LynxOS
RTOS (Real-Time Operating System)

Stephen J. Franz
CS-550 Section 1
Fall 2005

1 Summary

LynxOS is one of two real time operating systems (RTOS) developed and marketed by LynuxWorks of San José, California. LynxOS is a mature operating system that was originally developed in 1988. LynxOS and LynxOS-178 were designed for systems where absolute determinism, specifically hard real-time performance, is required. The focus of this paper is the list of LynuxWorks design goals and the approach used to develop LynxOS. Of special interest is the LynuxWorks-patented approach designated as kernel threads and priority tracking. As a final wrap up, we look at the overall success of LynuxWorks and the LynxOS.

2 Table of Contents

3 Main Body

Goals of Real Time Operating Systems (RTOS)

Real time operating systems need to respond to events in a timely and predictable manner. By definition, real time means that missed or late responses by these systems constitute a failure. For hard real-time, a failure would likely lead to catastrophic consequences up to and including loss of human life. This is the case for aircraft collision avoidance, anti-lock brake, pacemaker and anti-missile systems. For soft real-time applications like communication switching systems and streaming audio or video, predictable response is required but occasional failures can be tolerated.

Who is LynuxWorks and what is LynxOS?

LynxOS and LynxOS-178 are embedded real time operating systems marketed by LynuxWorks of San José, California. LynxOS, the first real time operating system developed by LynuxWorks, was introduced in 1988. LynxOS-178 was built on the LynxOS framework but was developed for systems seeking to meet the DO-178B standards. For the sake of this paper, it is sufficient to state that DO-178B is a standard that provides a means of certifying new aviation software. Additional details regarding the DO-178B standard are beyond the scope of this paper.

The balance of this discussion focuses exclusively on the LynxOS product. It should be noted at this point that much of the available information about LynxOS came directly from the LynuxWorks web site or from documents and articles authored by LynuxWorks representatives. While this should not have an impact on the available facts, opinions in some cases could prove to be somewhat biased.

LynxOS Design Goals

Four design goals outlined by Vik Sohal, LynuxWorks Technical Sales, are summarized below:

1. The operating system kernel had to be preemptive and reentrant. This was important so time-critical tasks will execute promptly.

2. The kernel supports multithreading. User programs, device drivers and other kernel services can create their own tasks that are called kernel threads. More detail on kernel threads including scheduling and advantages are provided later in this paper.

3. LynxOS uses a processor's page memory management unit (MMU) to provide each instance of a user process its own protected logical address space. The MMU also protects the kernel by placing it in a separate address space.

4. LynxOS utilizes Unix and POSIX APIs, allowing:
    - a variety of Unix programs to be ported to LynxOS and
    - offering a shallow learning curve for those programmers already familiar with the interface

LynxOS Approach

In order to achieve the first two goals (prompt execution and multithreading), LynxWorks developed and patented techniques known as kernel threads and priority tracking. To address goal three (memory protection), LynuxWorks utilizes the processor's page memory management unit (MMU). For the last noted goal (portability and less steep learning curve), LynxOS utilizes Unix and POSIX APIs. Each of these approaches is discussed below.

Process Management

In order to explore the techniques known as kernel threads and priority tracking we need to understand the issues associated with task scheduling and execution.

The basic LynxOS scheduling entity is the thread and LynxOS scheduling is preemptive, reentrant and based on one of three selected scheduling policies:

- First-In First-Out
- Round Robin
- Priority Based Quantum – Lynx proprietary policy that is similar to round robin but includes a configurable time quantum for each priority level.

A primary issue with task scheduling is known as priority inversion. As shown in figure 1, priorty inversion is the situation where a high priority task is running but is preempted by an asynchronous interrupting device for a lower priority task. Priority inversion can also be seen as hardware interrupts or kernel processes that steal cycles from high priority tasks and degrade an application's ability to meet real-time deadlines. In general, routine processing in most systems can lead to problems where high priority processes find themselves constantly interrupted by hardware events

Asynchronous interrupting devices are most any unit that generates a requested or unsolicited signal that supplies the system with information. Examples of these include, but are not limited to: network interfaces, console displays, keyboards, disk controllers and external timers.

In order to understand how LynuxWorks deals with priority inversion, it is important to understand that LynxOS is POSIX-conformant and that the basic LynxOS scheduling entity is the thread. This means that threads are the running entities of LynxOS.

The operating system kernel is a large, monolithic program that seldom offers predictable real-time response. Further, most operating systems make a strong distinction between internal operation and the functioning of user processes. This means that the operating system kernel can supersede user processes and delay routine processing.

By the LynxOS approach, the driver's interrupt handler does a minimum of work and signals the kernel thread that interrupt-related data is available. LynxOS then treats these threads like normal user threads, with software rather than interrupt priorities. If LynxOS implemented kernel threads by themselves, the issue of priority inversion would still exist. As shown in figure 2, by incorporating a technique known as priority tracking, kernel threads can process interrupts but, at the same time, will be scheduled along with user threads based on an “appropriate” thread priority.

The previous statement fails to explain how an appropriate thread priority is determined. This determination is a function of priority tracking. LynxOS is divided into 256 priorities. These priorities are further subdivided to make 512 priorities. This includes the original 256 plus a half step above each of the original priorities. Kernel threads begin their existence with a very low priority (usually 0) as created by a driver. When a user thread opens the device, the kernel thread promotes its own priority and "inherits" the priority of the user thread opening the device. If another user thread of higher priority opens the device, the kernel thread bumps its priority up to match the other thread; when the I/O is complete the kernel thread returns to the next pending thread's priority level, or to its starting level. With the changes, we can see that the kernel thread tracks or follows the priority of device that calls it. The following example demonstrates how the priority of the kernel thread tracks the user threads that opened the device and cause the kernel thread to be created.

Example:

	Kernel Thread Priority
Kernel Thread initial creation	0
User thread (priority 10) opens device - kernel thread promotion	10
User thread2 (priority 20) opens device – kernel thread promotion	20
User thread2 completes – kernel thread demotion	10
User thread3 (priority 60) opens device – kernel thread promotion	60
User thread4 (priority 30) opens device – kernel thread remains unchanged	60
User thread3 completes – kernel thread demotion	30
User thread4 completes – kernel thread demotion	10
User thread2 completed – kernel thread demotion	0

As the kernel thread goes about servicing any interrupts associated with the device, it does so in step with the user thread consuming (or producing) the data. The kernel thread priority increases as higher priority user threads open devices thus causing kernel thread to be created. Likewise, the kernel thread priority decreases as the associated user thread completes.

Address Space Protection

As noted above, LynxOS had a design goal to utilize a processor's page memory management unit (MMU) to protect each processes logical address space. The MMU also protects the kernel by placing it in a separate address space.

Most competing RTOS products do not offer the flexibility and memory protection afforded by a complete thread and process model, and rely on unprotected tasks running in a single flat address space. (LynuxWorks (2005d))

The MMU is used to physically isolate processes from each another so that they cannot trample on each other's memory. The MMU translates the virtual addresses referenced by the thread into physical addresses. If a process attempts to address a page that is not currently mapped to it, the MMU generates an exception.

The MMU also allows the process' memory regions to grow and shrink in order to meet changing needs of the executing thread. With this, each process has its own protected address space and can communications between processes through kernel services.

Utilization of Unix and POSIX APIs

As the final of the four goals note, LynuxWorks wanted to insure the portability and usability of LynxOS. In order to achieve portability, 90% of LynxOS is written in C. By utilizing readily available interfaces, the development learning curve is substantially flattened and allows programmers that already know the APIs to quickly become productive.

LynuxWorks Success

LynuxWorks is a privately held interest with 75 employees and sales of $16.8 million for the fiscal year ended April 2005. The fact that LynxOS has been available for more than 16 years and by review of the impressive list of customers shown in table 1, it can be inferred that LynuxWorks products are both commercially and economically successful.

Conclusion

LynuxWorks established four design goals including prompt execution, multithreading, memory isolation and portability/usability. Development and implmentation of kernel threads along with priority tracking and utilization of POSIX thread scheduling allows LynxOS to achieve the first and second goal. The third is achieved through the usage of MMC and the fourth is fulfilled by the fact that LynxOS was primarily developed using

C and by adhering to POSIX standards. Together these steps make LynxOS a real time operating system that is utilized by an impressive list of customers across multiple industries.

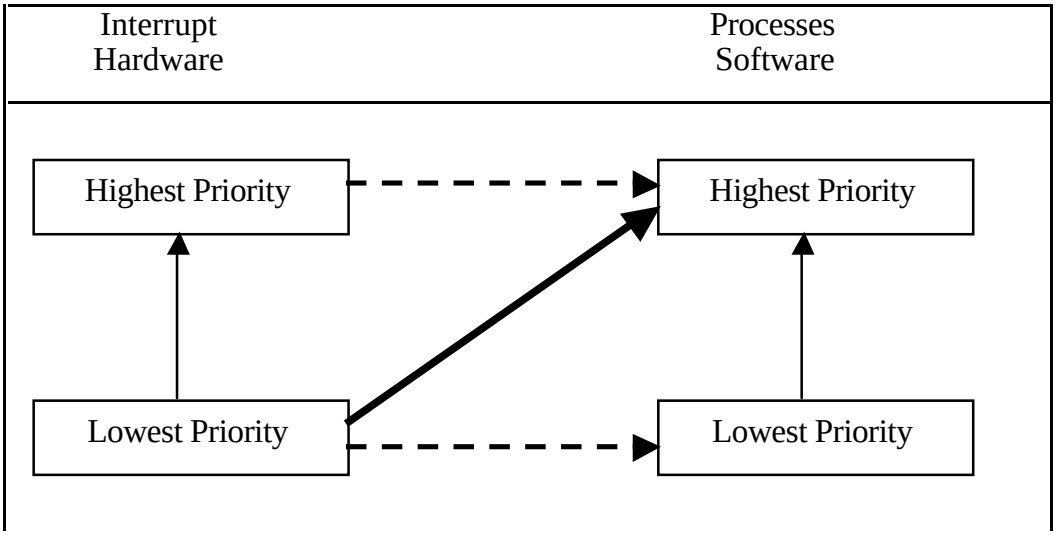
4 Figures and Tables

Table 1

Partial list of LynxOS/LynxOS-178 Customers and Applications

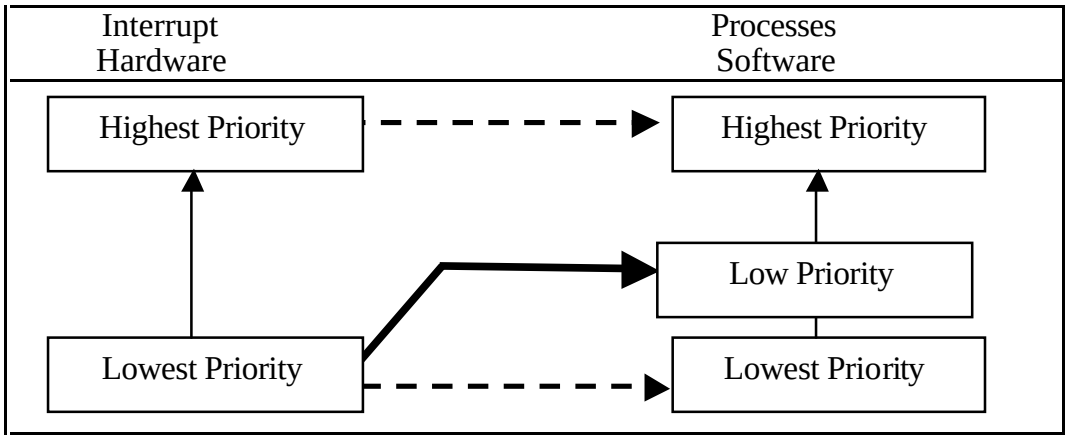
USAF – KC-135
Boeing 777 - cabin services system
NASA SLR2000 Satellite Laser Ranging System
Bombardier Challenger 300 Flight Display
Shipboard Self-Defense System
StarWorks Video Networking Software
Paradise Datacom Satellite Data Modems
Lucent – MultiMedia Comminications eXchange Server (MMCX)
Viking Power Plant Control Room
U.S. Mail Sorting by Scio Systems
CDS' M6000 Data Acquisition System (Jet Engine Vibration analysis systems)

Figure 1



In the above scenario, data is being requested for low priority process. Without kernel threads, hardware interrupts would trigger when the data becomes available. Since hardware interrupts run at higher priority than processes, the interrupt will pull resources from the high priority time-critical process.

Figure 2



In the above scenario, data is again being requested for a low priority process. With kernel threads and priority tracking, the device is opened and a kernel thread is created and scheduled with a priority that is consistent with the requesting process. With this, the kernel thread will be processed at an appropriate time as determined by the scheduling rules without pulling resources from the higher priority time-critical process.

5 Bibliography

Kevin M. Obenland (2001), "POSIX in Real-Time"; URL:
http://xtrj.org/collection/posix_rtos.htm

LynuxWorks (2005a), "Partitioning Operating Systems Versus Process-based Operating Systems"; URL: http://www.lynuxworks.com/products/whitepapers/partition.php

LynuxWorks (2005b), "LynxOS RTOS 4.0 Feature List"; URL:
http://www.lnxw.net/rtos/lynxos40features.php

LynuxWorks (2005c), "What is DO-178B"; URL:
http://www.lynuxworks.com/solutions/milaero/do-178b.php3

LynuxWorks (2005d), "The LynxOS 3.0.1 Performance Page"; URL:
http://www.ro.feri.uni-mb.si/predst/martin/4_12_2000/301specs.html

LynuxWorks (2005e), "LynuxWorks Patented Technology Speeds Handling of Hardware Events"; URL: http://www.lynuxworks.com/products/whitepapers/patentedio.php3

LynuxWorks (2005f), "Processes, Name Spaces and Virtual Memory"; URL:
http://www.lynuxworks.com/products/posix/processes.php3

LynuxWorks (2005g), "Threads in POSIX"; URL:
http://www.lynuxworks.com/products/posix/threads.php3

OCERA (2002), "WP1 - RTOS State of the Art Analysis"; URL:
http://www.ocera.org/archive/deliverables/ms1-month6/WP1/D1.1.html
(OCERA = Open Components for Embedded Real-Time Applications)

Scott Spetka (2004). "Real-time Linux"; URL:
http://www.cs.sunyit.edu/~scott/realtime/RTLinux.notes

Vik Sohal and Mitch Bunnell (1996), "Sophisticated real-time products need an OS designed for real-time work"; URL: http://www.byte.com/art/9609/sec5/art1.htm