require synchronization mechanisms to prevent interferences between processes while operating on shared data.

The objective of this paper is to develop memory protection in RTEMS RTOS (an open source RTOS) with the help of semaphore for developing a Commercial-off-the-shelf (COTS) RTOS [4] software in aviation systems – MoRTOS (Moog RTEMS based POSIX kernel) is mainly concentrated for targets from PowerPC & DSP families.

2. RELATED WORKS

A kernel connects the application software to the hardware of a computer. The most basic level of control of the computer's hardware devices are given by this kernel only. It manages memory access for programs in the RAM. It also determines which programs should get access to which hardware resources. Then it organizes the data for long-term non-volatile storage with file systems on media such as disks, flash memory, etc.

A multi-programming operating system kernel must be responsible for managing all system memory of programs which is currently in use. This makes clear that a program does not interfere with memory already in use by another program. Since programs share time, each program must have independent access to memory. The memory management is an important module in any operating system since this module makes the interface between the operating system and the MMU [5] that has the responsible to verify and check if one certain user or application could access to some portion of memory. Measures have to be taken to ensure that memory being accessed is valid and that it corresponds to actual physical storage. We need to perform checks by the processor to ensure that an executing task does not access memory locations. This should be performed only when memory protection mechanisms are in place. In many embedded systems, the kernel and application programs execute in the same space. This shows the lack of memory protection.

Cooperative memory management, which is used by many early operating systems, realizes that all programs make voluntary use of the kernel's memory manager and do not exceed their allocated memory. Today we can never see this system of memory management, since programs often contain bugs which can make them to exceed their allocated memory.

If a program fails, it may cause the memory used by one or more programs will be overwritten or affected. Viruses or malicious programs may purposefully alter another program's memory. Or it may affect the operation of the operating system itself. The inexistence of a memory protection mechanism implies that any task could read or write in any place on the memory, such as data, code or stack area, of another task, leading that the execution of one task could terminate in an erroneous state. The problems caused by this unwanted access depend on the importance of the task that we have. The RTEMS version 4.10.2, which is the version adopted for the development of the RTEMS do not offer a memory management module. One design has introduced in Google Summer of Code Programme. But it is based on paging concept.

3. PROPOSED SYSTEM

This work defines memory protection in RTEMS [6] by the use of rtems_semphore [7] for i386 processors [8]. It is to produce a RTEMS kernel version with a minimal set of functionalities
and managers that satisfy the software requirements for space & defense applications. Hence a feature (memory protection) is added to the existing kernel of the RTEMS. Then bring up the modified RTEMS RTOS to POSIX standards [9] & modified POSIX compliant RTOS can be adapted to offer the application interface & functionality required by the ARINC 653 standard [10]. This is the design approach of the paper. In this, the contribution given is the addition of a feature to the existing RTEMS kernel which is none other than memory protection. The usage of a memory management module is important in memory protection since without usage of such module, memory access violations are undetectable. These unwanted accesses need to be avoided because one task could change data in the memory of another task. The lack of memory protection leads the applications to be more susceptible to errors.

We chose RTEMS kernel as it is freely available for academic purposes and because of its small memory footprint, availability of light weight kernel, POSIX compliance, ARINC 653 interface feasibility, previously used in any of the Avionics, Space & Defense domain applications and well organized source code. These features make RTEMS easy to understand, to use and to extend.

However, it lacks support for memory protection. So here, memory protection is developed with the help of rtems_semaphore. RTEMS (Real Time Executive for Multiprocessor Systems) is a free open source Real Time Operating System (RTOS) designed for deeply embedded systems that aims to be competitive with closed source and commercial products. It is developed to support applications with strict timeliness requirements, making possible for the user to develop hard real time systems.


This is the initial set-up for cross development environment to use with RTEMS. Without this set-up, we cannot move to further steps.

- Make sure that the native GNU tools are installed properly.
- Install prebuilt toolsets
- Build RTEMS 4.10.2
  - Obtain the RTEMS-4.10.2 source code & examples-v2-4.10.2.tar.bz2. Extract both in opt named folder.
  - Add /opt/rtems-4.10/bin to $PATH environment variable (executable path) in order to compile RTEMS so that the compiler can use them to build the cross-compiler.
  - Build RTEMS for a specific target (here i386) & BSP (here pc386).
- Build the same application
  - The sample application sets use the RTEMS Application Makefiles. This needs that the environment variable RTEMS_MAKEFILE_PATH point to the appropriate directory containing the installed RTEMS image built to target our particular CPU and board support package combination.
  - A folder named o-optimize is automatically generated with .exe, .ralf, .bin, .num, .o files.
  - Install QEMU(Qemulator) – a processor emulator for executing .exe file.
  - Execute the .exe file in o-optimize folder.
  - Output is generated in the QEMU window.
3.2 Memory Protection with Semaphore

Consider we have two tasks (Temperature & Pressure Chambers) with priority p1 & p2. It is connected to temperature & pressure sensors. For controlling the actuator, real time data acquisition is performed. So sensors are connected with microcontroller. Assume we have two shared sensor variables (V1, V2) for the processing of tasks. Both are stored in different locations with the help of malloc(). According to the position of actuator, we can decide which task should be processed first. Then the selected task will use the two shared sensor variables & perform data processing. So analogue values are taken from the chambers & passed to the sensors. Then task1 or task2 will use the shared variables based on the actuator position. RTEMS is the target OS platform that we use. If T1 is selected, temperature sensor values are taken for data processing with the help of sensor variables.

There is a main.c file that contains the processing details such as filtering, sampling, algorithm etc. to be used. Then that c file will get executed using shared sensor variables. After that we will get final output & is stored in another location. If both tasks are having the same priority, we can go with Round-robin algorithm otherwise FIFO (First In First Out). Here we use semaphore manager for memory protection of shared sensor variables. This helps to prevent the interference of other tasks for memory during the time of processing.ie, prevents the access of shared variables by other tasks during the time of processing. A semaphore can be viewed as a protected variable whose value can be modified only with rtems_semaphore_create, rtems_semaphore_obtain, rtems_semaphore_release directives. In the set of valid semaphore attributes, RTEMS_FIFO (tasks wait by FIFO-First in First Out) is the default attribute. So actuator position determines which task should be processed first. Fig.1 shows the block diagram for memory protection with semaphore.

![Fig. 1: Memory Protection with Semaphore](image)

4. RESULTS

Here sensor_variable1 & sensor_variable2 are taken as the shared sensor variables. There are two tasks T1 & T2 with same priority. First T2 tries to access the shared sensor variables. But sensor variables are locked with the semaphore & they are used for T1 because of RTEMS_FIFO.
The First In task is T1 (depends on actuator position). When T1 completes the process, semaphore will be released and T2 is able to take the shared sensor variables. If T1 tries to access shared sensor variables at that time, sensor variables can be used by T2 only. So T1 cannot access them. In the QEMU window, we can see sensor_variable and my_variable. These are chosen by the actuator position and sensor_variable represents T1 and my_variable represents T2. So it is clear that, when T1 is chosen by the actuator, shared sensor variables are not available for T2 & vice versa. Final output will get stored at a particular memory location. Fig.2. shows the memory protection with semaphore.

5. CONCLUSION

The goal was to add one feature (memory protection) to the existing RTEMS RTOS with the help of semaphore for developing a Commercial-off-the-shelf (COTS) RTOS software in aviation systems. This paves the way for the design & development of MoRTOS. It is mainly concentrated for targets from PowerPC & DSP families. The LINUX version used is openSUSE 13.1 (Bottle) (x86_64) & host architecture version is x86_64. For developing an embedded system we need target & an operating system. The target used here is i386 & OS is RTEMS with version 4.10.2. The BSP used for i386 is pc386. The major contribution that we have given is the memory protection support for RTEMS kernel.

REFERENCES


