Improving SMP Deficiencies Regarding Real-Time Application Characteristics

David Akre, Jon Anderson, Miguel Martinez-Violante,

Nwachukwu Okwudirichi, Greg Romero

Electrical and Computer Engineering, University of Arizona  
Tucson, Arizona 85721

dakre@email.arizona.edu, jonanderson@email.arizona.edu, mmartin01@email.arizona.edu, oknwachukwu@email.arizona.edu, gregrom@email.arizona.edu

Abstract— Multicore Symmetric Multiprocessing systems (i.e. SMP) running real time applications is an area that requires improvements (in particular with systems running native Linux). Linux out of the box is a time sharing operating system that allows multiple processes to share memory and processing resources. This concept is great for increasing bandwidth for multi-process intensive systems, but adds additional latency overhead that is not desirable for real-time applications. The goal for this research paper is to identify these deficiencies in multicore systems running SMP Linux, and to propose improvements to these systems to become more real-time oriented.

***Key Terms: SMP, AMP, OS, RTOS***

1. **Description**

Symmetric Multiprocessing is a hardware and software operating system concept of sharing resources amongst multiple processes across multiple processors. The SMP OS assigns tasks among the processors to provide a natural load balancing amongst all processors [1]. Prior to SMP, there was Asymmetric Multiprocessing (i.e. AMP) which was used for uniprocessor systems, where only one process was allowed to use the processor at a time. Evidently, allowing this type of separation between processes leads to less interfering tasks [2]. Many real-time applications desire AMP behaviours because context switching and the sharing of resources is much more predictable and controllable in comparison to SMP. Overall, running applications on an SMP multicore processor is advantageous to increase the system’s bandwidth to process data [1], but requires certain improvements to the SMP architecture to decrease process latency and increase multi-process determinism. These improvements could be locking certain processes to certain processors, restricting memory regions to pertain only to specific processes, allowing the ability to have user space applications run at the highest operating system priority, and other such mechanisms made at the kernel level which can improve an SMP system to abide by more real-time characteristics. The research we are proposing to do incorporates improvements made to the SMP architecture running Linux on desktop and embedded platforms. Understanding how these improvements help on different chip sets is also immensely important for ensuring the portability and generic usages for real-time applications running on an SMP architecture.

1. **Motivation For Topic**

In industries such as Aerospace and Telecommunications there is a lot of emphasis on real time systems and applications. These systems require predictable or deterministic responses from processes due to the fact that mistimed responses from processes can lead to system failure or a loss in quality of service. Achieving real-time typically means there’s a trade-off in increase bandwidth unless there is unlimited CPU resources (normally not the case). Ideally, these systems have increased bandwidth while maintaining its real-time determinism. Thus Linux is a perfect operating system to sit on top of desktop or embedded platforms to test real-time capabilities versus high bandwidth programs by running SMP. Since Linux is open source and runs SMP “out of the box”, we have a free platform to run simulations on and test out improvements made over the course of this research. Many companies also desire cheaper options to traditional real time operating systems (i.e. RTOS’s) such as VxWorks which a year license can cost over $100,000. Since Linux has the ability to run on many different platforms, this also adds to the research motivation to test on the Linux SMP operating system.

1. **Related Work**
2. **Proposed Methodology**
3. **Proposed Timelines**

The following section describes the project timeline and how it maps to deadlines. Additionally this section covers the allocations of tasks assigned to each team member.

1. Project Timeline

The section below describes the projects timeline in terms of development stages and the allocation of work to each team member.

1. *Stages of Development*

The figure below is a visual representation of the different development stages for this project.



Fig. 1. Project development stages

1. *Breakdown of Work*

At this stage of the project there is still the initial research being performed on the topic. Additionally, at this time possible simulations and test implementation details are still being investigated. If an item is TBD it means that tasks have not yet been allocated.

The contents and this chart will be updated as the project progresses.

TABLE I  
Early Stage 2 Breakdown of Tasks

|  |  |  |
| --- | --- | --- |
| Name | | Tasks |
| Akre, David | Initial research and path finding | |
| Anderson, Jon Mychal | Initial research and path finding | |
| Martinez-Violante, Miguel Angel | Initial research and path finding | |
| Okwudirichi Nwachukwu | Initial research and path finding | |
| Romero, Gregory | Initial research and path finding | |

1. *Configuration Management*

The scope of this project allows for software development. Since there are multiple members in our team some type of configuration management must be maintained.

All software will be kept in a group repository using GitHub [4]. The process for modifying files is outlined below.

TABLE II  
Early Stage 2 Breakdown of Tasks

|  |  |
| --- | --- |
| Step 1 | Check file out from repository |
| Step 2 | Perform updates on file |
| Step 3 | Perform regression test (if needed) |
| Step 4 | Commit file back into repository |

1. **Anticipated Results**

There are different degrees of “real-time” such as hard real-time, and soft real time. Hard real time means that there are no system failures and follows the “5 9’s” rule of thumb which means the systems can only be down 5.26 minutes per year or in other words is highly available 99.999% of the time. Soft real time does not necessarily have to follow that policy, and allows processes to miss deadlines on an infrequent manner which does not result in system failure. All other systems are considered non-real time. Since Linux is naturally a time-sharing operating system utilizing SMP features, improvements can be made to make the operating system to become more real-time capable, but overall the improvements would not be considered hard real time compatible and most likely lead to more a “soft” real time system. Thus, improvements made to an SMP architecture (in particular systems running Linux) to be more real-time capable would be considered soft real time. The reason why is because of the timeline of this research being too short to ensure enough rigorous testing to enforce hard real time requirements.

**References**

1. Kenneth Vines, and Len Day, “Multicore Considerations for Legacy Flight Software Migration” JPL NASA [Online] Available:<https://trs.jpl.nasa.gov/bitstream/handle/2014/43215/12-5292_A1b.pdf?sequence=1> Accessed: Feb. 7, 2017
2. Nico De Witte, Robbie Vincke, Sille Van Landschoot, Eric Steegmans, and Jeroen Boydens, “Evaluation of a Dual-Core SMP and AMP Architecture based on an Embedded Case Study” July 2013 KU Leuven <https://www.researchgate.net/publication/309829008_Evaluation_of_a_Dual-Core_SMP_and_AMP_Architecture_based_on_an_Embedded_Case_Study>, Accessed: Feb 7,