Performance enhancement of real-time computing for small unmanned helicopter autopilot

Nodir Kodirov
School of Computer Science and Engineering, Konkuk University, Seoul, Republic of Korea

Doo-Hyun Kim
School of Internet and Multimedia, Konkuk University, Seoul, Republic of Korea

Junyeong Kim and Seunghwa Song
School of Computer Science and Engineering, Konkuk University, Seoul, Republic of Korea, and

Changjoo Moon
School of Aerospace Engineering, Konkuk University, Seoul, Republic of Korea

Abstract

Purpose – The purpose of this paper is to make performance improvements and timely critical execution enhancements for operational flight program (OFP). The OFP is core software of autonomous control system of small unmanned helicopter.

Design/methodology/approach – In order to meet the time constraints and enhance control application performance, two major improvements were done at real-time operating system (RTOS) kernel. They are thread scheduling algorithm and lock-free thread message communication mechanism. Both of them have a direct relationship with system efficiency and indirect relationship with helicopter control application execution stability through improved deadline keeping characteristics.

Findings – In this paper, the suitability of earliest deadline first (EDF) scheduling algorithm and non-blocking buffer (NBB) mechanism are illustrated with experimental and practical applications. Results of this work show that EDF contributes around 15 per cent finer-timely execution and NBB enhances kernel's responsiveness around 35 per cent with respect to the number of thread context switch and CPU utilization. These apply for OFP implemented over embedded configurable operating system (eCos) RTOS on x86 architecture-based board.

Practical implications – This paper illustrates an applicability of deadline-based real-time scheduling algorithm and lock-free kernel communication mechanism for performance enhancement and timely critical execution of autonomous unmanned aerial vehicle control system.

Originality/value – This paper illustrates a novel approach to extend RTOS kernel modules based on unmanned aerial vehicle control application execution scenario. Lock-free thread communication mechanism is implemented, and tested for applicability at RTOS. Relationship between UAV physical and computation modules are clearly illustrated via an appropriate unified modelling language (UML) collaboration and state diagrams. As experimental tests are conducted not only for a particular application, but also for various producer/consumer scenarios, these adequately demonstrate the applicability of extended kernel modules for general use.

Keywords Small-scale unmanned helicopter, Operational flight program, Earliest deadline first scheduling, Non-blocking buffer mechanism, Real-time operating system, Helicopters, Computer software

Paper type Research paper

Introduction

Current technology trend is leading towards embedded computing. Most of daily appliances, portable devices, industrial and automotive tools are armed with an embedded system. All of them are targeted to do a particular job. However, what they have in common is to produce a presumed output by given time and quality. Generally, the above two qualities are understood as being an efficient. Computing efficiency is pivotal for all systems and embedded systems are not an exception. Efficiency of the system is a general term and there are number of ways to achieve it. In this work, it is approached from two grounds, where both have a direct relationship with efficiency and indirect relationship with control application execution stability through enhanced timely execution of the system.

The first approach to enhance operational flight program (OFP) performance and timely execution is to add earliest deadline first (EDF) scheduling algorithm to the embedded configurable operating system (eCos) kernel (Massa, 2002). EDF (also known as least time to go) is deadline-based dynamic scheduling algorithm used in RTOSs (Baker and Baruah, 2008; Burns *et al.*, 2009; Liu and Layland, 1973). The second approach

The current issue and full text archive of this journal is available at **www.emeraldinsight.com/1748-8842.htm**

Aircraft Engineering and Aerospace Technology: An International Journal
83/6 (2011) 344–352
 [ISSN 1748-8842]
[DOI 10.1108/00022661111173225]

This work was partially supported by Defense Acquisition Program Administration and Agency for Defense Development under the contract (UD060048AD).

Aircraft Engineering and Aerospace Technology: An International Journal
Volume 83 · Number 6 · 2011 · 344–352

is to provide application-specific thread message communication mechanism in OS kernel. It is based on so-called non-blocking buffer (NBB) mechanism to achieve finer-timely execution. NBB is a lock-free thread message communication mechanism (Kim *et al.*, 2007; Anderson *et al.*, 1995). It provides lock-free thread communication, with the advantage of less scheduler locks at OFP runtime; thus, having more room for the other urgent computation to be executed earlier. The decrease in amount of the scheduler locks in the shared resource management results to more a responsive kernel for timely critical OFP tasks. In this way, indirect positive impact can be made in timely execution of the OFP and real-time operating system (RTOS) kernel in a whole.

Rest of the work organized as follows. First, problem definition will be provided, where reasons to follow these approaches and expectations from eCos-EDF and eCos-NBB will be stated. Next, related works addresses similar issues, such as deadline and thread communication on real-time embedded systems will be presented. In the next section, an insight view to the computational characteristics of UAV OFP will be provided. Unified modelling language (UML) collaboration diagram (Grady *et al.*, 2000) for OFP core modules, hardware structure and servo/sensor connection architecture and UML state diagram will be included. After demonstration of performance improvements introduced by EDF and NBB for OFP, the work is concluded with the future plans.

Problem definition

Rotary blade control

The rotary aircraft or rotorcraft flies with rotor blades that revolve around a mast. It gives a lift to the helicopter. Representative rotorcraft is a helicopter that consists of one main blade and one tail rotor blade for stability. Unlike fixed wing aircraft, rotorcraft is able to hover or change orientation quickly. Owing to this agility, rotorcraft can easily lose its centre of gravity and can be an unstable. Therefore, autonomous controller system for rotorcraft must compute a control signal with an accurate timeliness.

The OFP is core software of autonomous control system for UAV. Timely control of the OFP is very critical to provide flight stability and avoid faults. Essentially, UAV system needs several external devices to monitor its attitude and location. Moreover, control signal should be accurately calculated. The OFP must take into account all data exchanges from/to external devices, such as attitude and heading reference system (AHRS) sensor, global positioning system (GPS) sensor, servo motor, main rotor engine, wireless communication module, etc. Each task handling I/O and control logic must not only guarantee correct functional operation, but also keep the time constraints. To meet the above objectives, two approaches to enhance RTOS kernel were proposed. The first approach is to append a deadline-based thread-scheduling algorithm. The second is to provide lock-free thread message communication mechanism for OFP.

Real-time guarantee

In this work, applying EDF and NBB techniques to the flight control program of the small UAV is targeted. UAV is used in disaster response and recovery phases of disaster management system (Kim *et al.*, 2009). It is expected to reach remote and hazardous places to take a moving and still pictures of the hot-spot location, together with capability of having up to 20-kg payload (which could be first aid box).

OFP consists of six threads for reading and writing data, and to execute control logic. In this scenario, one group of threads play role of the producer the others as a consumer. These producers and consumer threads manipulate global data to communicate with each other. Basically, eCos provides several synchronization primitives (Massa, 2002), such as message boxes, mutex (mutual exclusion semaphore) to provide thread communication mechanism. However, their problem is on lock-based nature for shared resource management. Instead, in this work, NBB mechanism was proposed as thread communication mechanism. NBB is lock-free counterpart of mutex and message box. It provides designer flexible retry strategies to access shared data in case of data unavailability or when it is being put (Kim *et al.*, 2007). NBB mechanism can be efficiently utilized to avoid the computation of wasteful and time-consuming lock-based operations.

Related works

The first seminal work in real-time scheduling domain was published at Liu and Layland (1973), which is an almost a half century ago. It defined and analyzed EDF and rate monotonic or fixed priority (FP) real-time scheduling algorithms. Since that, vast amount of research material was produced to examine/contradict schedulers by a several aspects; some of them appeared with a practical implementations and results. This section provides a brief look to those works.

Lightweight EDF scheduling with deadline inheritance (Jansen *et al.*, 2003) is one of the works appeared in a literature. Their approach is to make EDF scheduler for feather-light micro kernels. Generally, in their design, they require and limit application thread context switch to be minimal, thus allowing an application programmer to think of threads behaviour to be run to completion. In this scenario, one thread enters a critical section via blocking others and during execution no other thread can preempt it. Mutual exclusion of shared resources is granted at a system level and programmer does not need to take care of synchronization: processes are simply not scheduled by the system when there is a potential resource conflict. However, in eCos-EDF, no implicit assumptions are made to limit a number of the context switch and threads with higher "priority" are allowed to preempt executing ones.

Burns *et al.* (2009) discusses combined implementation of EDF and FP scheduler in Ada. It has a good mix, having efficiency from EDF and predictability from FP, as it has been done in case of hybrid EDF proposed in this work. Although their approach and theoretical framework has a good foundation, their implementation is in Ada programming language. However, this is not a standalone scheduling algorithm in RTOS kernel, as it will be shown later.

eCos-NBB implementation is based on Kim *et al.* (2007). It illustrates lock-free mechanism for an event message communication for distributed computing objects and other domains. In order to achieve less scheduling locks at shared resources control (thus, obtaining more responsive kernel) extension of eCos kernel with an additional thread communication mechanism was proposed.

Relevant and interesting part of Tsigas and Zhang (2001) is a clear illustration of non-blocking mechanism to outperform its lock-based counterparts. Anderson *et al.* (1995) illustrate advantages of a non-blocking object sharing approaches in uni-processor systems, holding responsibility for the scheduler to

bound interference of threads. However, their approach is simply retrying, in case of shared data access fault. On the other hand, NBB has an advantage of providing designer with flexible retry strategy when buffer is not available due to its states.

Case study for OFP characteristics

This section discusses a functional and non-functional requirement of the OFP for small-scale rotorcraft, GSR-260 shown in Figure 1. Also, OFP core modules' design is illustrated in respective UML diagrams (Grady *et al.*, 2000). First half of this chapter illustrates rotorcraft module structure and UML collaboration diagram of OFP threads. Next, UML state chart diagram will provide more insight look on OFP execution scenario.

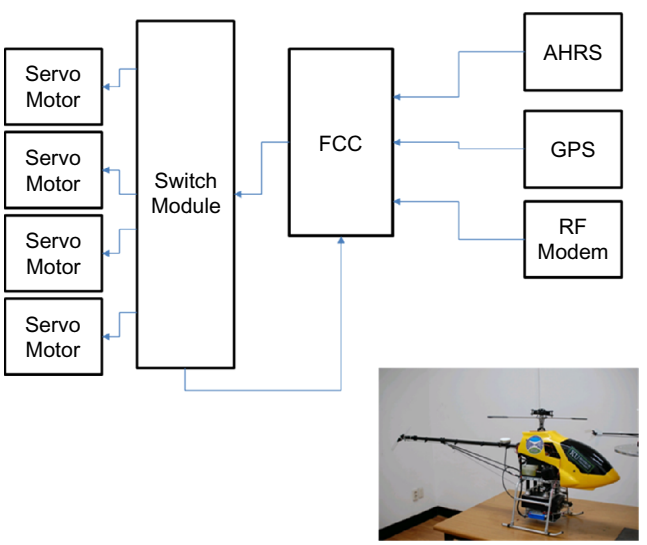
Rotorcraft features and hardware structure

GSR-260 rotorcraft contains AHRS sensor and GPS device for control and guidance operations. Those devices send data through serial communication port. Data frequency of GPS device is about 10 Hz and AHRS sensor is relatively faster as 100 Hz. The UAV has wireless communication device for interaction with ground control system (GCS), which is user-interface module to monitor aircraft status and to manually send control commands. Typical rotorcraft has four control modes (longitudinal, lateral, heading, and throttle), and each control is driven by servomotor. Longitudinal, lateral and throttle servomotors change tilt of the swashplate, and it reflects on attitude of a main rotor. Another servomotor is to control a tail rotor pitch for heading control. Common servomotors are controlled by specific signal named pulse width modulation (PWM). Converter circuit, called as a switch module is used to generate PWM signal. Flight control computer (FCC) with 800-MHz processor is used to integrate external devices and operate OFP software. Figure 1 shows hardware structure of this target system.

Sensor and controller modules

As it was mentioned in previous section of this chapter, UAV system consists of several external devices, such as sensor, actuator and wireless communication module. The OFP should accept every sensor data and calculate control signal.

Figure 1 FCC servo/sensor connection system



UAV system has AHRS and GPS for control and guidance. Wireless communication device is used to interact with GCS. Also, OFP accepts a control signal feedback to evaluate and analyze behaviour of the controller.

Figure 2 shows UML collaboration diagram of OFP core modules. There are four reader modules to accept input data from sensors (`Current_AHRS_Reader`, `Current_GPS_Reader`, `Current_Ctrl_Reader`, `Current_Asyn_Reader`). Each reader module accepts data from a serial communication port and stores it into the `Glob_AHRS_Sens`, `Glob_Ctrl_Pack`, and `PacketAdt` shared storage modules. `Do_Guid_Ctrl` reads a shared data and computes proper control signal. `Do_Guid_Ctrl` module contains a core logic code for control and guidance of UAV, such as autopilot, forwarding, way pointing, etc. Four reader modules and one controller module compete for an access to the shared storage. In other words, the shared storage module access should be done as critical section.

OFP performance enhancements

Two categories of experiments were carried out to check applicability of the implementation in eCos kernel and validate enhancement introduced in OFP performance. Both of the approaches: hybrid EDF scheduling (eCos-HybEDF) and NBB-based (eCos-NBB) eCos kernel, were first tested at OFP-motivated prototype application. The prototype application is a producer-consumer application, which shares computational characteristics with the OFP. Next, implementation was done for OFP to check its behaviour. Two sub-sections below will illustrates experimental results for eCos-HybEDF and eCos-NBB in both categories, starting from OFP-motivated prototype one.

eCos hybrid EDF performance

This section illustrates experimental results for the sample producer-consumer application, and OFP over EDF and MLQ schedulers. As it is shown in Figure 3, two implementations differ in their execution logic. Control logic is implemented as an infinite loop, where two different mechanisms are used to provide periodicity in EDF and MLQ schedulers. Once thread running control logic starts execution, it enters `uav_control_logic ()` function (line 1). Going through basic initialization (lines 2-5) both logics enter control loop (line 6). As its name stands, `control_logic_computation ()` is computation done for the UAV control logic (line 8). It is common for both schedulers and should be executed periodically. On EDF, this periodicity is achieved through `invoke (next_period)` function (line 6 at (a)), which asks scheduler to start re-executing logic from the point where it left. In order to specify next invoke time, argument `next_period` is passed to the EDF scheduler. On the other hand, MLQ scheduler utilizes thread `delay ()` mechanism (line 10 at (b)). Generally, thread delay mechanism registers timer for this thread to be triggered after specified amount of time. In eCos, timer mechanism is also provided via threads. In this scenario, once `delay ()` is called, control is transferred to the thread which is taking care of timer. Thus, we will have one extra context switch and same happens when context is returned back to the while loop thread. As EDF does this job without context switch, but via asking scheduler to return context to the left point when `next_period` is reached. Thus, extra context switches among control thread and timer thread can be avoided. It will be shown how this enables EDF scheduler to have better deadline keeping characteristics (Figure 8).

Figure 2 UML collaboration diagram of OFP core modules

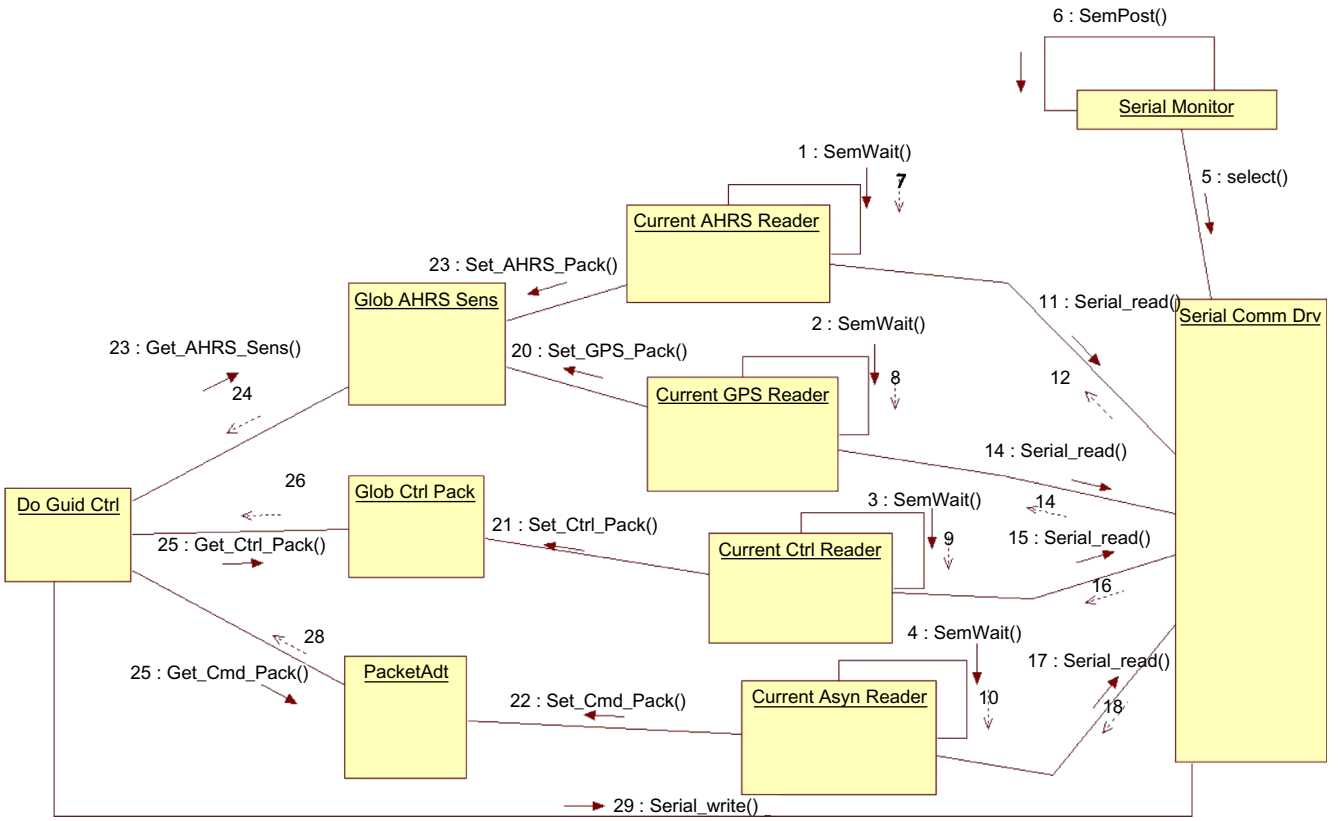


Figure 3 Helicopter control application logic implementation

(a) Application control logic on EDF

```
1.  function uav_control_logic()
2.  {
3.     variable_set();
4.     device_initialization();
5.     [...]
6.     while(invoke(next_period))
7.     {
8.        control_logic_computation();
9.        [...]
10.    }
11. }
```

(b) Application control logic on MLQ

```
1.  function uav_control_logic()
2.  {
3.     variable_set();
4.     device_initialization();
5.     [...]
6.     while(1)
7.     {
8.        control_logic_computation();
9.        [...]
10.       delay(next_period-current_time);
11.    }
12. }
```

EDF implementation is based on data structure with what eCos kernel was extended. It consists of three properties representing start time, worst-case execution time (WCET) and period. These values are provided for each thread by an application programmer. As name of variables state, start time represents a time when this thread is supposed to get started for the first time. For example, in OFP scenario, control logic threads should start execution after initial output by sensor reader threads. WCET specifies the shortest time amount required for the thread to complete execution. EDF scheduler uses it to determine deadline of the thread for it to be able to run once in a within given period (without deadline miss).

This sub-section provides an experimental and OFP practical results for eCos with hybrid EDF schedulers (eCos-HybEDF), starting from OFP motivated prototype application. As it was stated in the problem definition, eCos-HybEDF performance is being checked based on a context switch between concurrent threads. When CPU ownership is changed between concurrently running threads, it is as called a context switch. Context switch among running threads is one of the critical units in ROTS kernel. As only one thread is executed (owns CPU) within one time unit, RTOS scheduler periodically changes executing thread to provide seamless execution of all threads. Context switch between threads always introduces additional CPU and memory operations, bringing extra overhead during application execution. Thus, it is desirable to lessen amount of context switches. It avoids extra CPU utilization spent for change of running thread in given time. As EDF scheduler executes thread based on their deadline, currently running thread is always thread which has shortest deadline among all other threads, unless new thread with shorter deadline joins

the ready thread list (which does not occur frequently). This is the main advantage of EDF scheduler, less context switch enables system to have better performance (removes extra overhead used to introduced by CPU ownership change). Figure 4 shows experimental results for single-producer and single-consumer (SpSc) application.

In this experiment, number of thread context switches were checked during produce/consume of N items. The produced item is simply written to the global variable, which is concurrently accessed by both producer and consumer threads. For all of the experiments (if not explicitly stated otherwise), *X*-axis represents the number of items produced by each producer thread. The axis *Y* represents a number of thread context switches introduced by an application. In above figure, overlapped lines mean SpSc application had the same performance in a both MLQ and EDF schedulers. The reason for the same performance is the number of threads in application. Only two threads do not make a big difference for the MLQ and EDF scheduler, thus having almost the same output.

As number of threads increases (thus having higher CPU utilization) difference between two schedulers can be seen more clearly (Figure 5). In this test, as it is in the OFP, there are five producers and one consumer thread. The producer threads concurrently write data to a shared variable and consumer reads it in a given frequency (20 Hz). In this figure, when item amount per producer is 500 (means each producer produced 500 items, resulting to 2,500 for all five threads) there were around 3,000 thread context switches in MLQ. However, in EDF, it is about 2,500, resulting to more than 15 per cent improvement. It shows eCos-EDF kernel is avoiding 15 per cent overhead introduced by thread context switch. This stands for the same amount of efficiency to be introduced by EDF implementation.

Although it does not match with practical application scenario, just for the experimental interest a test was conducted to check kernel behaviour for even more producer and consumer threads. This resulted to far dramatic change into the behaviour of two implementations. Figure 6 shows the case of five producers and five consumer threads; there is almost 30 per cent improvement with kernel performance in terms of thread context switch. When produced item amount per producer is equal to 100 items (totally 500). This trend was present even in further increase of items' amount.

Based on the above figures, UAV OFP was tested on hybrid EDF kernel it had slight improvements as it was expected (Figure 7). This time thread context switch amount was measured by OFP run time represented in a horizontal axis. *X*-axis represents an amount of thread context switches for each 5 min, ending execution in exact 25th minute. The vertical axis represents amount of thread context switches during specified

Figure 4 MLQ and EDF in SpSc

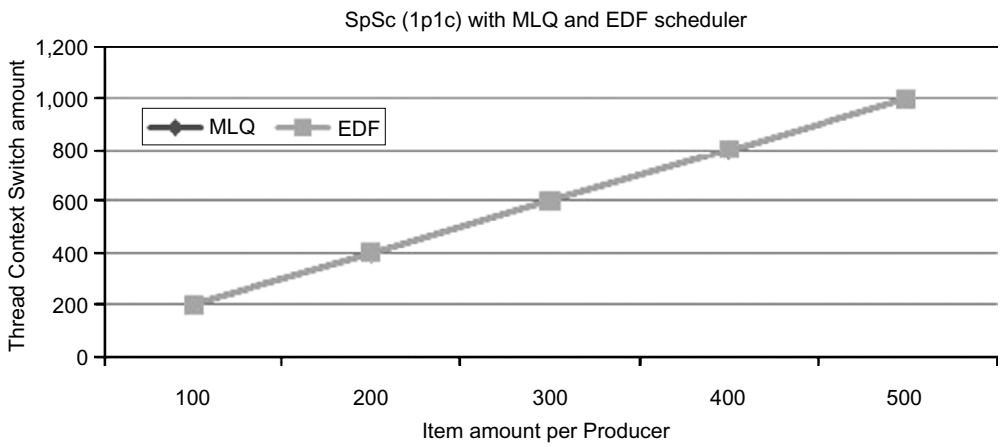


Figure 5 MLQ and EDF in MpSc

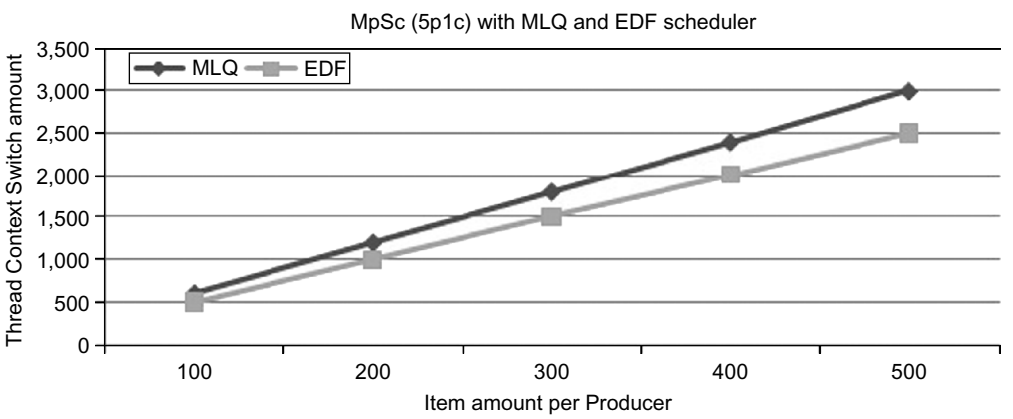


Figure 6 MLQ and EDF in MpMc

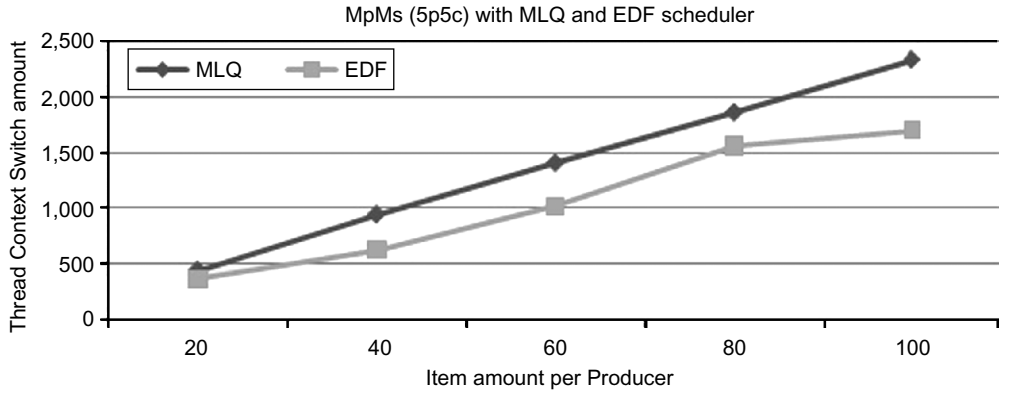
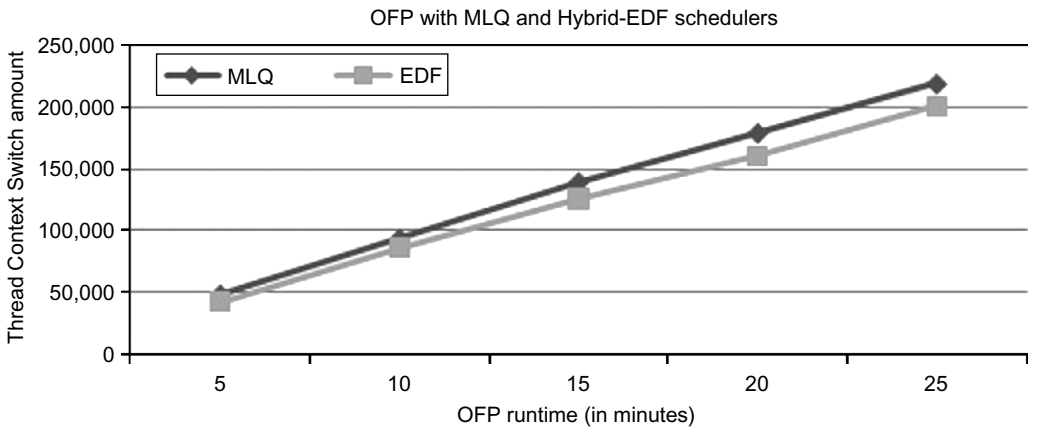


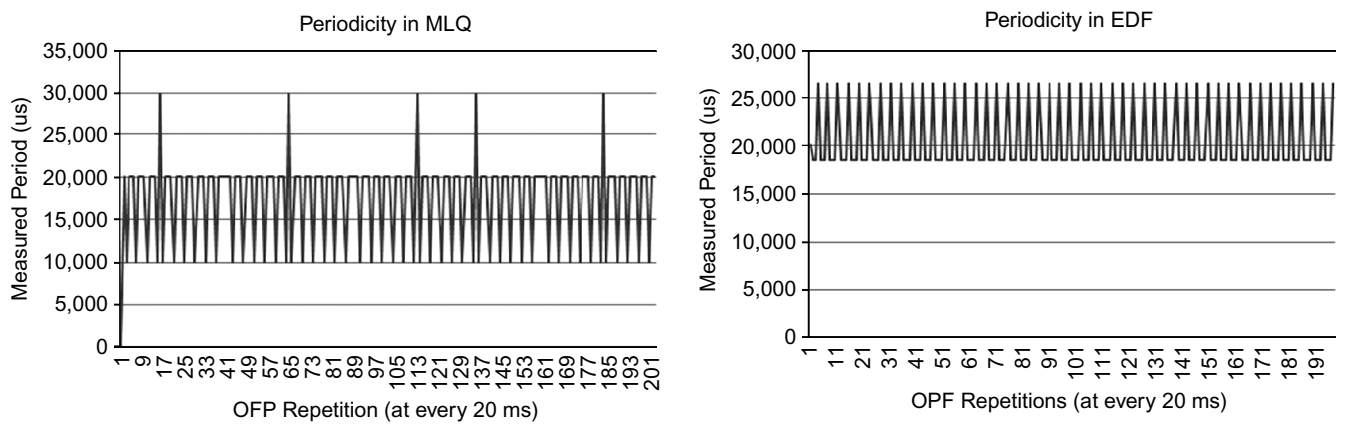
Figure 7 OFP with MLQ and hybrid EDF



time units. As it can be seen from Figure, performance improvements are around 2 per cent, not as dramatic as it was expected. However, as it is the result from our initial implementation, we are working/expecting performance to be better with more accurate implementation of our eCos hybrid EDF. Further improvement shall be demonstrated in future works.

EDF and MLQ were tested for their timely execution characteristics and deadline-keeping features (Figure 8). In this figure, vertical axis represents time frame spent between consequent executions of the control logic. As control logic has 50 Hz frequency, it should be executed every 20,000 μs) which is equal to 20 ms. Horizontal axis represents amount of time OFP was run for the experimental purpose. In this case, it is 30 s. Owing to utilization of the timer mechanism and lack of deadline keeping features, sometimes MLQ scheduler (Figure 3) fails to provide accurate execution time between consequent executions of the control logic. As it is shown in Figure 8 (left), control logic

Figure 8 MLQ and EDF over OFP for deadline measurement



is re-triggered after time frame varying from 10 to 30 ms. Basically, time frame of 30 ms between consequent control logic execution stands for the deadline miss.

As it was stated earlier, EDF contributes for the improvement of the deadline-keeping characteristics.

eCos NBB performance

Similar to the case of EDF, before testing our NBB mechanism directly at OFP, it was tested in OFP-motivated prototype application. As the OFP computational characteristics are better suited to eCos hybrid EDF, NBB implementation was done on with hybrid EDF scheduler. In order to make a general conclusion about eCos kernel performance, NBB performance was compared with existing eCos thread communication mechanism – Message Box (MBox). MBox is one of the message communication mechanisms provided by eCos kernel. As it has a common usage scenario with NBB, we opt for MBox[1]. In eCos, shared data are accessed in a critical section. It is achieved via blocking all other threads from running. In another words, eCos scheduler is locked not to allow assignment of the CPU by any other thread, but running one (via which shared data are being accessed in this particular time). Thus, scheduler lock amount is used as a measurement unit for NBB performance advantage. The reason it was said that this measurement unit to have indirect relationship with timely execution of the system is again (as it was for an EDF) based on importance of the RTOS scheduler for application execution. As it was stated above, scheduler plays primary role to check upcoming tasks with more urgent computation request. In order to be able to detect those requests as early as possible, RTOS scheduler needs to be run more frequently. However, in shared data access, eCos scheduler simply blocks it, which may lead to the late detection/execution of the timely critical task. It increases possibility of the deadline miss. That is why scheduler lock amount was chosen as another judgement criteria to measure timely execution of the tasks. Figure 9 shows SpSc threads, which share the same MBox and NBB for data exchange.

As it is indicated in the *X*-axis, totally 100 items were produced and consumed by consumer threads. *Y*-axis represents amount of scheduler locks introduced by SpSc application during runtime. As scheduler is locked in the critical section (read or write) of the MBox, more scheduler locks are expected to be introduced than it is in NBB (it does not use locks). Figure 9 shows scheduler lock amount in NBB to be slightly less than in MBox. The reason for non-significant difference on amount of the scheduler lock is due to less interference between a single consumer and producer threads. Increase at number of producer threads will make more interference for a shared data ownership, and result to have far different output. In order to emulate OFP characteristics (where control logic unit is concurrently fed with UAV status information by several sensor data reader threads) multiple producer and single consumer application was selected as a test case. It consists of five producers and single consumer thread communicating via single instance of NBB (Figure 10).

Difference in the number of scheduler locks will increase proportionally with a number of items exchanged via NBB. Figure 10 shows the amount of locks to be an almost same, when number of produced items by each thread is equal to 5, resulting to 25 produced items by all five threads. However, once a number of produced items are increased to 25 (resulting to 125 items by all five) 35 per cent less scheduler lock amount can be noticed on eCos-NBB. Based on enhancements illustrated in the experimental scenarios, NBB behaviour was checked at OFP. Originally, OFP used mutex to control access of global data storage. OFP code was re-written to use NBB to feed control logic thread with up-to-date UAV status information. Figure 11 shows the difference in the amount scheduler locks introduced by mutex and NBB, during first 25 min execution of the OFP.

As it can be seen in the figure below, generally NBB has more than 25 per cent improvements in terms of scheduler lock amount. It means NBB-based OFP is 25 per cent responsive than mutex-based counterparts, which will allow NBB-based OFP to have improved kernel behaviour to execute time critical operations.

Conclusion

This work addressed two issues to enhance RTOS kernel. They are extensions of eCos RTOS kernel with deadline-based EDF scheduler and with lock-free message communication mechanism called NBB. Experimental results illustrate usability of above mechanisms to improve timely execution of timely critical applications. Especially, when particular thread message communication mechanism specifically fits with application logic (in our case, it is similarity between NBB's lock-free feature, and producer-consumer relationship among sensor I/O and control logic) it brings much improved behaviour of the application and RTOS kernel in a whole. Tests carried out for general

Figure 9 NBB and MBox in SpSc

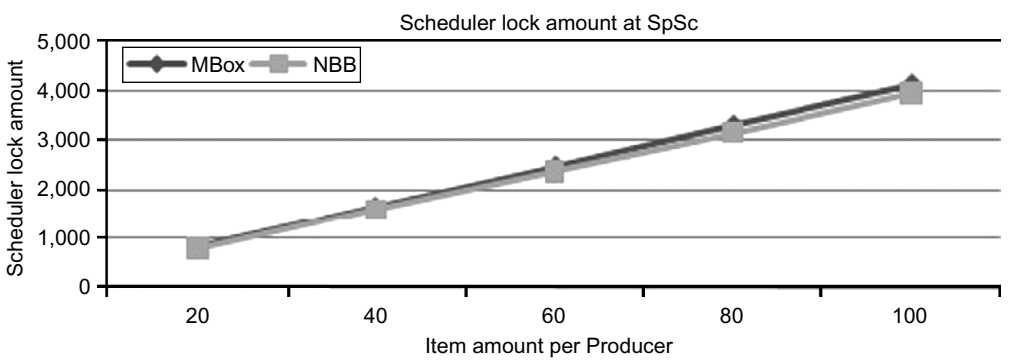


Figure 10 NBB and MBox at multiple producers single consumer (MpSc)

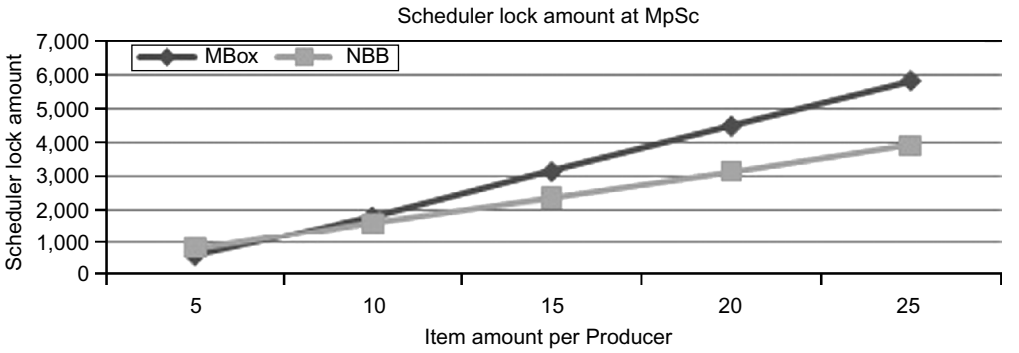
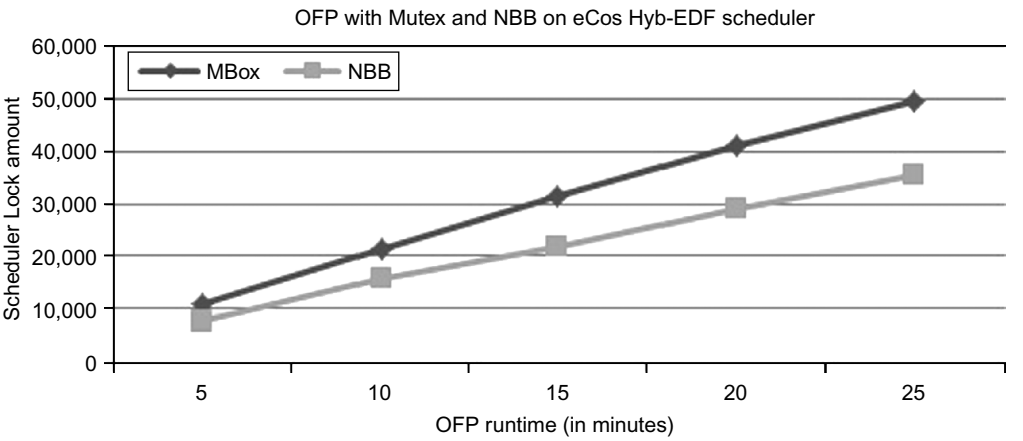


Figure 11 Mutex and NBB in OFP



producer-consumer application approve acceptability of above two mechanisms for general use too. Results of experiments performed with eCos on x86 board showed that EDF contributes around 15 per cent finer-timely execution and NBB enhances kernel's responsiveness around 35 per cent with respect to the number of thread context switch and scheduler lock amount.

Further work

Future works will include more detailed experiments on the correlated effects of the EDF and lock-free buffer for diverse flight modes such as vertical takeoff and landing, auto-hovering, point navigation and so on.

Note

1 *eCos Reference Manual*, available at: http://ecos.sourceware.org /docs-latest/ref/ecos-ref.html

References

Anderson, J.H., Srikanth, R. and Kevin, J. (1995), "Real-time computing with lock-free shared objects", *Proceeding of the 16th IEEE Real-time Systems Symposium, Pisa, Italy*, pp. 28-37.

Baker, T.P. and Baruah, S.K. (2008), "Schedulability analysis of global EDF", *Real-Time Systems*, Vol. 38 No. 3, pp. 223-35.

Burns, A., Wellings, A.J. and Fengxiang, Z. (2009), "Combining EDF and FP scheduling: analysis and implementation in Ada 2005", Lecture Notes in Computer Science, Vol. 5570, Springer, New York, NY, pp. 119-33.

Grady, B., Ivar, J. and Jim, R. (2000), "OMG unified modeling language specification", 1st ed., Version 1.3, March (accessed 12 August 2008).

Jansen, P.G., Mullender, S.J., Havinga, P.J.M. and Scholten, H. (2003), "Lightweight EDF scheduling with deadline inheritance", Internal Report, University of Twente, Enschada, available at: www.ub.utwente.nl/webdocs/ctit/1/000000c6.pdf (accessed 18 August 2010).

Kim, D.-H., Nodir, K., Chun-Hyon, C. and Jung-Guk, K. (2009), "HELISCOPE project: research goal and survey on related technologies", *12th IEEE International Symposium on Object/Component/Service-oriented Real-time Distributed Computing* (*ISORC 2009*), *Tokyo, Japan, March 17-20*, pp. 112-18.

Kim, K.H.(Kane), Juan, A.C. and Kee-Wook, R. (2007), "Efficient adaptations of the non-blocking buffer for event message communication", *10th IEEE CS International Symposium on Object & Component Oriented Real-time Distributed Computing* (*ISORC2007*), *Santorini, Greece, May 7-9*, pp. 29-40.

Liu, C.L. and Layland, J.W. (1973), "Scheduling algorithms for multiprogramming in a hard real-time environment", *Journal of the ACM*, Vol. 20 No. 1, pp. 46-61.

Massa, A.J. (2002), *Embedded Software Development with eCos*, Prentice-Hall, Englewood Cliffs, NJ.

Tsigas, P. and Zhang, Y. (2001), "A simple, fast and scalable nonblocking concurrent FIFO queue for shared memory multiprocessor systems", *Proceeding of the 13th Annual ACM Symposium on Parallel Algorithms and Architectures* (*SPAA'01*), *New York, NY, USA*, pp. 134-43.

Further reading

Baruah, S.K. and Burns, A. (2006), "Sustainable scheduling analysis", *Proceedings of the 27th IEEE International Real-time Systems Symposium*, IEEE Computer Society, Washington, DC, pp. 159-68.

Bertogna, M., Cirinei, M. and Lipari, G. (2005), "Improved schedulability analysis of EDF on multiprocessor platforms", *Proceedings of the 17th Euromicro Conference on Real-time Systems, Palma de Mallorca, Spain*, pp. 209-18.

Buttazzo, G.C. (2005), "Rate monotonic vs EDF: judgement", *Real-Time Systems*, Vol. 29 No. 1, pp. 5-26.

Chao, Y.-H., Lin, S.-S. and Lin, K.-J. (2008), "Schedulability issues for EDZL scheduling on real-time multiprocessor systems", *Information Processing Letters*, Vol. 107 No. 5, pp. 158-64.

Cheng, A.M.K. (2002), "Real-time scheduling and schedulability analysis", *Real-time Systems: Scheduling, Analysis, and Verification*, Wiley, Upper Saddle River, NJ, pp. 45-55.

Corresponding author

Doo-Hyun Kim can be contacted at: doohyun@konkuk.ac.kr