



# Ark RTOS Overview

**James R. Bergsten, Founder**

*[jim@thebergstens.com](mailto:jim@thebergstens.com)*

**April 27, 2013**

# Executive Summary

Ark Systems owns internally developed Intellectual Property (IP) consisting of:

- ✓ A highly efficient, easy to support, portable, multiprocessing Real Time Operating System (RTOS)
- ✓ A full-featured, data storage appliance application

Ark seeks to be acquired by an established enterprise wishing to leverage Ark's IP to bring new product offerings to the marketplace.



# Ark History

- *Ark Research* was formed in 1995 to provide IT disaster recovery and business continuance solutions for the midrange open systems marketplace
- Ark was acquired by LSI Logic in 2000 (\$23M cash) to give LSI's customers remote data mirroring capabilities
- Ark and LSI installed over sixty systems worldwide that ran for years without a single high severity problem
- Due to business strategy changes, LSI quiesced the project in 2002
- *Ark Systems* was formed in 2005 to spin-off, significantly enhance the IP, and reintroduce storage products to the marketplace
- Ark is funded by its founder – it never reached critical mass and was quiesced in mid-2008, though product development continues.



# Ark Value

- Cross industry – used “everywhere” – industry, medical, power/energy, telecommunications, IT, manufacturing...
- Scalable – horizontal (multiple nodes) or vertical
- Portable – remove dependency on single platform
- Build customer base – compete against the small, migrate toward the big – lower end cheaper COGS
- Simple – easy to control software process, develop, predictable, lower NRE and OPEX, faster TTM, simpler upgrade
- Robust – high quality, reliability, and provable correctness lead to easier certification
- Build IP resource, introduce data-storage offerings, add expertise
- No GPL / overseas issues
- Capture customers / applications



# RTOS History

- In-house RTOS started in 1995 – nothing on the market suited our needs
- Built on the principle of “zero-based budgeting” – features added only to optimally address application requirements
- Used for Ark’s Storage Appliance products
- Production-stable since 1997
- Shipped to over sixty customers worldwide
- All RTOS code developed internally (CA) – only “licensed” code is from chip vendors (device drivers and loadable firmware)
- No “severity one” defect, ever!



# Proprietary RTOS

- Highly optimized for real-time processing
- Multiprocessing and multitasking support
- “Portable” – many potential uses
- NOT a Linux/UNIX “knock off” ALL code developed in-house (CA)
- Small footprint (approx. 230K x86)
- Quick, simple build (“compile → link → load → boot” < 30 seconds)
- Especially useful for critical applications that must be “provably correct” and tamper free
- Full SCSI device emulation – 1-12 Gb SAS, 1-16 Gb FC, parallel SCSI, iSCSI
- Full SCSI device control – 1-12 Gb SAS, 1-16 Gb FC, parallel SCSI, SATA, SAS
- IPv4 TCP/IP stack – application TCP/IP, Telnet, FTP, SNMP, HTTP...



# Industries

- ✓ Education -- Drexel University
- ✓ Telecommunications / Cellular -- MobilCom
- ✓ Entertainment / Media / Video -- MTV
- ✓ Back Office / Payroll Ceridian
- ✓ Real Estate -- ECE
- ✓ Manufacturing / Industrial -- Wacker Siltronic, PPG
- ✓ Legal -- First American Title Insurance Company
- ✓ Retail -- Next Retail Ltd.
- ✓ Health Care -- Robert Bosch Krankenhaus
- ✓ Banking / Finance -- Savings Bank Organization, SWIFT
- ✓ Medical -- St. Olav Hospital
- ✓ Transportation -- Siemens Transportation Systems
- ✓ Utilities -- Southern Company / Alabama Power



# RTOS Architecture

- Multiprocessing/Multitasking built in from “day one”
- Portability built in from “day one”
- Monolithic Kernel
- Privileged and non-privileged user states
- Single, flat V=R address space
- Everything is reentrant





# Locks / Semaphores...

- **“Non-deterministic” Locks**

- ✓ MP-Safe
- ✓ Callable by Kernel and Privileged Tasks
- ✓ Conditional/Unconditional

- **“Deterministic” Locks**

- ✓ MP-Safe
- ✓ Callable by Kernel and Tasks
- ✓ Conditional/Unconditional
- ✓ Single write holder, multiple read holders



# RTOS Queue Support

- **Single and Doubly linked Queue primitives**
  - ✓ **MP-Safe Add, Delete, Insert, Scan**
  - ✓ **Callable by Kernel and Tasks**
- **Doubly linked queues are used everywhere in the design – fixed-length “arrays-of” avoided**



# Tasks

- Kernel mode has no “tasks”...
  - ✓ Interrupt-driven (I/O, events, timers...)
  - ✓ Schedules tasks (time-slice end, task-wait...)
- Kernel supports FIFO “task” queue to handle “back-end” of interrupt handlers
- Privileged tasks can disable/enable, issue I/O, access Kernel memory
- Everything runs in the same address space (though some memory is protected from user tasks)
- Tasks get separate 4K stacks, protected by Intel’s stack segment register facility
- Each CPU has separate 8K Kernel stack
- Tasks are just callable “routines”



# Common Data Structure Pools

- Kernel primitives for allocation/initialization/release of common data structures, such as:
  - ✓ Task blocks
  - ✓ I/O blocks
  - ✓ Wait/post related blocks
  - ✓ Timer related blocks
- Pools are automatically replenished/freed
- Less frequently used structures have application independent allocation/initialization/release routines
- All data structures have unique “eyecatchers” used to verify requests
- Fast, application-independent structure use/validation



# Task Scheduling

- Four priority levels – high, medium, low, “none”
- All runnable tasks at a higher priority level run before tasks of lower priority. Tasks can change their priority
- Tasks run for a user-definable time-slice, at which point they move to the bottom of their respective priority queue
- Tasks can be assigned affinity to a particular processor or group, otherwise they run “anywhere”
- Tasks can wait on one or more events, a period of time, completion of child tasks, I/O operation end
- A posted waiting task receives a “return code” from each posting task / system



# Signaling

- The Kernel and/or any task can “post” any other (waiting) task
- Tasks can wait on events (one or more of 32 “bits”). There is an array of 32 “return codes” allowing multiple events to each signal their success/failure
- Tasks can wait for a count to go to zero
- Tasks can wait for all spawned child tasks to complete
- A waiting task can optionally specify a wait “time limit”
- A task can wait for “work”



# Intertask Communication

- In the simplest case, information may be passed by “shared” data structures – in other words, memory is not “owned” by any particular entity (though it can be protected from reads or writes)
- The Kernel or any task can “post” a completion event to a waiting task, including a “return code”
- The Kernel or any task can post “work” for another task. The “worker” task can poll or wait for work to arrive. Work can be posted FIFO or LIFO
- The work request data structure can be a part of a larger data structure – it is the responsibility of the application to allocate and delete these structures



# Memory

- All “text” (instructions, R/O data) are protected/checksummed
- All memory is shared by everybody, though some is only accessible by the Kernel and/or privileged tasks
- Memory above 4GB is reserved for I/O and VM’s. Not in address space. Data cache memory in 4,224 byte blocks.
- One “BSS” area shared by everybody (and its use is highly discouraged!). “Global variables” are also discouraged
- No “heap”...
  - ✓ Free memory maintained in aligned 4K “pages”
  - ✓ Unallocated memory access protected by MMU
  - ✓ Memory requests granted in aligned “next-higher-power-of-two bytes” from 4K page dedicated to blocks of that size
  - ✓ Empty segmented 4K page returned to 4K free page queue
  - ✓ Each “power-of-two” buffer size has separate free queue – allocated/freed FIFO
  - ✓ I/O buffer memory can be protected from CPU’s
  - ✓ Cached/uncached memory supported





# Nonvolatile Storage

- Support for ATA/SATA/USB devices (simplex or mirrored)
- “Enterprise” storage (SAS, FC, iSCSI...) device drivers
- SCSI initiator and target modes supported
- Disk / Tape / Optical support
- Error detection / recovery
- Support for mirrored writes, read-from-anywhere, auto-repair
- FLASH device support
- Uninterruptable Power Supply monitoring to handle cached data flushes



# Multitasking

- Fully symmetric/asymmetric multiprocessing support
- Any task can run on any processor, alternately a task can run on a specific processor / set of processors
- Interrupts can be steered to any processor / group
- Certain interrupts have implicit affinity (local keyboard, serial ports, timers)
- All Kernel services are MP-safe, removing most MP concerns from application developers



# I/O Subsystem

- I/O subsystem schedules I/O operations
- Has concepts of adapter, port, path, controller, and device
- Only issues I/O to devices/paths whose active I/O count is less than maximum I/O count
- Supports reordering of device I/O for device optimization
- Device type and location agnostic
- Supports multipathing including automatic path redrive / recovery
- Supports multiple “read-from” algorithms for established, mirrored devices
- Supports redirection of I/O to remote Ark system



# Device Drivers

- All device drivers share common interface – 21 calls, including:
  - ✓ Initialize, reset, I/O, expected interrupt, unexpected interrupt, missing interrupt, various resets, return/clear counters, etc.
- Drivers passed common data structure (I/O block)
- Drivers can queue interrupt post-processing
- PCI(e) scan performed by RTOS – drivers called based on PCI configuration information
- Serial port, keyboard, display, timer, parallel port, ATA/SATA, SCSI, FC, iSCSI, SAS, USB drivers
- Any driver function can be called anytime
- No restriction on driver's Kernel requests



# Interrupts

- Bus and platform I/O interrupts supported, including “steering” to specific processor core or group
- Legacy, MSI, MSI-x support
- Timer interrupts supported
- Unexpected event interrupts, including:
  - ✓ Application/Task exceptions
  - ✓ Single/double-bit memory errors
  - ✓ PCI (and other bus errors)
  - ✓ Processor errors
- Supports hardware-based interrupt priorities
- Device driver receives interrupt, may queue “above-the-line” processing if desired



# Timers

- Periodic incrementing 64-bit storage-based timer location maintained
- Interval timer supported
- Number-of-cycles timers supported
- Elapsed time supported
- Date/time clock support, including time zones and DST
- Kernel supports single shot and repeated expiration events – used for:
  - ✓ Delays
  - ✓ Task timeslice
  - ✓ Specific processing at given time
  - ✓ Specific processing at given interval
  - ✓ Time-outs (e.g. I/O, event time-out, etc.)
  - ✓ Watchdog timers



# SCSI Simulation

- Built-in SCSI target simulation
- Supports disk, tape, optical device types
- Supports “complete” SCSI protocol set – virtualizes some commands – SAM-5, SPC-4, SBC-3, SPL-3
- Protocol independent implementation (SAS, FC, SCSI, iSCSI... handled at lower layer)
- Supports registering with name servers
- Can be configured to limit access to specific initiators
- Supports LRU caching of data blocks – Kernel “fast path” for “hits”
- Supports local device based nonvolatile copy of “dirty blocks”
- Application supports LBA granular virtualization, transparent local/remote device access, “n”-way mirroring...



# Networking

- “Proprietary” IPv4 TCP/IP user-task-based protocols – DoS resistant
- 10/100/1G/10G Ethernet drivers included
- Support for TOE offload functionality
- Optional application control over ACK generation – optimizes latency for “block-based” TCP protocols
- Supports “fast-path” packet handling, zero buffer copy I/O, and some RDMA
- Support multiple ports – applications may access one, several, or many ports at once
- Supports multiple or single IP addresses
- Supports restricting access to specific IP source addresses
- Supports multihoming – any number of simultaneous paths, path error detection and recovery





# Networking Protocols

- **Generic UDP / TCP application calls**
- **User-mode task basic support for the following protocols:**
  - ✓ Telnet server (CLI support)
  - ✓ FTP server (new code download, log upload...)
  - ✓ SNMP (send events, respond to requests)
  - ✓ HTML server (help-files)
  - ✓ DHCP client
  - ✓ DNS client
- **“Typical” CLI commands (ping, traceroute, nslookup...)**
- **No SSH, encryption, etc. (access IP-source based), no IPv6**



# Initialization, Shutdown, Restart

- Intel implementation has small 16-bit RTOS loader – loads code and configuration, calls RTOS initialization in 32-bit mode
- Single kernel runs in various modes depending on hardware
- Initialization initializes hardware, Kernel data structures, scans busses and calls drivers, initializes other processors, starts initial task (which starts other initial tasks)
- Three kinds of “shutdown”:
  - ✓ Restart – branches to start of loaded RTOS
  - ✓ Warmboot – branches to start of loaded boot loader
  - ✓ Coldboot – does hardware reset
- RTOS code is read/only and checksummed – will not restart if checksum is invalid



# Bootstrapping

- Boot loader supports booting from flash, “floppy,” USB, or local ATA/SATA disk
- Flash/disk supports multiple copies of RTOS (usually four – two redundant copies of “current” and “previous” – loads failing integrity checks cause fallback to other copies
- Flash/disk also contains up to eight “configuration files” – a text file containing system parameters, options, etc.



# Configuration

- The RTOS is passed a “known storage address loaded” text configuration file
- Configuration file contains “CLI-like” command lines
- This file contains system parameters and options, for example:
  - ✓ System parameters
  - ✓ Tuning parameters
  - ✓ Networking parameters
  - ✓ Port and device parameters
- Configuration parameters can be supplemented and/or overridden any time after boot (add/delete a device, add/delete a userid, etc.)
- Code and configuration can be placed on portable USB drive allowing for quick hardware replacement



# Software Exceptions

- Generally, software exceptions cause a “restart”...
  - Restart of existing RTOS if R/O checksum is valid
  - Otherwise, “warm boot” of existing RTOS
- Option to stop at “white screen of death” for real time debugging (using serial port or “keyboard / display”)
- Option to dump storage to local disk device before reboot
- Option to dump all or selected memory
- Task failure can be trapped via task “user exit” avoiding a reboot



# Exception Handling

Exception handling is dependent on the nature of the exception...

- Device I/O errors go through device-dependent error recovery...
  - For example, bad disk block read may cause retry, read/repair from mirror device, etc.
- I/O operation timeout may cause resets, attempt at I/O down alternate path, etc.
- Single-bit memory errors are “scrubbed” and logged
- Double-bit memory errors cause “cold boot” (nothing is safe)
- PCI errors cause retry, or logging and “cold boot”



# Command Line Interface

- About 375 distinct CLI commands – syntax is “verb modifiers” – command names reasonably “obvious”
- Four command privileges – user, privileged, administrator, maintenance.
- Separate “debug” flag for commands valid while “stopped”
- Non-volatile userid/password table defines user privileges
- Source IP address can be used to restrict privileges
- Command synonyms supported, primitive “macro” facility
- Supports auto-scrolling console display with multiple levels of messages (normal, error, attention...)
- Multiple simultaneous Telnet, serial port, keyboard/display users running at different command privilege



# Portability

- RTOS was designed from “day one” to be potentially portable
  - ✓ Little/big Endian conditionals already present
- Architecture-dependent code written in assembler – “C” code should be completely portable
- Ports to new “same processor architecture” boards typically takes less than a day
- Porting new device drivers can sometimes be done in under a week





# Compatibility

- Minimal, in-house-developed POSIX support
- GNU (or other such runtime libraries) not supported
- C++ not supported
- No floating point
- Bare-bones “filesystem”
- Ports of JVM, ZFS, GNU runtime under consideration
- Device drivers have been semi-ported, semi-rewritten on a case-by-case basis
- “Easy” to add compatibility layers for migration



# Debugging

- “Messages” are non-blocking and can be displayed from anybody at any time
- Messages can be logged (by severity level)
- Symbol table built into the RTOS – primitives exist to display “entry-point plus offset” in messages
- MP-safe, timestamped trace table(s) – user entries supported – types can dynamically be enabled/disabled
- Myriad number of CLI-based debugging commands
- CLI works even when system is “white-screened” from keyboard/display and/or serial port(s).
- Primitive “address stop” facility
- Primitive task suspend / resume facility



# Build Process

- Single file compile, link, download, reboot completes in under 30 seconds
- Full source build takes under five minutes
- Currently using MS compilers and make (NOT MS IDE)
- Multiple compilers tested to ensure tool compatibility, find errors
- GNU tool port started but incomplete
- Simple, single-file makefile
- Simple one-level source tree (source, includes, objects, listings)



# Documentation

- End-user guides / HTML pages
- Partial CLI documentation (“maintenance” commands not divulged to customers)
- Several “white papers”
- The source itself (very heavily commented!)
- Some online CLI help



# Lines of Code (RTOS + App)

Source: LocMetrics

| Symbol       | Count     | Definition                        |
|--------------|-----------|-----------------------------------|
| Source Files | 393       | Source Files                      |
| Directories  | 18        | Directories                       |
| LOC          | 609,823   | Lines of Code                     |
| BLOC         | 52,900    | Blank Lines of Code               |
| SLOC-P       | 463,506   | Physical Executable Lines of Code |
| SLOC-L       | 249,645   | Logical Executable Lines of Code  |
| MVG          | 55,903    | McCabe VG Complexity              |
| C&SLOC       | 145,295   | Code and Comment Lines of Code    |
| CLOC         | 93,417    | Comment Only Lines of Code        |
| CWORD        | 1,015,252 | Commentary Words                  |
| HCLOC        | 7,505     | Header Comment Lines of Code      |
| HCWORD       | 38,892    | Header Commentary Words           |

