Upgrade of Radiation Monitoring System for National Synchrotron Radiation LAB

Wang Ruopeng
Engineering Physics Dept., Tsinghua University
Email: WANGRP@thinker.ep.tsinghua.edu.cn

Abstract

The prototype of this Radiation Monitoring System, namely Digital Data Logger (DDL), has been on service for nearly 10 years. It is used as local station of gamma-ray radiation monitor for large accelerator or nuclear facilities by National Synchrotron Radiation LAB in China. Now the necessity comes up that the information sharing with other computer systems and a higher performance of hardware and software must be achieved. For this aim, the main control MCU is upgraded, system memory enlarged and power system improved. However our chief revision is on software. Here we applied a tiny real-time multitasking kernel to improve the overall performance of the system. Since the kernel is written in efficial assembler language, it itself takes up minimum system resources while functioning basically like any commercial real-time system. We have done some modifications on the kernel and made it accustomed to our specific situation. Considerate time and effort are saved by this way on software development. The system can work more smoothly, with shorter response latency, less chance of fatal running errors and higher resistance to electric and magnetic disturbance.

1 Objective

In 1987, an Environment Radiation Monitoring System, namely Digital Data Logger (DDL), was set up for the National Synchrotron Radiation LAB in China,Hefei. It is mainly used as local station of gamma ray and neutron radiation monitor for accelerators or nuclear facilities.

The first and most significant function of this device is to collect current pulse specifying detection of gamma rays or neutrons from ionization chamber (There can be 16-channel counters working concurrently). The monitor counts the pulse one by one and stores the result in memory for further inquiry.

The second function is to calculate the dose rates of all the channels every 6 seconds.

Thirdly, the monitor will check and report the running status of the accelerator from a port of 32-channel inputs.

If the dose rate of every channel is above either of two different thresholds, the monitor will display the warning message to a 16-channel output port.

The Monitor can also keep and maintain a clock to work without break, which is necessary in order to collect radiation data continuously for a long period. It is able to receive several kinds of commands from operator through RS-232 port, to provide a measure for uploading data, or modifying parameters.

During nearly 10 years of operation, the monitor has given us much about the additional dose rate attributed by running of the accelerator. What is more, it can often give us suggestions when some bugs occur and we have to do something to diagnose the whole accelerator. This is greatly invaluable especially at time when all other measures fail.

However, now we find several shortcomings in the old version of the monitor.

First, the storage capacity of the monitor is not large enough since it can not store all the 16-channel data needed continuously for a whole month, for the convenience of operation.

Second, the former monitor is devised with M6809 MCU, which is also out of date - it is actually discontinued from production for some time and difficult for further hardware or software development or maintenance.

Third, the human interface of the former monitor is far from perfect. To set parameters, one has to pass several trivial routines and the one-line display of the former monitor gives a poor effect for instant data report.

Fourth, because the monitor has to do, maybe, several jobs at the same time, i.e. when the monitor is refreshing the display, it will be rather difficult to have our input commands responded. In other words, it is not ″real-time″.

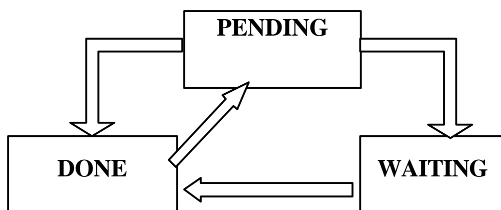
2 Measures of fulfillment

Concerning the deficiencies mentioned above, we did a great deal of hardware and software revision on the old system. The hardware framework, the crate, is preserved and the CPU board, memory board are replaced. A more popular and powerful MCU, M68HC11 is applied with memory expanded to 128K byte (and still further expandable), sufficient for a storage of one full month without any data loss. On the control panel, the old one-line ASCII LCD gives way to a new 320X200 color graphic LCD, which can lively show dose rate of every channel, graphically tell us the switches status of the whole accelerator, and something else. Further, via RS-232, the monitor can be conveniently connect to a PC, on which a Windows program is running, when the operator wants to access the data already stored. Various configuration parameters are also transferred to the monitor through RS-232. A rechargeable cell is incorporated in the device to ensure the continuous operation even in case of AC power failure.

However our chief revision is on software. Here we applied a tiny real-time multitasking kernel to improve the overall performance of the system. The kernel, namely MCX11, is written in efficient assembly language, it itself takes up minimum system resources while functioning

basically like any commercial real-time systems. It s code takes up only 1.3K byte when in full operation, the consumption on RAM is barely 5K byte - mainly for stacks of application written in C language which normally eat up large amount of memory. That is the reason why it is very appropriate for 8-bit MCU application.

MCX11 is a real-time preemptive multitask kernel. Up to 64 tasks can reside in memory simultaneously, which is sufficient for complicated usage on M68HC11. Every task has its own independent stack as well as its code. The tasks are each given a priority, which is a reference of system schedule and can not be changed by user after the kernel starts up. Different status has to be given to tasks since at any moment there can be only one task running. The following is a diagram depicting a task transferring between three different status.



A PENDING state indicates that the event associated with the semaphore has not yet occurred and is therefore pending. The WAITING state shows that not only has the event not yet occurred but a task is waiting for it to happen. The DONE state tells that the event has occurred. MCX11 semaphores have a very strict state transition protocol which is automatically managed by MCX11.

The communication and synchronization between the tasks are achieved with semaphores, messages, and message queues which are all maintained internally by the kernel. Dozens of system functions are provided to access these internal facilities, to operate on tasks, and so on. The following is a list of some system functions incorporated in MCX11:

- Delay: delay a task for a period of time
- Dequeue: get an entry from a FIFO queue
- Enqueue: insert data into FIFO queue
- Execute: execute a task
- Pend: force a semaphore to a PENDING
- Purge: purge a task's timers
- Receive: receive a message
- Resume: resume a task
- Send: send a message to a task
- Sendw: send a message and wait
- Signal: signal a semaphore
- Suspend: suspend a task
- Terminate: terminate a task
- Timer: start a timer
- Wait: wait on semaphore

We have made some modification on the kernel and made it accustomed to our specific situation. Because multitasking is supported by the kernel, functions can be fulfilled one by one with independent modules, thus minimizing the normal efforts to write codes for scheduling, timing and cooperation jobs between different modules. Considerable time and effort are saved by this way on software development.

Now it has become quite easy for one to define his/her own tasks and start the kernel on a specific hardware platform. Below is a list of jobs needed to run MCX11 for our purpose.

- Devide your target system into several relatively less interdependent blocks.
- Convert these blocks into different tasks, and give each task a priority according to their significance – requirement on processing latency.
- Select appropriate programming language (Assembly or C), and write module codes for the tasks.
- Submit a form to kernel and specify everything about number of tasks, entry code and stack size preserved for each task, and so on.

The following is a brief frame of this kind of form.

```
TSKNUM  EQU 21     Set up total number of tasks
ORG    TCBDATA

* define task #1
TASK1    EQU    STATLS+TCBLEN
              * Task #1 TCB address
STAK1    EQU    STKBASE-20
              * Base address of task #1 stack
STK1SIZ    EQU    18
              * Size of stack for task 2
FCB    0    * INITST is RUN
FDB    Task1_entry, STAK1, TASK1
              * STRTADR, RSTSP, TCBADDR

* define task #2
TASK2    EQU    TASK1+TCBLEN
              * Task #2 TCB address
STAK2    EQU    STAK2-STK2SIZ
              * Base address of task #2 stack
STK2SIZ    EQU    21
              * Stack size for task #2
FCB    0
              * INITST is RUN
FDB    Task2_entry, STAK2, TASK2
              * STRTADR, RSTSP, TCBADDR
```

As to the great conveniences the kernel provides to small control system, MCX11 on M68HC11 has a quite good prospective to be further applications.