Real-time programming platforms in the mainstream environments

*Husein A. Al Ofeishat and **Ahmad A. Al-Rababah,

*Al-Balqa Applied University, Computer Engineering Department.Jordan
**Applied Science University, Software Engineering Department, Faculty of IT,
Amman, Jordan 11931

Abstract

The functionality of a real-time system is divided into separate run able entities, referred to as tasks. The term real –time system refers to generally-rendered system as processing unit having a set of input & output. There exists a group of notions closely intertwined when speaking of real-time system. The term embedded system refers to equipment or devices not classified as standalone general–purpose computer architectures but whose functionality resembles that of dedicated computer architectures. while stand alone general-purpose computer architectures designed to run a commonly used set of application ,there is large diversity of embedded system architectures, where as essentially one general- purpose computer architectures. In this paper, a description of existing mainstream general-purpose real-time programming platforms is presented.

Keywords: Real-time systems, embedded system, QNX, Neutrino, Java.

1. Introduction

The system functions by performing its tasks and responding to external asynchronous events. the time leg between event occurrence and realization of the required behavior is bounded .the logical correctness of a real-time system is based on both the correctness of the output and the ability to satisfy the repined time bounds. The term real-time operating system refers to general-purpose operating system with imposed response-time constrains. Failure to satisfy the time constraints results in system malfunction [1,9].

Depending on the hardware architecture, the system may be able to support multitasking, that is the process of scheduling and switching tasks, making use of the hardware capabilities or emulating concurrent processing using the mechanism of task context switching [2]. The real-time system grants access to system recourse that the tasks may use. The access policy may require that specific recourse be used by only one task at a time. The access policy is controlled by synchronization mechanisms. With respect to tasks, real-time systems provide the functionality of scheduling and synchronization.

Synchronization is delegated to the kernel, the core entity of the system. The kernel is responding for ensuring the recourse allocation policy realization of context switching [2,10]. Scheduling is delegated to the scheduler, one of the parts of the kernel. The scheduler is responsible for determining which task to run next multitasking system. The algorithm that is used in this process is referred to as the scheduler algorithm. The non-preemptive scheduler, also referred to as cooperative scheduler, realizes the scheduling policy which requires that the processor be explicitly freed up by the tasks. Higher-priority task waiting for execution is queued until the currently running lower-priority task is complete.

The preemptive scheduler realizes the scheduling policy which requires that’s the currently running

lower-priority task be suspended when higher-priority task waits execution.

the other tasks changes as tasks are released and completed.

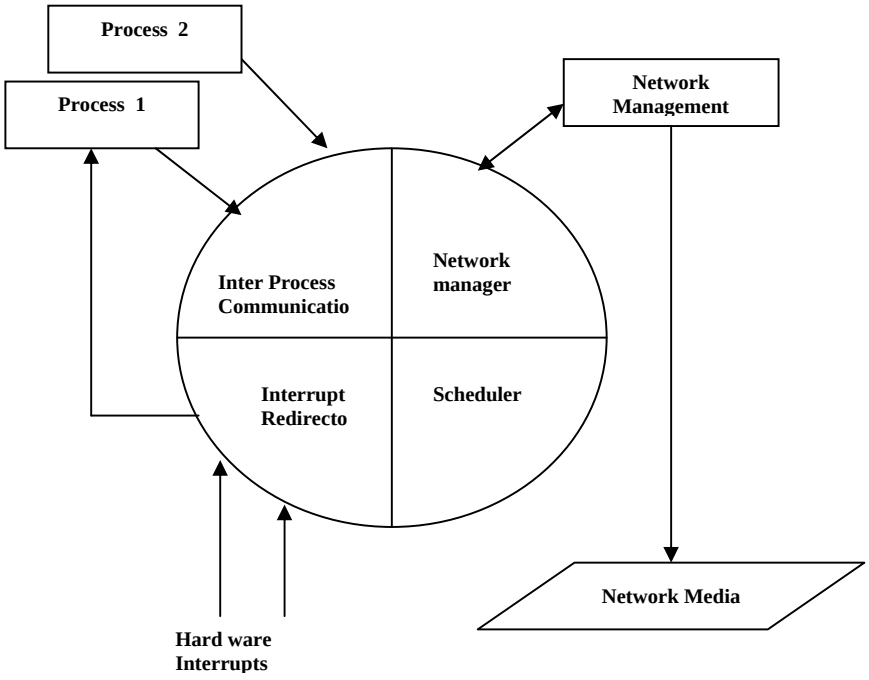

Fig.1 the QNX Microkernel

The fixed-priority scheduler realizes the scheduling policy referring to the priorities assigned to tasks at system design stage. In the fixed-priority scheduling policy, the priority of each periodic task is fixed with respect to other tasks. The rate-monotonic algorithm implements the fixed-priority policy and is used in scheduling periodic and independent tasks. The algorithm is preemptive and assigns higher priorities to tasks with shorter periods.

The algorithm is stated optimal by the rate-monotonic theorem. The priority mechanism is often supported by the hardware and exhibits latency by prioritizing the interrupt handling system. The dynamic priority scheduler realizes the scheduling policy allowing for run-time modification of the priorities assigned to task at the system design stage. In contrast to fixed-priority algorithms, in dynamic-priority schemes the priority of each periodic task with respect to that of

The earliest-deadline-first algorithms used deadlines as criterion for taking scheduling decisions rather than execution time. The task that is ready for execution and has the earliest deadline is assigned the higher priority [3].

Strictly fixed-priority schedulers exhibit priority inversion problem. Contemporary fixed-priority solutions account for auxiliary dynamic priority assignment protocols resolving this problem.

2. Real-Time Programming Environments

The ready-made real-time system solution usually comprises a microkernel realizing the scheduler functionality. The microkernel is also assigned the functionality of the recourse allocate and the synchronizer. This design principle is followed by the QNX/Neutrino real-time operating system. A fairly original solution is the real-time extension to the Java Virtual Machine which, together with base platform components,

constitutes a portable and reliable real-time general-purpose programming environment.

The QNX/Neutrino operating system design philosophy aims at implementation of the POSIX standard independent of the UNIX system structure [4,9]. The POSIX features that are not implemented in the microkernel are provided by optional process and libraries. The Neutrino solution realizes the microkernel architecture for the sake of better scalability to a wider range of applications. It provides multitasking, threads, priority-driven, Preemptive scheduling and fast context- switching. [5].

threads throughout the entire system is handled by the kernel [5]. The process management as well as inter process communication functionality is implemented in the microkernel. This is depicted in Fig.1.

In addition, the functionality spans low-level network communication and first-level interrupt scheduling. The scope of implementation of the POSIX standard spans the range of the thread synchronization primitives, scheduling service using standard algorithm and standard set of timer services. The kernel process is designed not to be scheduled for execution. The kernel process is executed only when being explicitly called by thread or in response to a hardware interrupt.

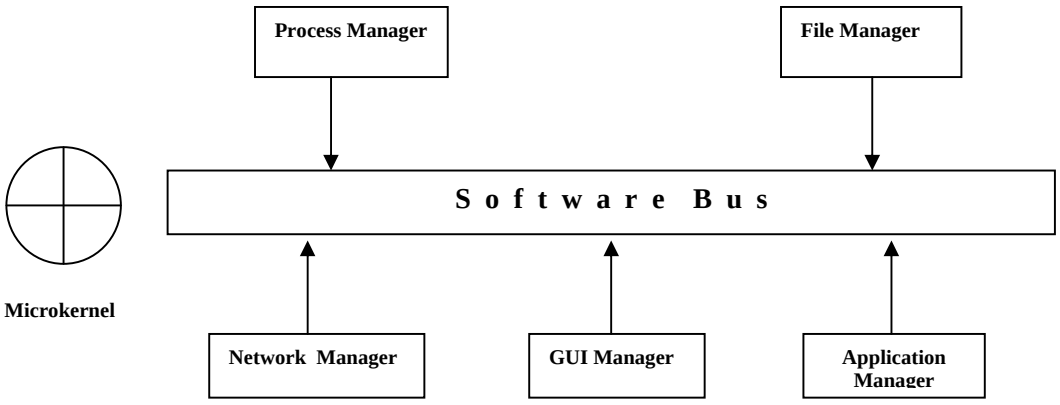

Fig.2 Functionality delegation in QNX Neutrino [6].

The architecture of the QNX/Neutrino system allows for tailoring its configuration involving projects consisting of a few modules as well as complex solution. The microkernel is designed to manage process treating them as a group of cooperating tasks rather than introducing heretical dependencies. The tasks interact with each other through the coordinating kernel. The architecture assumes message-based inter process communication as the fundamental design principle. The routing of all messages between all

The services of the QNX/Neutrino system are realized as standard processes. The microkernel functionality is dedicated to fundamental scheduling and synchronization services solely. This design philosophy, depicted in Fig.2, effectively eliminates the border between a user application and a system application, allowing for better and more comprehensible extensibility of standard system structure. Since the device drives also follow this design principle. They may be developed and debugged like any other application

[6]. Neutrino is fully preempted able, even while passing messages between processes. It resumes the message pass where it left off before preemption. The functionality of the Neutrino microkernel is designed to span lightweight processing, leaving out the heavy weight processing to external processes. Neutrino and process manager may be configured to realize concurrent environments as a group of cooperating thread , a group of single-thread cooperating processes or a group of multi-thread cooperating process . The money management unit mechanism is provided as means of memory protection between the processes [6].

The message passing mechanism is the fundamental means of inter process communication used in the QNX/Neutrino operating system. Message is realized as chunks of bytes passed from one process to another. The data contained in the message is interpreted only by its sender and its receiver. The message passing mechanism is used as means of synchronizing exaction of process. Processes are viewed upon as a group of entities each of whose changes its state as it sends or receive message .The changes of state and process priorities are the criteria by which the Neutrino kernel schedules processes. The aforementioned mechanism is used in a similar way as means of synchronizing exaction of thread. A scheduling decision is made whenever that execution state of any threads changes. Threads are scheduled globally across all processes. The Neutrino kernel realizes the functionality needed to support threads, message passing, timers, interrupt handlers and basic synchronization mechanisms [6]

Even thread is assigned a priority. The scheduler selects the next thread to run by looking at the priority assigned to even thread this is ready to be executed. The thread with the higher priority is selected to run. Each thread is assigned one of the possible 32 priorities. The priority assignment mechanism incorporates priority inheritance. The QNX-Neutrino operating system implements FIFO, round-robin and adaptive scheduling algorithms, using them interchangeably to run the system threads. The POSIX-standard thread-level synchronization primitives, including mutual exclusion locks and semaphores, and provided. The implemented interrupt mechanism supports priority assignment and preempting. The interrupt latency is minimized by maximizing the time with in which the interrupts are fully enabled [6].

3. Java

The Java environment process attributes that make it a powerful platform to develop embedded real-time applications. The most crucial is dynamic class loading and built-in multi-threading with language-level synchronization mechanism. On the other hand, the behavior of Java standard synchronization mechanisms is not guaranteed to be deterministic, which is mandatory in real-time applications. A number of solutions have been developed to port the standard Java platform to use in embedded real-time programming. The major solution is developed as the Real-timer specification for Java. The real time specification augments the existing standard Java platform with the real-time functionality [7]:

1- Real-time thread is provided whose scheduling attributes are more carefully defined than is the scheduling for ordinary Java threads.
2- Tools and mechanisms are provided to allow writing program not using the Java standard platform automatic garbage collection mechanism.
3- The asynchronous event handler class is provided to allow for programming interrupts.
4- The mechanism of asynchronous transfer of control is provided to support control flow change between cooperating threads.
5- The mechanism object memory allocation control is provided.
6- The mechanism of direct memory access is provided.

The limitation of the standard Java platform involving memory management sandbox is addressed by including mechanisms allowing byte-level memory access. This solution is an indispensable approach as the possibility of inserting system native calls to Java programs reduces the programs reduces the program portability and is not as secure as bytecode.The Raw Memory Access physical memory, mapping objects into RAM or flash memory, manipulating virtual memory, implementing device drives and access to memory-mapped devices. The functionality if the thread class is extended for real-time applications by introducing the Real Time Thread class. The extended thread class cooperates with scheduler class which may implement an arbitrary scheduling algorithm other that the base algorithms, which is preemptive and priority-based [7].

The real-time specification for Java (RTSJ) requires that the priority-based scheduling be implemented as it is universally used in commercial real-time systems. Moreover, priority-based scheduling is implemented also by the standard java virtual machine (JVM).The 10 possible task priorities implemented in the standard JVM are extended to 28. The specification requires that the introduced priorities be scheduled using the strict fixed-priority preemptive scheduling algorithm. The specification indicates the priority inheritance protocols as the default for locks between real-time threads, allows priority ceiling emulation protocols for priority inversion resolution and provides for other protocols [8] .

The RTSJ provides room for implementation to support other schedulers. The specification does not define the way the new schedulers are integrated with the system. It only provides the possibility that an implementer may design alternate and defines scheduler interfaces that are general enough to support a wide variety of scheduling algorithms. The most convenient way to implement real-time task in Java is to extend the thread class to support real-time functionality. This design approach is accounted for by the specification. The standard Java virtual machine monitors exhibit the priority inversion problems. These maybe resolved by means of the priority inheritance and priority ceiling and protocols. The priority in inheritance protocols does not prevent deadlock when a chain of blocking is formed. The priority ceiling protocols avoid this by assigning a priority ceiling to each semaphore associated with a critical section, but it's more complement to implement. The standard Java monitors are not guaranteed to be deterministic in respect of the queuing policy [7,8]

The specification extends the Symantec of the synchronized key word to avoid the priority in versions problems and require that the queues be priority scheduled. The monitor control super class abstract monitor control policy and the monitor policy is implemented by its subclasses. The specification is complaint with the standard Java virtual machine as task synchronization is naturally ported to real-time by the synchronize statement extension. Many real-time systems are event-driven, taking actions in response to event occurrences in handler routines. The particular design in which the event handlers are implemented as separate threads is maintainable and comprehensible. Moreover, the handler threads may be taken account for by the scheduler algorithm similarly as other threads. Speaking of the standard Java Virtual Machine. This approach is inefficient. In terms of the real-time system latency. The time lag between an event occurrence and execution of the corresponding handler may appear long because of the necessary resource allocation associated with thread creation [7,8]. An even is not synchronized with the execution of a particular task and may be generated by any other task or by the hardware. Synchronous exceptions allow a thread to be synchronously interrupted by another.

The real-time specification introduces two classes, one of which supports the event functionality, the other the functionality of an event

handler. Objects of the A sync Event class resemble a POSIX single or a hardware interrupt and are associated a set of handlers. Objects of the A sync Event Handler class are associated a scheduling parameters object that control the handler execution [7, 8]. Synchronous event handlers are an attempt to capture the advantages of creating threads to service evens without taking performance penalty. The standard Java platform has efficient mechanisms for handling user – interaction events, but lacks a general-purpose mechanism for associating events that happen out side the Java environment with method invocation inside the environment. The RTSJ introduces happenings as a pathway between events outside the Java platform and asynchronous event handlers. A asynchronous transfer of control is mechanism that lets a thread through an exception into another thread throw exception in to another thread and is an improvement over the standard Java Virtual Machine thread. Interrupt mechanism. The standard Java Virtual Machine Garbage collection mechanism (GC) is a separate thread that is executed when the system is being idle. Once, the garbage collection is started, it must be exacted until completion, and must not be stopped or preempted at the risk inconsistent state of the of the system heap.

The real-time equivalent of the standard GC mechanism is scheduled as other tasks, may be preempted by a higher-priority task. Unbounded garbage collection pauses must not take place in real-time applications [7, 8] . one of the solution provided by the real time specification of a pre allocation memory pool for real-time object in depend it of the recourse allocation policy inherent to standard Java virtual machine . The specification opted for not relying on special implementation of the garbage collection mechanism to resolve the secluding problems that occur .rather it is assumed that a particular implementation of the real-time Java virtual machine may introduce an arbitrary GC mechanism. In turn, the specification introduces new memory mechanism that is never delayed by garbage collection. The first tool for avoiding garbage collection is no-heap, real-time threads. The threads of this class are not allowed to access memory in the heap. Since there is no inter action between no-heap threads and garbage collection or compaction, no-heap threads may preempt the garbage collector with out waiting for the garbage collector to reach a consistent state. Ordinary thread and heap-using , real-time thread maybe delayed by garbage collection while creating object in the heap and are preempted by the garbage collector until it reaches a consistent state if activated while the Garbage collector is running. No-heap, real-time, threads are not effective by these timing problem [7,8] . by it self support, for no-heap , real-time threads is in insufficient as it restricts the thread classes to elementary data type. The specification defines two new types of memory that are free from garbage collection. These are [7,8]:

1- Immortal memory which contain object that are never garbage collected. Object allocated with in immortal memory remain in use until the JVM terminates .the memory is also shared by all other thread.

2- Scoped memory which has specified lifetime. object allocated from a memory scope stay allocated until the scope no longer active .most the scope becomes in active, all the object stored with in it are freed .the memory area abstract class support scope and immortal memory .the garbage collector class provides methods for getting information about the GC mechanism behavior, including information about the introduce latency and recourse reclamation rate. Immortal memory has its counterpart approaches in the large class of real-time systems that allocate all resource at single initialization phase and subsequently run indefinitely without allocating or freeing any resources. The immortal memory mechanism there fore resembles the C or a assembly language programming paradigm. Scoped memory functions like a stack for object. Upon entering a memory scope, the thread starts allocating object beginning from that scope. it continues allocating object until it enters a nested scope or exit from the scope.

Once the threads exit the scope , the object allocating with in the scope are no longer accessible and the JVM is free to recover the memory use [7,8].

4. Comparative Analysis

The main stream real-time QNX programming platform implement a set of the POSIX standard, currently dedicated to C and Ada programming languages. The real-time specification for Java is an extension of the existing standard java programming platform. The POSIX-dedicated real-time programming platforms are observed to exhibit better efficiency than the counter part Java solutions due to the inherent binding to system-closed C/Ada code. Java solutions in term offer stronger type semantics and better program execution security. Both QNX and Java real-time platform aim at portability of the provided functionality. In QNX, this is achieved through POSIX compliance. Java is portability /oriented by its natures. the standard POSIX implementation solutions is more suitable for static solution , in with the number of tasks present in the particular real-time system in known in advance this due to the lack of the dynamic thread creation facility . In QNX, the portability of the POSIX standard is extended by the architecture of the system which allows for the dynamic creation and registering of new modules, in depend it of how closely they integrate with the system. The real-time Java platform is suitable for dynamic solution, as dynamic thread creation is achieved through object creation. The core synchronization mechanism of the QNX/Neutrino system is message passing. The real time Java platform achieves synchronization mainly through monitor classes and the synchronized statement. The message passing mechanism may appear more difficult to handle than the monitor based solutions. The main structuring unit of the C programming language is functions. The modules them self are not so well formalized as are the classes in the Java programming language. The signaling synchronization mechanism may thus appear more suitable for advanced programmers. The real-time Java platform offers then inheritance class and interfaces hierarchy to logically group task.

The package mechanisms result in more formal structuring of the solutions. In the POSIX implementation, the mechanism responsible for asynchronous transfer of control is implemented by means of the message passing mechanism. The real-time Java platform integrates the asynchronous events handling with the exception mechanism.

5. Conclusion

The main stream real-time QNX programming platform maybe regarded as reference POSIX implementation of a real-time system. The QNX/Neutrino microkernel separate the core scheduling and interrupt handling functionality from the higher level process, allowing for there dynamic creation and registering. An important aspect indicating the future development of the QNX real-time system is its integration with both the C and Java programming languages, the later being augmented with a dedicated virtual machine. The real-time specification for Java appears as attractive in view of real time programming platforms due to the strong object – orientation and semantics of the Java programming language.

However, certain limitation of the specification may be the partial process of integration of the real-time enabled Java with the standard Java platform classes.

6. References

[1] P.A. Laplant, Real-time Systems : Design and Analysis, Ontario, NJ : John Wiley & Sons,2004.

[2] R. Grehan, Moot, and I.Cyliax, Real-time Programming : A Guide to 32-bit Embedded, Ontario,MA : Addison Wesley, 1998.

[3] M. Joseph, Real- time Systems : Spécification, Vérification and Analysis , London, England : Prentice Hall Int., 1996.
[4] M.G. Harbour, ‘ Real-time POSIX : An Overview ‘ VV Conex 93 International Conférence, Moscu, Spain, June,1993.
[5] QNX Software Systems Ltd., QNX System Architecture, Ontario, Canada : QNX Software Systems Ltd., 1997.
[6] QNX Software Systems Ltd., ONX Neutrino Real-time Operating System : System Architecture, Ontario, Canada : QNX Software Systems Ltd ., 2002
[7] P.C. Dibble, Real-time Java Platform Programming, CA : Prentice Hall PTR,2002.
[8] M.Higuera-Toledano, V. Issarny,M.Banatre, G.Cabillic.J. Lesot, and F. Parain,’ Java Embeddede Real-Time Systems : an Overview of Existing Solutions ‘, Third IEEE International Symposium on Object-Oriented ‘Real-Time Distributed Computing , NEWport Beach, California, arch, 2000.
[9] A. Silberschatz, P. Baer Galvin, and G., Gagne Operating System Concepts with Java , 2006)
[10] C. Venkateswara Penumuchu. Simple Real-time Operating System: A Kernel Inside View for a Beginner, 2007)