PROCEEDINGS

International Conference on

IT Promotion in Asia 2009

in conjunction with

International Summit on Information and Communication Technologies

September 21-25, 2009

Tashkent University of IT Tashkent, Uzbekistan

Hosted By

ETRI (Electronics and Telecommunications Research Institute)

TUIT(Tashkent University of Information Technologies)

ITIRC(IT Internationalization Research Center)

ETRI Electronics and Telecommunications Research Institute

TUIT

ITIRC Information Technology Internationalization Research Center

International Conference on

IT Promotion in Asia 2009

September 21-25, 2009
Tashkent University of IT Tashkent, Uzbekistan

Hosted By
ETRI (Electronics and Telecommunications Research Institute)
TUIT(Tashkent University of Information Technologies)
ITIRC(IT Internationalization Research Center)

Sponsered By
KETI (Korea Electronics Technology Institute)
PRIMUS' COS Co., Ltd.
JABC (JoongAng Business Consulting) Ltd

ETRI Electronics and Telecommunications Research Institute

TUIT

ITIRC Information Technology Internationalization Research Center

A Design Strategy for Enhancing eCos with EDF for Disaster Response System

Nodir Kodirov, Doo-Hyun Kim*

School of Internet & Multimedia Engineering
Konkuk University, Seoul, Korea

e-mail:{ knodir, doohyun}@konkuk.ac.kr

Abstract. In this paper, we mainly address issues of real-time operating system kernel's configuration, while focusing on design strategy for enhancing EDF scheduling algorithm and testing kernel behavior afterwards. Additionally we will introduce the target scenario of our HELISCOPE project with the emphasis on the four modes for flying-camera task, where configured RTOS will be applied.

Keywords: Embedded Configurable operating system, real-time application, real-time kernel test, unmanned helicopter.

1 Introduction and research motivation

Recently, according to the progress of global warming, the magnitude and frequency of natural disasters tend to be increased. Similarly, industrial disasters are also getting huger due to the progress of industrialization. The disaster prevention has already become a nation-wide issue and prepared by the government through the disaster management which consists of four phases: mitigation, preparedness, and response and recovery phases. The HELISCOPE project is to develop on-flight computing system, embedded S/W and related services for unmanned helicopter that shall be used especially in the phase of disaster response and recovery intensively.

In this chapter, we introduce the target scenario of our HELISCOPE project with the emphasis on the four modes for flying-camera task – which is supposed to be one of the most important functions of our flying system.

Rest of paper organized as follows: firstly we will give brief introduction about main tasks of *on flight system* and functionality of our real-time application. Next we will focus on selection of RTOS (Real-time Operating System), which meets our certain basic requirements, whilst further highlighting valuable features of it. Next we will describe our approach to configure RTOS such that, it fulfills our on flight system's control and real-time application's requirements; additionally we will provide information how to check appropriateness of changes, which was made to RTOS kernel. Finally, we will conclude with works has been done and future works has to be done.

2 Target scenario and real-time application functions

On-flight embedded S/W intrinsically requires real-time features for the computations related to getting current status of attitude, heading and global position, computing the commands for the next movement, communicating with GCS (Ground Control System), and feeding the commands out to the servo actuators. These series of computations should be guaranteed to be finished with every specific period suitable for sustaining safe and stable flight.

Although the unmanned helicopter, HELISCOPE, can be uses in diverse aspect of the disaster management phases, we have special emphasis on the usage of HELISCOPE in the phases of disaster response because it requires immediate and interactive use on-the-spot. The HELISCOPE is to be included in the mobilization of the necessary emergency services and service equipments or can be used as one of first responders. Immediately upon its arriving at the spot of disaster, the HELISCOPE takes off and shoot the still pictures and

* Corresponding author

moving images to send GCS through mobile internet such Wibro (Wireless Broadband) [4] and HSPA (High Speed Packet Access) [6]. In addition to these first stage tasks, the HELISCOPE performs various tasks related to taking detailed video shooting by using the capabilities of unmanned helicopter such as hovering, point navigation, and waypoint navigation.

2.1 Four Modes of Flying-Camera Tasks

The on-flight equipment including camera, i.e., *camera block*, is responsible to accomplish such flying-camera tasks as taking overall/exact view of remote or hazardous location or shots of target object. In order to accomplish those tasks, the camera block can interact with FCC (Flight Control Computer) to compute certain commands directed to servo actuator of camera, such as gimbals. This control can be divided to four modes:

- Mode 1: control by Joystick. As shown in Figure 1, camera can be controlled manually via joystick, where in this mode camera is able to track target object, which is located in camera's area of view.

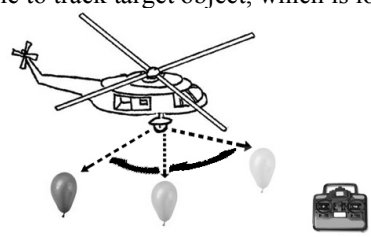

Fig 1.Joystick-Search mode

- Mode 2: Object can be steady also, while human wants to take a shot of target object from different sides. Here, as we said previously and as shown in Figure 2, camera itself cannot see all sides of steady target, while interacting with FCC camera will be able see target object from all sides.

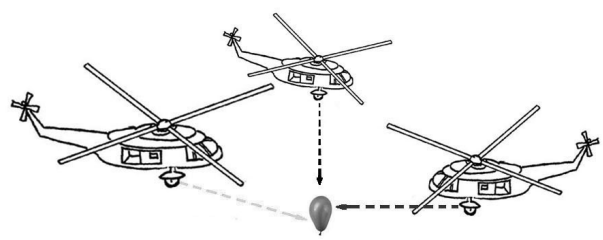

Fig 2.Target-Around mode

- Mode 3: Object moves from one location to another. Here, in order to camera to be able to fully track target object, camera servos interacts with FCC. As shown in the left part of Figure 3, camera is not able track target if it moves to out of view location, which is marked as "out of view", hence in order to track object in that area, camera servos interacts with FCC and FCC giving commands to helicopter servos, helicopter itself moves to appropriate position (marked as "visible").

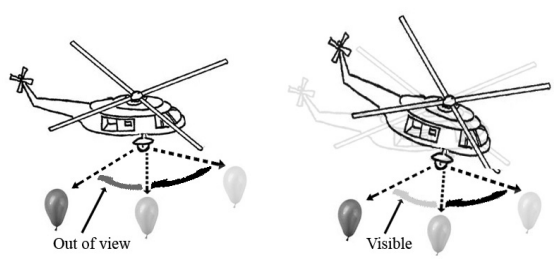

Fig 3.Target-Watch mode

- Mode 4: Helicopter and object moving. Both of helicopter and target object can move, however camera has to always track target object, keeping it in the visible area. As shown in Figure 4 (left to right) in the first phase object located in visible area of camera and in the second phase target object has moved from the first position to the second, which is out of camera's visible area now and horizontally far

from camera itself. In this case camera servos interacts with FCC, while computing appropriate commands to helicopter servo actuators, it changes its longitude and latitude in order to point to target object properly.

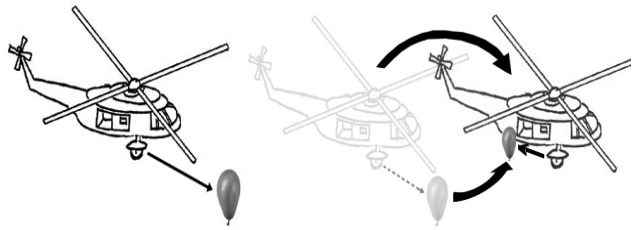

Fig 4.Target-tracking mode

Taking into account abovementioned functions of real-time application, which is to support camera related multimedia tasks - while keeping helicopter itself flying steady and providing interaction between camera servos and helicopter flight control motors requires strictly meet of real-time task deadlines. Otherwise, keeping helicopter itself on the flight maybe endangered. In order to safely run real-time application, on flight system has to have suitable RTOS, which can provide appropriate functionality for it.

3 RTOS selection criteria – eCos

As our project is carried in the purpose of research and implementation, we need for an RTOS which is provided as an open source and without any restricted license; hence we can develop it for our needs with appropriate properties. During survey of active open source RTOSs we found eCos (embedded Configurable operation system) as the most appropriate one, for its highly configurability and other properties [7]. Table 1 illustrates part of widely used RTOSs in more detail.

Table 1. RTOS features

Name	License	Source model	Target usage	Platforms
eCos	modified GNU GPL	open source	general purpose	ARM/XScale, CalmRISC, 68000/Coldfire, fr30, FR-V, H8, IA32, MIPS, MN10300, OpenRISC, PowerPC, SPARC, SuperH, V8xx
FreeRTOS	modified GNU GPL	open source	embedded	ARM, AVR, AVR32, HCS12, IA32, MicroBlaze, MSP430, PIC, Renesas H8/S, 8052
Neutrino	proprietary	Source code provided	microkernel	ARM, MIPS, PPC, SH, x86, XScale
QNX	mixed	Closed and Open source	general purpose	IA32, MIPS, PowerPC, SH-4, ARM, StrongARM, XScale
µC/OS-II	proprietary	Available under license	embedded	ARM7/9/11/Cortex M1/3, AVR, HC11/12/S12, NIOS, 8051, x86, Win32, H8S, M16C, M32C, MIPS, PowerPC, STM32 …

eCos is an open-source, configurable, portable real-time operating system, developed by RedHat™, now owned by eCosCentric™, while primarily maintained by this company and wide open community developers [3].

The main goal behind eCos is to provide embedded developers with a common software infrastructure for delivering a diverse range of embedded products, also plans to let developers focus on the applications, rather than the real-time OS.

One of essential property of eCos is it's highly configurability in source code level, where one can easily add/remove packages depending on his application requirements. Current release of eCos provides more then 200 configurable points, additionally due to its flexible configurability it has vary small footprint, for example - eCos 1.3.1 at its smallest size occupies 3K of ROM and 1K of RAM, for a kernel that includes a scheduler, memory management, real-time clock support, several threads and interrupt handlers.

4 eCos - kernel configuration approach

As it was spoken in previous chapters - eCos provides highly configurability in the source code level. Together with command line configuration tool, eCos provides graphical tool (GUI) to easily add/remove specific package and to make more detailed configuration of them.

It is widely known that real-time systems' behavior mostly depended on its scheduling algorithm. Especially in our research project, where embedded application should take care of flight control and multimedia tasks together, requires operating system to provide sufficient functionality to support number of tasks with strict dead-lines. During last 30 years number of researches has been done on so called "class of real-time scheduling algorithms" field, to fulfill real-time tasks requirements [1, 5].

Currently, eCos version 3.0 fully supports bitmap and multilevel scheduling, while for further releases it is going to include lottery scheduling as well [2]. During our research we are focusing on implementation of one more real-time scheduling algorithm - EDF (Earliest Deadline first), with which system's CPU can be highly utilized [1].

4.1 EDF algorithm design for eCos kernel

In order to implement EDF scheduling algorithm in eCos kernel we may follow several approaches, such as implementing EDF algorithms on the top of existing schedule algorithms (bitmap, multilevel) or designing whole system again. Approach which we will follow is, to construct support for EDF algorithms in data structure level, while doing more specific changes during methodology realization phase. In order to have suitable data structure, we need to add EDF thread specific properties as data fields of our eCos threads, which we want to schedule on deadline basis.

In general, threads running on the bitmap and multilevel scheduling algorithms are created with such properties:

```
void cyg_thread_create(
          cyg_addrword_t sched_info,
          cyg_thread_entry_t* entry,
          cyg_addrword_t entry_data,
          char* name,
          void* stack_base,
          cyg_ucount32 stack_size,
          cyg_handle_t* handle,
          cyg_thread* thread)
```

where `sched_info` is mainly priority of thread, `entry` defines name of function which serves as an entry point for a created thread, `entry_data` is additional data to be passed in function, `name` - given name of thread in human readable form (mainly for debugging purposes), `stack_base` and `stack_size` are pointer to the base of the stack and the total size of this stack, which are used to store local variables of current thread and to keep track of its function calls and returns, `handle` is handler to the created thread with the help of which resume, delay, release operations can be done. Lastly `thread` is used to hold information about thread itself, such as the current state of that thread.

As we want to implement EDF scheduling algorithms, where deadline of thread plays key role and execution order depends on, EDF thread has to have information about starting time, worst case execution time and absolute deadline [1]. Here we add one more property to our threads, which we will call as `cyg_thread_edf_info` structure containing three fields, such as:

```
typedef struct
{
    cyg_tick_count_t    start_time,
     cyg_tick_count_t    wcet_run_time,
     cyg_tick_count_t    deadline
} cyg_thread_edf_info;
```

Here `start_time` is starting time of thread to run, `wcet_run_time` – shortest run time of current thread and `deadline` – defines time before thread has to be run.

However, we need tool for *schedulablity analysis*, where EDF scheduler needs to determine that the requested `wcet_run_time` is available between the `start_time` and the `deadline` and reject the call if it is not. Most EDF systems are split into two parts - off-line analysis tool that determines that the schedule is possible, and a runtime scheduler that simply executes the schedule according to the EDF rules. In our case we are going to explore this step after basic implementation of EDF, with the assumption of guaranteed schedulablity.

Ultimately, implementation of new algorithms requires testing of that system for workability and performance evaluation. Here eCos itself comes along with a solution, where eCos repository - together with wide range of packages and support for diverse platforms, includes *test cases*, where we can test each of real-time system features, running them separately from each other. For example: after implementation of new scheduling algorithms to the eCos kernel, we can test various kernel features such as threads creation, real-time clock, mutual exclusion semaphores, binary semaphores, message boxes, scheduler locking mechanisms and some others for appropriate behavior. To do these tests we should build our test case application with the eCos library itself and run our test application in the target machine.

In overall, our purpose to modify eCos kernel is to implement more efficient scheduling algorithms, while improving embedded system's performance.

5 Future works and conclusion

Currently we have completed eCos application level API analysis and kernel level interactions between various kernel objects such as – real-time clock, interrupt handlers, synchronization tools (semaphores, message boxes, flags and etc.) and system/application threads. Now we are constantly moving forward for more detailed analysis of kernel scheduling algorithms and construction of new data structures needed for EDF scheduling algorithm implementation. Next step after scheduler implementation is to develop kernel level timing analysis tool for our embedded system.

Acknowledgement

This research was supported by the MKE (Ministry of Knowledge Economy), Korea, under the ITRC (Information Technology Research Center) support program supervised by the IITA (Institute for Information Technology Advancement) (IITA-2008-C1090-0804-0015), and by the Konkuk Univ. WCU project, No, R33-2008-000-10068-0 sponsored by the MEST, Korea.

References

1. Modern Operating Systems, by Andrew S. Tanenbaum, Pearson Education, 2nd edition, p. 132-152, p. 473-475, (2001).
2. Embedded software development with eCos, Anthony J. Massa, Prentice Hall, (2002).
3. eCos home page - http://ecos.sourceware.org/
4. WiBro - Wireless Broadband internet service home page - http://www.wibro.or.kr/
5. Embedded Real-Time OSs: eCos and RTLinux - www.cs.nthu.edu.tw/~ychung/slides/NSoC2005/RTOS.ppt
6. Nomor Research, "Technology of High Speed Packet Access", http://www.nomor.de/uploads/b0/2m/b02mwrVvIa5ZUtVrFeSP1w/Technology_of_HSPA.pdf
7. Smaller, faster, open source, free: the eCos RTOS - http://www.ecoscentric.com/docs/ew06-paper-larmour-ecos.pdf