J. Software Engineering & Applications, 2010, 3: 81-90
doi:10.4236/jsea.2010.31010 Published Online January 2010 (http://www.SciRP.org/journal/jsea)

Analysis and Comparison of Five Kinds of Typical Device-Level Embedded Operating Systems

Jialiang WANG, Hai ZHAO, Peng LI, Hui LI, Bo LI

School of Information Science and Engineering, Northeastern University, Shenyang, China.
Email: wjl123321@163.com

Received September 1st, 2009; revised September 21st, 2009; accepted September 27th, 2009.

ABSTRACT

Today, the number of embedded system was applied in the field of automation and control has far exceeded a variety of general-purpose computer. Embedded system is gradually penetrated into all fields of human society, and ubiquitous embedded applications constitute the "ubiquitous" computing era. Embedded operating system is the core of the embedded system, and it directly affects the performance of the whole system. Our Liaoning Provincial Key Laboratory of Embedded Technology has successfully developed five kinds of device-level embedded operating systems by more than ten years' efforts, and these systems are Webit 5.0, Worix, μKernel, iDCX 128 and μc/os-II 128. This paper mainly analyses and compares the implementation mechanism and performance of these five kinds of device-level embedded operating systems in detail.

Keywords: *Embedded System, Operating System Core, Real-Time Ability, I/O Delay and Jitter, Pervasive Computing and Internet of Things*

1. Introduction

Embedded system has played a significant role in the field of industrial manufacture, process control, instrumentation, consumer electronics and military devices and so on. Embedded operating system has a wide range of space not only in the traditional industrial control and business applications, but also in the field of information household electrical appliances, which has brought much convenience to us [1,2].

The using scope of industrial automation devices based on embedded singlechip has been greatly expanded in recent years. The network is the mainly method not only to improve production efficiency and product quality but also to reduce the cost of human resources, such as the application of industrial control, digital machine tool, grid power system, security power system, device inspection, system monitoring and petrochemicals and so on. Embedded system originated in the age of microcomputer, however the size, price and reliability of a microcomputer are unable to meet the need of majority of embedded system applications. Therefore, embedded system must follow the way of independent development, and this way is just the way of the singlechip. Singlechip has enhanced the fast improvement of embedded technology. Among the field of almost traditional process control, 8-bit singlechip is widely used, but the device -level embedded operating systems can be used for them are very few. So the developing of device-level embedded operating system is very necessary.

2. Embedded Operating System

Embedded operating system is mainly used to monitor and control devices, and general desktop operating system is largely based on the orders of keyboard and mouse. Relatively speaking, the movement of device has very strict timing requirements, but the timing of human action and reflection are not so strict. So for many application fields of device-level control environment, the general desktop operating system is not well competent. Comparing to the micro-computers and large-scale general-purpose computer operating systems, the embedded operating system has the basic features of real-time performance, small core code, preemptive kernel, being configured, reduction and high reliability and so on.

The technology of EI (Embedded Internet) makes it possible of a large number of traditional devices and home electrical appliances instruments achieving network interconnection. It has become a trend for RTOS (real-time operating system) is used in EI applications, and it mainly due to the RTOS not only improves the reliability of the system, but also it can reduce the difficult of development of embedded software and improve development efficiency. Different from general embedded applications, EI applications require RTOS not only has good real-time kernel, but also can provide the capabilities of network protocol stack and certain document management. For most of the existing commercial and free RTOS, they are either very expensive or not having network protocol stack modules, especially for the 8-bit microcontroller with a network of RTOS functions are very few.

Our Liaoning Provincial Key Laboratory of Embedded Technology has successfully developed five kinds device- level embedded operating systems running on 8-bit singlechip, and these systems are Webit 5.0, Worix, μKernel, iDCX 128 and μc/os-II 128. The experimental platform used to develop and test is ATmega 128L manufactured by the United States ATMEL corporation. ATmega 128L is one of the most powerful singlechip in the series of AVR, and AVR singlechip is the first general RISC architecture singlechip, so its process speed and performance has greatly improved than the MCS-51 series singlechip of CISC architecture.

3. Introduction of Five Kinds of Typical Device-Level Embedded Operating System

3.1 Webit 5.0 Embedded Operating System

Webit is an embedded internet device, which has been successfully embedded into the fieldbus devices and can successfully access to devices through the internet. Webit takes full advantage of singlechip system's limited resources, and combined with TCP/IP protocol and high-performance network to process data. Webit has been successfully developed and manufactured by our Liaoning Provincial Key Laboratory of Embedded Technology, and it has passed the appraisal of national science and technology department, and it has also achieved the new product patent of state intellectual property office.

Webit 5.0 operating system directly runs on the device driver, and it mainly achieves the functions of task management, synchronization and communication between tasks. The modules of network file system, protocol stack and I/O management are running outside of system core. The design of task scheduling strategy of Webit 5.0 system adopting the multi-tasking kernel based on priority scheduling, thus the time performance of the kernel is better, and it makes the tasks with high real-time ability can well gain the system resources and have the ability of quick response. The system uses the mechanisms of mailbox and semaphores to achieve synchronization and communication for inter-task, while it also provides effective mechanism of network authentication and user rights so as to ensure the safety of the system. Webit 5.0 is developed in accordance with the method of modular design, using the micro-kernel technology. TCP/IP protocol suite, I/O interface and user interface are separated from the kernel, so it results in the architecture of hierarchy. Each layer corresponds to a module respectively, and each module is separated designed. The call between modules is given interface specification, so it can realize the purposes of flexible reduction based on user demands, and the system is well applied in the field of industrial control.

3.2 WORIX Embedded Operating System

WORIX operating system has the following features:

1) System can accomplish a variety of concurrent operations through multi-task. If the user's application requires more additional tasks, they can modify the constant value of kernel THREAD_NUMS to reset the maximum number of tasks that the system supports;

2) Scheduling algorithm of the system is based on static priority preemptive scheduling, so that system can ensure that high-priority task can have the ability of fast response;

3) The system can support different network protocol stack, and users can develop their own network protocols by wireless modules driver interface supported by the operating system, so it can be well applied in the field of wireless sensor networks;

4) WORIX kernel implements the mechanism of semaphores and mutual semaphores so as to let the task to access to exclusive resources;

5) The clock beat of WORIX kernel was set by the internal timer /counter Timer 0. The timer runs interrupt service program every time interval, thus sleep task and wake-up function were handled in the timer interrupt service.

3.3 μKernel Embedded Operating System

The core of the μKernel operating system retains the most basic and important system services, which including the functions of task management, task scheduling, mutual exclusion semaphores and clock management. Other specific system function modules can be selected in the application part so as to minimize the kernel code size. Time management mechanism is designed as follows: set the time of delay with the unit of clock beat, when the system delays a task for some time, it gets the task from the task ready table, and puts it into the waiting list. This mechanism allows the task can be delayed for a number of clock beat, and it also provides the basis to judge whether waiting tasks exceed their timeout. The design of semaphore aims at establishing a sign of whether the shared resource is occupied. Thus while accessing to shared resource; task may check the sign to know whether the shared resource is occupied. The processing of shared data uses semaphores doesn't increasing the amount of interrupt delay time. If the interrupt service program or the current task activates a high-priority task, the high-priority task will be immediately started.

3.4 iDCX 128 Embedded Operating System

A real-time system must have the ability to respond to the external random events quickly, and the iDCX 128 kernel was used the design method of modular while being designed, and it was implemented by AVR assembler language, so its core code size is relatively smaller. The core mainly realizes the functions of task management, task inter-communication, interrupt handling and timing services, and these services enable users can easily respond to external asynchronous events. The four categories of services are described as follows:

1) Task management provides services as follows: create a user task; delete a task; know function value of task; suspend a task.

2) The service of inter-task communication allows tasks can transmit information each other by exchanging data. By using this service it can achieve synchronization between tasks and shared system resources well. It provides the following services: allocate memory buffer; send information to other tasks; hang a task to let it wait for messages; release the memory buffer.

3) The service of interrupt handling enable tasks to communication with the various types of peripherals, which provides the following services: set interrupt source for task while initialization; disable some interrupts; enable certain interrupts; synchronize task with interrupt.

4) The service of timing is implemented by using of Timer 0 to provide a soft clock for each task in the system, and the soft clock provides time interval (it allows a task perform some functions at a particular time interval) and time out (it gives a task longest limit time allowing the task to wait for), which provides to users the following services: set up time interval; wait for coming of time interval; wait for report of timeout.

The iDCX 128 operating system always let one task to run at certain time and other tasks to wait for the arrival of some incidents or to be in the ready state. Owing to its unique mechanism of time slice, it can mask multiple tasks share the CPU well, and it seems several tasks are running synchronously. Due to the unique communication mechanism of task 0, it can well be applied in the task parallel communication of multiple singlechips.

3.5 μc/os-II 128 Embedded Operating System

The design method of μc/os-II 128 system is similar to the famous μc/os-II operating system. It is the preemptive real-time kernel, which means that μc/os-II 128 is always ready to run the highest priority task in the condition of ready, and μc/os-II 128 can manage 64 tasks orderly with less core code, which retains eight tasks to the system, so it can support 56 tasks for application grogram, and the priority of these tasks is not the same. Because the application of μc/os-II 128 only use those services that system requires, it can reduce the memory space (RAM and ROM) that system required. This scalability is achieved through conditional compilation. μc/os-II 128 allows each task to have a different stack space in order to reduce the application program requirement about the RAM. The μc/os-II 128 can deter-

Table 1. Comparison of mechanism of the five kinds of embedded operating systems

	Webit 5.0	WORIX	μKernel	iDCX 128	μc/os-II 128
Preemptive Kernel	Yes	Yes	No	Yes	Yes
Number of priority	16	6	8	5	64
Change of priority	Static	Dynamic	Dynamic	Static	Dynamic
Method of task scheduling	Time slice around scheduling	Time slice around scheduling	Fixed priority preemptive scheduling	Time slice around scheduling	Fixed priority preemptive scheduling
Support same priority scheduling	Yes	No	No	Yes	No
Number of tasks	32	16	8	8	64
Determinability of time	Yes	Yes	Yes	Yes	Yes
Mechanism of synchronization	Message queue、Incident sign	Semaphore、Exclusive semaphore、Incident sign	Semaphore、Exclusive semaphore、Incident sign	Message queue、Incident sign	Semaphore、Exclusive semaphore、Incident sign
Mechanism of communication	Message queue	Mailbox、Message queue	Mailbox、Message queue	Message queue	Mailbox、Message queue
Method of avoid priority reversion	Priority inherit、Priority top	Priority inherit	Priority inherit、Priority top	Priority inherit	Priority top

mine the stack of each task by the unique handling mechanism of stack space validation function. If the higher-priority task is waked up by interrupt; the high-priority task will run immediately after the withdrawal of all interrupts.

The comparison analysis of the mechanism about the above five kinds of typical device-level embedded operating systems are shown in Table 1.

The determinability of time means that the execution time of embedded real-time system functions has the feature of determinability, which means the implementation time of system service does not depend on the number of application tasks running in the system. Therefore, basing on this feature, the time of system completes certain task can be predicted.

4. Test and Analysis of System Real-Time Ability

The real-time performance is the most critical performance indicators to all control systems. The real-time performance of embedded operating system is measured primarily by the response time that is the time of computer recognizes an external event and before reacting it. The response time is an important indicator for running system. The specific response factors of RTOS are: interrupt delay time and task switching time. The two time factors are described as follows:

4.1 Test and Analysis of System Interrupt Delay

While the real-time operating system running on the state of kernel or performing certain system calls, it will not respond to the interrupt once it arrivals. Only when the real-time operating system returns to the user state, and it can respond to external interrupt requests, so the maximum time required for this process is called the maximum interrupt prohibition time [3–7].

maximum interrupt prohibition time = $T_{closeINT} + T_{doISR} + T_{saveReg} + T_{startService}$, all parts are introduced as follows:

$T_{closeINT}$: The maximum time of closing interrupt;

T_{doISR}: The beginning time of executing the first instruction of interrupt service program;

$T_{saveReg}$ = The time of saving CPU internal registers;

$T_{startService}$ = The execution time of kernel runs system call.

The test tool is oscilloscope (TDS5054B), which has following features: three kinds of bandwidth of 1 GHz, 500 MHz, 350 MHz; 2 and 4 channel; the rate of collecting date is 5 GS/s; the record length can reach to 16 MS, and the capture rate of maximum waveform is 100,000 wfs/s and so on.

The test method is as follows: at the key location of program, set the output instruction of I/O, and make the state of I/O output level exchanges between high and low, so the time can be calculated by the waveform collected by oscilloscope. The test results are shown in Picture 1.

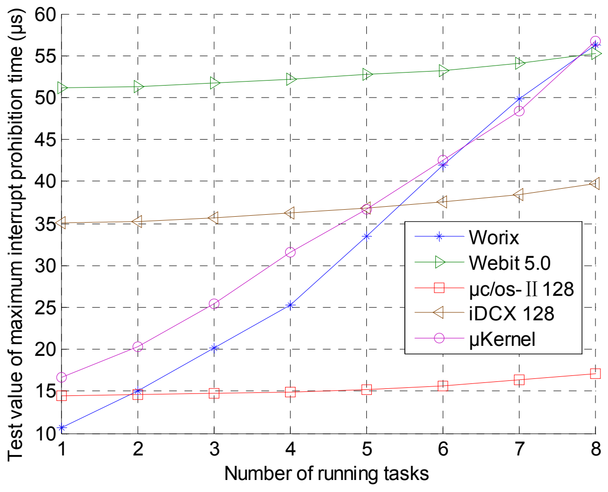


Figure 1. Comparison of the maximum interruption delay time of the five embedded operating systems

The maximum interrupt prohibiting time of the WORIX is the timer interrupt. The execution time of timer interrupt service program is related with the numbers of tasks in the task sleep queue. So the maximum interrupt prohibiting time is not stable. We should calculate the average value by many experiments, and use the average value to reflect the interrupt service program in the timer.

The maximum interrupt prohibiting time of Webit 5.0 also in the timer interrupt. Different from the WORIX system, the processed incident is not sleep queue in the timer interrupt of Webit 5.0, but the timer_info queue of timer. The corresponding timing information is stored in the timer_info queue. In addition to set the sleep time, it also can set the numbers of task sleep. Therefore, Webit 5.0 will handle more related interrupt information, so the timer interrupt time of Webit 5.0 system is longer than WORIX system.

The maximum interrupt prohibition time of μKernel is mainly related with the execution time of timer interrupt service program, and the execution time of timer interrupt service program is related with the numbers of tasks whose sleep time will decrease to 0 in this timer interrupt service program. So the maximum interrupt prohibiting time is also not stable, and we should calculate the average value by many experiments like the WORIX system. The maximum interrupt prohibiting time for iDCX 128 is also in the timer interrupt, and the two ways of handing interrupt are Timer 0 server (handling Timer 0 interrupt) and common server (handing the others interrupt). The handing information of common server are event vector (identify the information that the task is waiting for), event coming (identify the information that the task is waiting for has come) and time out and so on, and determining whether to switch tasks. Timer 0 server mainly finishes the time interval and the handling of task ready table, and it also determines whether to switch tasks.

The maximum interrupt prohibition time of μCOS-II 128 is the most optimal among the five kinds of operating systems. Because the μCOS-II 128 only to wake up the tasks in the sleep queue, and it does not need to clear the task from the sleep tab and put it into the task ready tab, it only needs to modify corresponding bit of task priority identification can complete change of the task priority status. Because the time complexity about operating linear form is O(1), the task switching time of μCOS-II 128 is basically stable, and it is not related with the number of tasks running on the system, task status and task priority. So it also reflects the good real-time ability of μCOS-II 128. However, μCOS-II 128 obtains a very small interrupt prohibition time basing on the use of more data structures and taking up more memory space.

4.2 Test and Analysis of System Task Switching Time

The task switching time means that while a task out of running, the real-time operating system will save its running information, and put it into information queue, and then choose a new task to let it run, so the needed time of this progress is named task switch time. It actually means the time that CPU stops a task and switches to run another task [8–13].

Task switching time=$T_{\text{to Do B Task Time}}-T_{\text{to Pause A Task}}$:

$T_{\text{to Do B Task Time}}$=The beginning time of running task B;

$T_{\text{to Pause A Task}}$=The time of stopping running task A.

The test results are shown in Picture 2.

The task switching time mainly consists of three parts: time of accessing to interrupt, saving and recovering related information and running system call. As the processor platform of the five kinds of operating system are same, so the time of accessing to interrupt, saving and recovering related information is basically same. Therefore, the difference of task switching time is due to the difference of system call time. By analyzing the scheduling mechanism of these five kinds of operating systems can explain the reasons about the different task switching time of these operating systems.

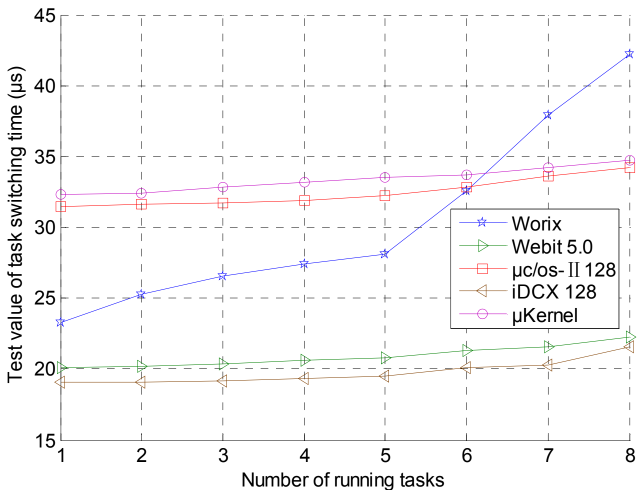


Figure 2. Comparison of the task switching time of the five embedded operating systems

The ready queue of WORIX operating system using a pointer array, and the elements of the array points to the pointer of the beginning of correspondingly priority ready queue. While the system schedules tasks, it will traverse the ready_queue from beginning, if the content pointed by the current element location is empty, it will continuously traverse to later elements of the ready_queue; if the content pointed by the current element location is not empty, then it will fetch the task in the beginning of the ready_queue pointed by the current element, and this task will be run as the next task. We can know that the scheduling time of WORIX is related with the priority of the ready task from the scheduling mechanism. While there has tasks whose priority are 0 among the tasks in the state of ready, the scheduling time of the system is shortest; and while priority of all the tasks is lower, the scheduling time of the system is longest, so the scheduling time is not stable. Due to the scheduling time towards higher tasks is shorter for WORIX, so it can well be used in the wireless sensor network. Because the wireless receiving and sending task with higher priority can be switched quickly, the wireless receiving and sending ability of the system is well.

The ready_queue of Webit 5.0 operating system is a single-linked list sorted by priority. The tasks in the ready_queue are listed strictly in accordance with the priority order from high to low. While each time creates some new tasks or tasks from other states to switch to the state of ready, and the tasks becoming ready state will insert into the appropriate position of the ready queue. Thus, at any time, the tasks in the ready_queue of Webit 5.0 is strictly sorted in accordance with the priority order from high to low, and every time system schedules task. The scheduling program only needs to fetch the task in the beginning of the ready_queue can complete the scheduling progress, so the scheduling time of Webit 5.0 is relatively shorter, and the time of scheduling is stable.

For the scheduling of the μCOS-II 128 operating system, it firstly makes certain the position of the group with the highest priority ready task, and then finds the highest priority task within the group. By this way it can easily obtain the highest priority task. The priority of μCOS-II 128 is in correspondence with the related task, so by finding the priority of the task it can find task control blocks of the task to be run. Only by this way can complete the process of scheduling. As the scheduling mechanism of μCOS-II 128 is the way of "table-driven", the overhead of the scheduling time is little smaller and the system has very good predictability, thus the ability of real-time can be guaranteed. From the test results we can know that the scheduling time of μCOS-II 128 is very stable, and the scheduling time is relatively little lower in the five kinds of operating systems. However, due to the scheduling way of this "table-driven" requires additional storage space of RAM and ROM, it does not apply to resource-limited applications.

For the μKernel operating system, while it happens task switching each time, the time of saving context of running task and recovering the context of waiting task is always similar. As μKernel system uses the fast localization algorithm, the system has a very good predictability, and the real-time ability also can be guaranteed. From the test results we can know that the scheduling time of μKernel is also very stable, but the task switching time is little larger than μCOS-II 128. The difference of task switching time is due to the difference of selecting task to be run from the task ready_queue. As the function of system scheduling function OSSched() in the μKernel operating system is to save context of current running task, and also choose a new task and recover the context task of new task, the running time of scheduling function OSSched() to be tested is just the time of task switching task of μKernel.

For the iDCX 128 operating system, its tasks are sorted by priority in the task_ready_tab, and every task is sorted in accordance with the order of priority from high to low. Each memory unit of task_ready_tab stores the ITD and priority of tasks' at the same time. While creating some new tasks or some tasks becoming the state of ready, these tasks will insert into the task_ready_tab by the order of priority, Thus, at any time, the task_ready_tab of iDCX 128 is sorted by the order of priority. While it schedules tasks, the scheduling program only needs to fetch the task in the beginning of the task_ready_tab can complete the scheduling process, so the scheduling time is the most smaller among these five kinds of operating systems, and the scheduling time is also stable.

4.3 Test of System Core Size

The minimum operating system kernel code space means the program space of the system needed to complete the most basic functions. Minimum kernel code space is also an important factor to evaluate an operating system. As the operating system kernel loads different basic functions each times, the minimum value of the kernel code is not exclusive, which means the smallest value of the kernel code is relative.

The testing tool is the AVR STUDIO software provided by ATMEL corporation, and it is integrated developing environment that used to program AVR series singlechip. The software has the following features: project manager; source code editor; assembler compiler; software simulation function (support assembly and high-level language); ICE real-time simulation function (with using simulator); supporting the AVR Prog serial programmer and STK500/AVRISP/JTAG ICE tools and so on. Using the AVR Studio 4 software to test the core size of the five kinds of operating system, and the test results are shown in Table 2.

Table 2. The test of kernel of the five kinds of embedded operating system (Unit:KB)

system	iDCX 128	Webit5.0	Worix	μKernel	μc/os-II 128
core	3.11	3.81	3.32	3.58	3.91

The size of core code is an important factor to measure an operating system code, as the storage resource of device-level singlechip is very limited. So the less core code can accommodate more application program, and can also leave more code room for users to program.

5. Analysis and Comparison of System I/O Delay and Jitter

The feature of physical in the hard real-time system reflects that there are inevitable phenomenon of I/O (Input/Output) delay and jitter in the process of device starting, and this phenomenon mapped to the real-time system is that there are inevitable exists I/O delay and jitter in the real-time scheduling. The existing of the I/O delay and jitter may effect the synchronous of different devices controlled by the operating system, and also the stability and reliability of the system. How to control I/O delay and jitter of hard real-time task has become a hot research issue for the real-time scheduling of device-level operating system.

For the following period task model, every period task is a series of basic working units that can be scheduled and run by system, and it is expressed by τi. Period Ti of task τi is the smallest value of tasks releasing time interval, and the controlling time Ci is the maximum running time of all tasks. Di is the relative deadline of task, and it means the tasks of τi in the releasing time of t must be finished in the time unit of Di after the time of t [14,15]. The period and controlling time of system is known for all the following analysis.

5.1 Test and Analysis of I/O Delay of Fixed Priority Operating System

The task critical time means while a task τi ($1 \leqslant i \leqslant n$) and all other tasks hp(i) that having higher priority than it are released synchronously, and the task τi has maximum responding time at that moment. So the critical time of maximum I/O delay for task τi can be described that the task is just beginning to run and it was immediately be interrupted by high priority task hp(i), and all higher tasks are released at this moment. We use the scheduling method with preemptive threshold value, and allow a time threshold value, and calculate it by the preemptive and not preemptive parts respectively.

The maximum I/O delay of preemptive part of task τ_i is:

$$L_i^p = TH_i + \sum_{k \in hp(i)} \left\lceil \frac{L_i^p}{T_k} \right\rceil \cdot C_k \quad (1)$$

L_i^p is the least solution that meets (1)，it can be calculated with the original value of $L_i^{p\ (0)} = TH_i$ by iterative calculation.

The maximum I/O delay of not preemptive part of task τ_i is:

$$L_i^{np} = C_i - TH_i \quad (2)$$

So the maximum I/O delay of task τ_i based on fixed priority scheduling is:

$$\begin{aligned} L_i^{max} &= L_i^p + L_i^{np} \\ &= C_i + \sum_{k \in hp(i)} \left\lceil \frac{L_i^p}{T_k} \right\rceil \cdot C_k \end{aligned} \quad (3)$$

$hp(i)$ is the task set whose priority is higher than the task τ_i, and L_i^{max} is the least solution that meets (3), and it can be calculated with the original value of $L_i^{max\ (0)}=C_i$ by iterative calculation.

5.2 Test and Analysis of I/O Jitter of Fixed Priority Operating System

The minimum I/O delay is the least responding time for fixed priority scheduling, so the minimum I/O delay of preemptive part of task $\tau_i\,(1 \le i \le n)$ is:

$$L_i^{(b)p} = \min(C_i^b, TH_i) + \sum_{k \in hp(i)} \left\lceil \frac{L_i^{(b)p} - T_k}{T_k} \right\rceil \cdot C_k^b \quad (4)$$

$L_i^{(b)pree}$ is the maximum solution that meets (4), and it can be calculated with the original value of $L_i^{(b)pree(0)}=T_i$ by iterative calculation.

The minimum I/O delay of not preemptive part of task τ_i is:

$$L_i^{(b)np} = \max(C_i^b, TH_i) - TH_i \quad (5)$$

So the minimum I/O delay of the task τ_i based on the fixed priority scheduling is:

$$L_i^{min} = L_i^{(b)p} + L_i^{(b)np} = L_i^{(b)p} + \max(C_i^b, TH_i) - TH_i \quad (6)$$

The maximum I/O jitter of not preemptive task τ_i based on the fixed priority scheduling can be calculated by the combination of (3) and (6):

$$J_i^{max} = L_i^{max} - L_i^{min} \quad (7)$$

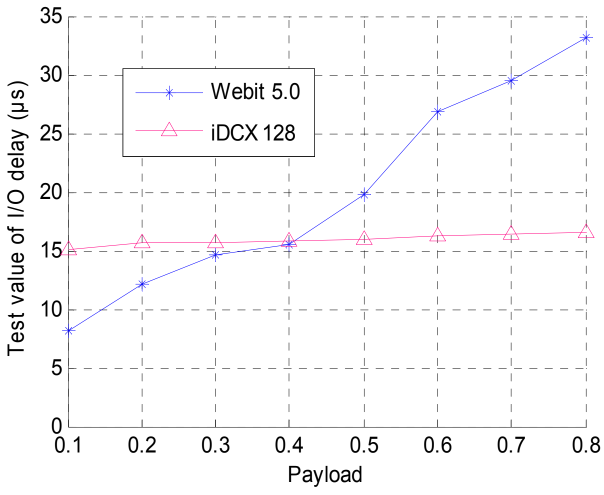
The operating system of iDCX 128 and Webit 5.0 are both based on fixed priority, testing the I/O delay and jitter while there are 7 tasks running on the system, and the test results are shown in Figure 3.

As we can see from the Figure 3, the Webit 5.0 operating system can well optimize the I/O jitter than iDCX 128 operating system under the condition of the system can be scheduled. But with the increasing of system payload, the optimized effect of Webit 5.0 is decreasing continuously, and the cost of this strategy is that the average I/O delay time of Webit 5.0 is much larger than iDCX 128 while system payload exceeds 0.4.

5.3 Test and Analysis of I/O Delay of Dynamic Priority Operating System

Allocate preemptive time threshold $TH_i\ (0 \le TH_i \le C_i)$ for task $\tau_i\,(1 \le i \le n)$, and the value of I/O delay created by EDF scheduling needed to be calculated by the preemptive and not preemptive part respectively.

The maximum I/O delay of preemptive of task τ_i is:



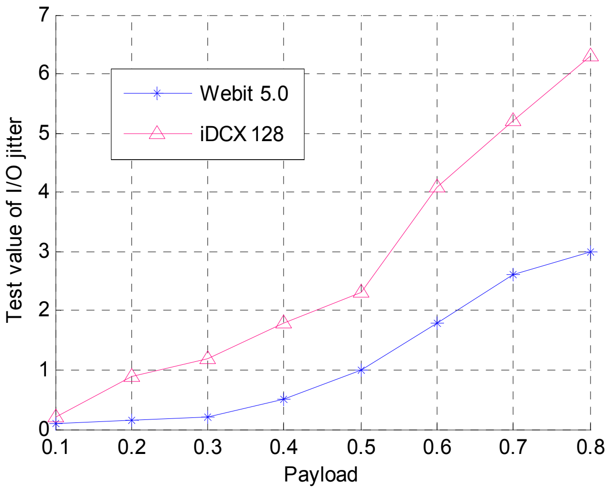


Figure 3. The test of I/O delay and jitter of iDCX 128 and Webit 5.0

$$L_i^p(a) = \max\left\{TH_i, I_i^p(a) - a\right\} \quad (8)$$

And among it includes:

$$I_i^p(a) = W_i(a,t) + \left\lceil \frac{a}{T_i} \right\rceil \cdot C_i + TH_i + \max_{D_j > D_i}(C_j - TH_j) \quad (9)$$

$L_i^{pree}(a)$ is the least solution that meets (9), and it can be calculated with the original value of $L_i^{pree(0)}=0$ by iterative calculation. For a job of task τ_i at the time of a, its payload of higher job is:

$$W_i(a,t) = \sum_{\forall j: D_j < L_i^{(b)p}} \left\lceil \frac{\min\left\{L_i^{(b)p}, D_i - D_j\right\}}{T_j} - 1 \right\rceil \cdot C_j^b \quad (10)$$

And the maximum I/O delay of not preemptive part of task τ_i is:

$$L_i^{np} = (C_i - TH_i) \quad (11)$$

So the maximum I/O delay of task τ_i based on EDF scheduling is:

$$\begin{aligned} L_i^{max} &= L_i^p(a) + L_i^{np} \\ &= \max\left\{TH_i, I_i^p(a) - a\right\} + (C_i - TH_i) \end{aligned} \quad (12)$$

5.4 Test and Analysis of I/O Jitter of Dynamic Priority Operating System

The value of least I/O delay of EDF scheduling is equal to the least responding time, so the minimum I/O delay of preemptive part of task $\tau_i(1 \le i \le n)$ is:

$$L_i^{(b)p} = \min(C_i^b, TH_i) + W_i(a,t) \quad (13)$$

$L_i^{(b)pree}$ is the maximum solution that meets (13), and it can be calculated with the original value of $L_i^{(b)pree(0)}=T_i$ by iterative calculation.

The minimum of I/O delay of not preemptive part of task τ_i is:

$$L_i^{(b)np} = \max(0, C_i^b - TH_i) \quad (14)$$

So the minimum I/O delay of task τ_i based on the EDF scheduling is:

$$\begin{aligned} L_i^{min} &= L_i^{(b)p} + L_i^{(b)np} \\ &= \min(C_i^b, TH_i) + \max(0, C_i^b - TH_i) + W_i(a,t) \\ &= C_i^b + \sum_{\forall j: D_j < L_i^{(b)p}} \left\lceil \frac{\min\left\{L_i^{(b)p}, D_i - D_j\right\}}{T_j} - 1 \right\rceil \cdot C_j^b \end{aligned} \quad (15)$$

The maximum I/O jitter of task τ_i based on EDF scheduling can be calculated from (12) and (15):

$$J_i^{max} = L_i^{max} - L_i^{min} \quad (16)$$

The operating systems of Worix, μKernel and μc/os-II 128 are all based on the dynamic priority scheduling, testing the I/O delay and jitter while there are 7 tasks running on the system, and the test results are shown in Figure 4.

Compared with other operating systems, the Worix can well optimize the I/O jitter, and the I/O delay will not increasingly obviously, and the ability of scheduling can also be guaranteed. The I/O jitter count of μKernel is middle, but the delay time is much longer. But the μc/os-II 128 operating system achieves less delay at the cost of having much I/O jitter count.

6. Future of Embedded Operating System – Pervasive Computing and Internet of Things

Pervasive computing was first proposed by American Mark Weiser in 1991. It refers to a new ubiquitous

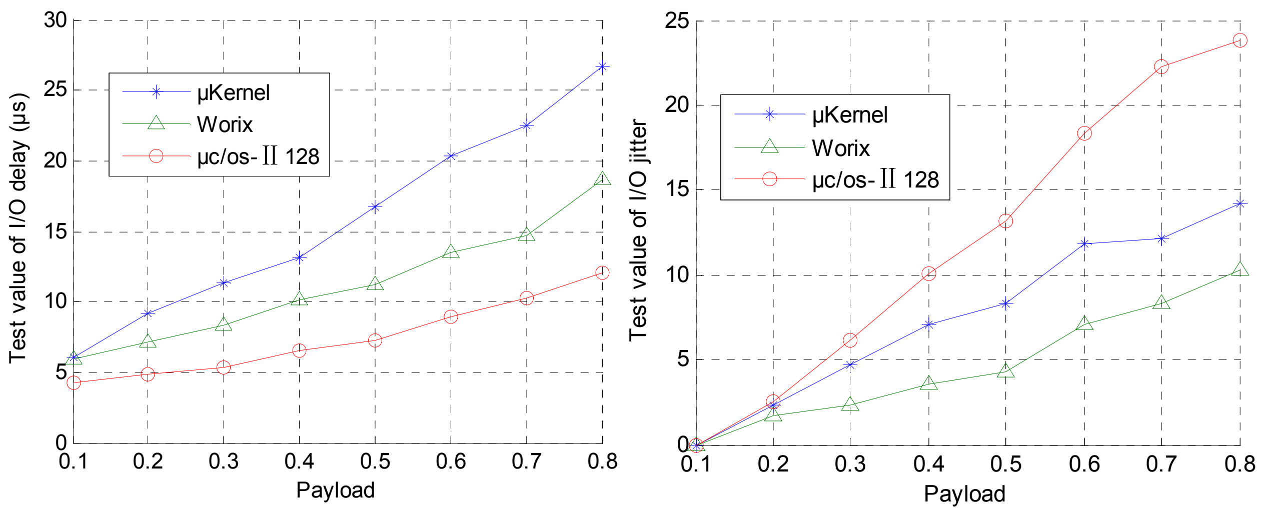


Figure 4. The test of I/O delay and jitter of Worix, μKernel and μc/os-II 128

method of computing can be applied to various information devices. In the era of pervasive computing, computing devices and the computing environment emerges together closely, and people can access to information and processing at any time and any place, and they will not fell the process of computing at all while using it. It was also considered as the next generation computing model, and the ubiquitous computing devices used are mobile computing devices mostly. Mobile computing devices are essentially part of embedded devices, so it can be said that ubiquitous computing constitutes the indispensable operating platform of the embedded system. The rapid development of embedded systems is also a strong impetus to the fast development of pervasive computing. While all physical objects of real world are linked by the internet and can be managed intelligently, it means the age of Internet of Things is coming, and people can access to appliances information easily through the internet, but this process is inseparable from the large support of operating system. So the device-level embedded operating system is the indispensable key technology to the coming of Internet of Things.

7. Conclusions

Embedded operating system with stable performance has played a very crucial role to the normal operation of devices. So the embedded operating system is considered as the cornerstone in the field of device control. This paper describes the features and implementation mechanisms of the five typical device-level embedded operating systems that are independently developed by our laboratory, and also tests and analyzes the real-time performance, I/O delay and jitter. The five kinds of operating systems has well applied in the fields of Chunlanjing air-conditioning, Taiwan Guanyu uninterruptible power supply, network intelligent devices, and monitoring devices and so on. As the five kinds of operating system cores all have good portability and performance by long-term using, and the most important is that they have the advantages that they can be easily operated in other device-level singlechip platforms in the fields of automation and device control, etc. So it brought broad application prospects for users to choose device-level embedded operating systems.

8. Acknowledgments

This work is supported by the Cultivation Fund of the Key Scientific and Technical Innovation Project, Ministry of Education of China (NO708026).

REFERENCES

[1] H. Zhao, “Embedded Internet [M],” Beijing: Tsinghua University Press, pp. 27–40, 2002.

[2] F. Robert, “Embedded Internet systems come home [J],” IEEE Internet Computing, Vol. 40, No. 14, pp. 52–53, 2001.

[3] Jacek W. “Embedded Internet technology in process control devices [J],” IEEE Internet Computing, Vol. 34, No. 3, pp. 301–308, 2000.

[4] R. Bergamaschi, S. Bhattacharya, and R. Wagner, “Automating the design of SoCs using cores [J],” IEEE Design &Test of Computers, , Vol. 18, No. 5, pp. 32–45, 2001.

[5] A. Garcey, and V. Lessey, “Design to time real-time scheduling [J],” IEEE Transcations on Systems, Man and Cybernetics, Vol. 23, No. 6, pp. 58–67, 1993.

[6] M. Hiroyuki, and E. Thomas, “An improved randomized on-line algorithm for a weighted interval selection problem [J],” Journal of Scheduling, Vol. 7, No. 4, pp. 293–311, 2004.

[7] D. L. Liu, X. B. S. Hu, D. L. Michael, *et al.* “Firm real-time system scheduling based on a novel QoS constraint [J],” IEEE Transactions on Computers, Vol. 55, No. 3, pp. 320–333, 2006.

[8] J. P. S L. Lehoczky, “Performance of real-time bus scheduling algorithms [J],” ACM Performance Evaluation Review, Vol. 14, No. 1, pp. 44–53, 1986.

[9] E. Tavares, P. Maciel, and B. Silva, “Modeling hard real-time systems considering inter-task relations, dynamic voltage scaling and overheads [J],” Microprocessors& Microsystems, Vol. 32, No. 8, pp. 460–473, 2008.

[10] Y. Zou, M. Li, and Q. Wang, “Analysis for scheduling theory and approach of open real-time system [J],” Journal of Software, Vol. 14, No. 1, pp. 83–90, 2003.

[11] C. L. Liu, and J. W. Layland, “Scheduling algorithms for multiprogramming in a hard-real-time environment [J],” Journal of the ACM. Vol. 20, No. 1, pp. 46–61, 1973.

[12] M. Sabeghi, P. Deldari, and S. Khajouei, “A fuzzy algorithm for scheduling periodic tasks on multiprocessor soft real-time systems [C],” Proceedings of the 17th IASTED international conference on modelling and simulation, May, Montreal, Canada, pp. 436–442, 2006.

[13] D. Liu, W. Y. Xing, R. Li, C. Y. Zhang, *et al.* “A fault-tolerant real-time scheduling algorithm in software fault-tolerant module [C],” Proceedings of the 7th international conference on Computational Science, Part IV. Beijing, China, pp. 961–964, May, 2007.

[14] D. D. Luo, “Optimizing scheduling for hard real- time tasks in embedded systems [D],” Shenyang: Northeastern University, 2008.

[15] M. Caccamo, G. Buttazzo, and D. C. Thomas, “Efficient reclaiming in reservation-based real-time systems with variable execution times [J],” IEEE Transactions on Computers, No. 2, pp. 198–213, 2005.

Research Background

The appearance of embedded internet technology enables a large number of traditional devices and home electrical appliances to achieve network interconnection. Powerful performance and strong stability of the embedded operating system will effectively control and make use of embedded devices. Some of existing RTOS having the function of network are very few especially for the 8-bit microcontroller RTOS. Therefore, Our Liaoning Provincial Key Laboratory of Embedded Technology has successfully self-developed five kinds device-level embedded operating systems running on 8-bit singlechip, these systems are Webit 5.0, Worix, μKernel, μc/os-II 128 and iDCX 128. These five systems bring broad selection and apply space for the device-level operating system.