Core Software Architecture for Memory Protection

Joint IRAD JHU/APL & NASA/GSFC

• Christopher A. Monaco – JHU/APL
  Christopher.Monaco@jhuapl.edu, 240-228-5387

• Alan P. Cudmore – NASA/GSFC, Software Engineering Division
  Alan.P.Cudmore@nasa.gov, 301-286-5809
Objectives / Goals

• Enhance the core Flight Executive (cFE) component architecture to take advantage of the process memory model
  – Utilize the hardware Memory Management Unit (MMU)
  – Protect software components from each other – protect code and data space.
• Maintain kernel mode compatibility
• Minimize impact to cFE and user applications
Project Team

NASA GSFC
- Dwaine Molock
- Alan P. Cudmore
- Jonathan Wilmot
- Kequan Luu

JHU/APL
- Christopher A. Monaco
- Bill Van Besien
- Jonathan Green
Research Process

- **Started With:** Standard cFE
- **Phase 1:** Application Process Model
- **Phase 2:**
  - **Phase 2a:** cFE Process Model with Shared Libs
  - **Phase 2b:** cFE Process Model Without Shared Libs
- **Phase 3:** Metrics And Benchmarks
- **Post IRAD:** More Efficient Shared Lib cFE
Phase 1: The “Application Process Model”

- The original vision for the IRAD was to modify cFE to enable each application on the bus to exist as a user-mode process: The “Application Process Model”
- The cFE, OSAL, and PSP would become a dynamically loadable “shared library” available to all cFE and cFS Applications.

- Drawbacks to Application Process Model:
  - Implementation required shared memory – cFE and OSAL data must be available to all processes.
    - Undermines the objective of the IRAD - Erroneous process can still easily cause system failure through corruption of shared memory.
  - Significant modifications to cFE and OSAL - undermines the demonstrated maturity of cFE and would require significant retest.
  - Concern for additional overhead associated with context switching of processes versus tasks
  - Concern over prevalence of system calls and associated overhead.
Phase 2: The “cFE Process Model”

- Put the entire cFE system into a process
  - Instead of putting all of the apps onto a single cFE system, the apps could be organized into 2, 3, or 4+ cFE processes.
- Similar to a partitioned system – allow multiple cFE systems to run on one physical processor, each within a separate user mode RTP.
  - Addresses memory isolation and processor consolidation
- Applications still communicate with each other through the Software Bus (SB)
- cFE Processes can communicate with each other via a Software Bus Network (SBN) application.
Phase 2: The “cFE Process Model”

- The SBN currently uses UDP/IP to communicate with all peers (local and remote).
  - SBN could easily use OS queues to communicate with local processes and use UDP/IP (or other protocol) between remote CPUs.
- cFE Process Model offers “complete” memory protection of a process - no shared (writable) memory.
- Although memory protections is “complete” the granularity of memory protection is relatively low compared to the Application Process Model.
- Implementing the cFE Process Model requires a scheme to manage cFE resources (Critical Data Store, RAM Disk etc), User-mode port of PSP and OSAL.
Phase 2: Startup Manager (SM)

- New kernel mode component
- Creates system resources formerly created by PSP
  - Per process: ES CDS, ES Reset Area, User Reserved
  - Shared by all processes: Nonvolatile RAM disk
- Launches each cFE process using a configurable startup script – defines process specific parameters
  - i.e. script specifies “Exception Action” – what to do upon SW Exception.
- Monitors each cFE Process
  - Upon error or request, the SM can restart a process or restart the board
- For convenience and portability the SM uses the legacy PSP and legacy OSAL
Phase 2: Linux Implementation

- cFE already runs in user-mode in the Linux Environment – Legacy cFE with Linux OSAL and PC-Linux PSP.
- Legacy Linux implementation only allowed one cFE Process on a machine due to the location and names of some system resources.
- The Linux OSAL and PSP were updated to make these cFE resources process specific – allowing multiple cFE Processes on a Linux host.
Phase 2: vxWorks RTP Implementation

• VxWorks on mcp750 PowerPC
• PSP Port for VxWorks User-mode
  – User-mode PSP is now HW independent – only depends on OS.
• OSAL Port for VxWorks User-mode
• Uses the Startup Manager to launch and monitor each cFE Process
Phase 2: Shared Libraries

- A goal of the cFE Process Model was to have cFE, PSP and OSAL to exist as a shared library.
- To achieve shared library version of cFE – each library must be able to take on the personality of the host process
  - Process name, message IDs, startup script name and location etc.
- Message IDs (MIDs) are currently #defines in header files.
  - Tried changing MIDs to refer to dynamic data rather than constant #define – However, cFE (ES, EVS, SB, TIME) uses switch statements to process messages.
- Added a SB API to “Register” an MID with a function and to Process a message (call the registered function)
  - Does not use switch on MIDs
  - This the only proposed change to cFE and it is not specific to user mode.
  - Helps make all applications uniform, avoid re-implementing message handing in each App.
- Until this new API can be added to cFE and cFE subsystems updated – each cFE process can be compiled as a fully linked executable image.
Phase 3: Metrics and Benchmarks

- Memory Utilization
- Memory Protection Test
- Software Bus Performance Test
- Memory Copy Test
- Dhrystone Benchmark Test
Memory Utilization

• A Comparison of Kernel Mode cFE and a single User Mode cFE instance

<table>
<thead>
<tr>
<th>cFE Implementation</th>
<th>Size of memory in Flash/EEPROM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kernel Mode cFE &amp; Simple Apps</td>
<td>378K bytes</td>
</tr>
<tr>
<td>User Mode cFE &amp; Simple Apps</td>
<td>581K bytes</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>cFE Implementation</th>
<th>Size of memory in RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kernel Mode cFE &amp; Simple Apps</td>
<td>1,335K bytes</td>
</tr>
<tr>
<td>User Mode cFE &amp; Simple Apps</td>
<td>1,662K bytes</td>
</tr>
</tbody>
</table>
Memory Protection Test

- Deployed three processes on mcp750
  - Two nominal
  - One with deliberate memory error
- Demonstrated the ability to detect and handle the memory error in the faulting process.
  - Either restart the board or restart the process
- Repeated the test serially over 2500 times with no noticeable impact on two nominal processes.
  - Suggests no memory leaks.
Software Bus Performance Test

- Software bus throughput test – cFE Kernel mode and cFE User mode

<table>
<thead>
<tr>
<th>cFE Implementation</th>
<th>Parameter/Data Size</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>cFE Kernel Mode</td>
<td>128 byte packets (1Gbyte total)</td>
<td>339 Seconds</td>
</tr>
<tr>
<td>cFE Kernel Mode</td>
<td>1024 byte packets (1Gbyte total)</td>
<td>47 Seconds</td>
</tr>
<tr>
<td>cFE Kernel Mode</td>
<td>16384 byte packets (1Gbyte total)</td>
<td>9 Seconds</td>
</tr>
<tr>
<td>cFE User Mode</td>
<td>128 byte packets (1Gbyte total)</td>
<td>928 Seconds</td>
</tr>
<tr>
<td>cFE User Mode</td>
<td>1024 byte packets (1Gbyte total)</td>
<td>137 Seconds</td>
</tr>
<tr>
<td>cFE User Mode</td>
<td>16384 byte packets (1Gbyte total)</td>
<td>25 Seconds</td>
</tr>
</tbody>
</table>
## Memory Copy Test

<table>
<thead>
<tr>
<th>cFE Implementation</th>
<th>Data Size</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>cFE Kernel Mode</td>
<td>65kbytes x 100,000 copies</td>
<td>36 Seconds</td>
</tr>
<tr>
<td>cFE User Mode</td>
<td>65kbytes x 100,000 copies</td>
<td>39 Seconds</td>
</tr>
</tbody>
</table>

## Dhrystone Benchmark Test

<table>
<thead>
<tr>
<th>cFE Implementation</th>
<th>Dhrystone Iterations</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>cFE Kernel Mode</td>
<td>1,000,000</td>
<td>434k dhrystones/sec</td>
</tr>
<tr>
<td>cFE User Mode</td>
<td>1,000,000</td>
<td>322k dhrystones/sec</td>
</tr>
</tbody>
</table>
Test Conclusion

- cFE Software Bus performance is close to 3 times slower in User Mode
  - Due to overhead of system calls
  - OSAL Queues can be optimized for in-process message passing
- Dhrystone benchmarks are 25% slower in User mode
- Memory Copy benchmarks are nearly the same in User and Kernel Mode
  - Due to little or no system call overhead

- Since most applications are a mixture of cFE/System calls, memory copies, and other computations, real apps will be somewhere in the middle for performance.
What’s Next?

• Submit the following to configuration management
  – User Mode vxWorks cFE Platform Support Package
  – User Mode vxWorks Startup Manager
  – User Mode vxWorks OS Abstraction Layer Port

• Work on integrating the cFE message handling interface into the cFE configuration management

• Re-Run benchmarks on the latest version of vxWorks (6.9)

• Optimize inter-process and intra-process communication (user mode OSAL queue and SBN implementations).

• FY12 – User mode in a multi-core flight environment.