TaraxOS: An Operation System for Wireless Sensor Networks

Wen-Yong Wang, Wen-Gang Han, and Yu Xiang

Abstract—An operation system (OS), named TaraxOS for wireless sensor networks is proposed. The functions of the TaraxOS such as interrupt, memory management and scheduling mechanism are researched and implemented. After introducing the node’s working flow, the performances of the TaraxOS are analyzed and some limitations of the scheduling mechanism are discussed. The obtained results show that the proposed TaraxOS has some desirable characteristics including small code capacity, low power consumption, quick response and robustness.

Index Terms—Data processing, interrupt, memory management, scheduling mechanism, TaraxOS.

1. Introduction

Operation system (OS) is the core of the software system on the node of wireless sensor networks (WSN) [1]. There are many kinds of embedded OS at present, such as Vxwork, Windows CE, Neculeus etc. [2], which are rich in APIs and embed application software; but for sensor apparatus, these embedded OS are too huge.

In order to satisfy diversity of the wireless sensor networks applications, we design an operation system according to the following conditions: 1) small code capacity: as node’s resource is so limited, OS’s kernel code capacity must be small; 2) low power consumption: a low-power-consumption OS is helpful for prolonging the lifetime of the node; 3) quick response: AD nodes must be able to sample data off environment timely and send them back to sink node; 4) robustness: node should always keep robust in certain environments.

Combing with the aforementioned conditions, we design an OS called TaraxOS suitable for TaraxNode, a WSN node with independent intellectual property rights that is researched and produced in our lab.

2. Design of TaraxOS

TaraxNode’s hardware includes micro-control unit (MCU) named TaraxCore and an 8-bit RISC processor with low consumption. Wireless receiving/transmitting module based on Chipcon CC2420 is suitable for IEEE802.15.4 standard. Data acquiring module includes different kinds of sensors and analog-to-digital (A/D) with independent intellectual property rights. According to the design principles discussed in the first section, we develop a high efficiency, low consumption and single-kernel OS for TaraxNode named TaraxOS.

The relationship among OS kernel, hardware and application are shown in Fig. 1.

TaraxOS’s kernel mainly includes: task scheduler acting as the core of the system, memory management and wireless communication interface, as shown in Fig. 2.

Node starts to run on the base of memory pre-allocated. After accessing the networks via wireless protocol, with tasks posted by interrupt, the system begins to process tasks. As an important part, we take the following settings.

Manuscript received July 16, 2008; revised August 12, 2008.

The authors are with Software Engineering Research Center, University of Electronic Science and Technology of China, Chengdu, 610054, China (e-mail: wangwy@uestc.edu.cn).
2.1 Task Scheduling

We adopt first input first output (FIFO) as the task scheduling mechanism in TaraxOS. In wireless sensor networks, a single node’s hardware resources are so limited that they can not maintain traditional process scheduling mechanism. So we use light-weight thread technology and two-level-scheduling mechanism which could make full use of the limited resources of the node. In this way, normal light-weight threads (i.e. tasks) are scheduled in FIFO, these threads are not preemptive. The hardware processing threads, namely interrupt processing thread (i.e. events), are allowed to interrupt the user’s light-weight thread in order to response the hardware interrupt immediately. Atomic-operation or synchronized primitive is needed to provide access protection while sharing resources.

In order to cut down the runtime of the interrupt service routine (ISR), design for ISR must be reduced to shorten interrupt response time. We encapsulate codes which need not to be executed in real time into functions as task in ISR. As ISR executing, address of the task number is put into task queue, then tasks will be executed on schedule by kernel after exiting ISR. Kernel uses a circle queue to maintain a task list. In default, the size of the task list is 8. Fig. 3 describes an empty task queue and represents a queue with 3 tasks waiting to be processed. Kernel schedules tasks according to the order as tasks’ entering sequences, that is, FIFO. The function TOSH_run_next_task( ) is to get and execute task pointed by the task pointer TOSH_sched_full. TOSH_Sched_free points to the location of next task which will enter the queue. The max number of the tasks in the queue is 8. The TOSH_sched_full and TOSH_sched_free are initialized to zero at the beginning of the scheduling algorithm. Kernel calls TOSH_run_next_task( ) in a infinite loop, then all the task functions execute while the task queue is not empty.

Post task:

```c
TOS_post(unsigned int tp)
{
    i=*(TOSH_sched_free);
    if((TOSH_queue_i_tp)==NULL)
    {
        *(TOSH_sched_free)=tp;
        *(TOSH_queue_i_tp)=tp;
        Return success;
    }

    TOSH_sched_free=0
    TOSH_sched_full=0
```

Fig. 3. Task queue.

2.2 Memory Management

The memory system of the TaraxCore includes 128K×13 program ROM, 4K×8 data RAM and 128×8 bits common registers. Program ROM is used to store executable code of the program and some initial values. 4K×8 data RAM is mainly used to store data, SRAM with 128×8 bits common register array is divided into four banks to store variables during compiling.

As applications are comparatively simple and the number of tasks is a const, in order to achieve higher code compile efficiency, we pre-allocate 4K×8 RAM to global variables of the program, task list, task scheduler, interrupt protection, the buffer for the TX/RX task and so on before booting the system. Common registers area is becoming variables’ buffer, storing public variables. We also design the data exchanging mechanism among public and pre-allocate variables using chip’s I/O port to read and write.

2.3 Interrupt Management

System’s interrupt sources are as following: SFD interrupt/FIFOP interrupts of the CC2420, ADC interrupt, timer interrupt, UART interrupt. There is no software interrupt. UART interrupt occurs only on the sink node connected to the gateway. In our system, we adopt a mechanism called multi-sources interrupt, namely, only one interrupt vector (i.e. only one interrupt service subroutine entry address) corresponds to multiple interrupt sources. After entering into interrupt program via interrupt vector, the system will query and confirm which interrupt sources triggers the interrupt, and then calls corresponding ISR.
The system uses software method to assign different interrupt sources priority and put some of them with higher priority anterior in the query program. The order of the query is timer/SFD/FIFOP/ADC interrupt.

In Fig. 4, we set the highest priority to timer interrupt because timer is benchmark for route construction and data acquiring, and timer interrupt is the most pivotal in the system. SFD and FIFOP interrupt are concerned with receiving and transmitting data frames and routing frames either from local or from other nodes. AD interrupt occurs after local node’s sampling, so it has the lowest priority.

3. Working Flow

3.1 System Initialization

The initialization of TaraxOS is as following.
1) Board initialization.
2) Reading TaraxOS into RAM from ROM.
3) System initialization: this step mainly implements hardware and software initialization which includes: a) SPI initialization mainly setting the interface between MCU and CC2420, b) CC2420 initialization, c) AD module initialization, d) TCC initialization, setting prescalar coefficient, clock signal source, interrupt interval, e) communication protocol initialization, and f) task queue initialization.
4) Entering into operation state.

Now there is no task in the queue, the system enters into sleep state. As long as an interrupt occurs at any time, MCU will be aroused, interrupt is processed and tasks are posted. Tasks are put into task queue of the scheduler, waiting to be processed. The processing of tasks may be broken by interrupt, after dealing with interrupt, the system keeps on running tasks and it works in that loop.

3.2 Node’s Self-Organizing

Node should access network before sending data. A repeat timer is set for this and is triggered one time at each period of time of hello-interval in timer ISR. When time expires, the system will post tasks of constructing route packets, then send packets into channel. When neighbor receives a packet, it will add source node’s route information into the neighbor table if there is no corresponding entry for source node. Looking like lipper, route information diffuses in the network from the sink node whose default hop is zero. By this, all the nodes access network gradually. After sampling data, AD nodes transfer data to sink node via up link-path.

3.3 Data Processing

A. Sampling Data and Transmitting Packet

Different sensors on the AD node can be installed according to actual requirement. It is free to set data’s sampling frequency and interval. Data sampling task is set via a repeat timer. When sampling interval reaches, node samples data and constructs data frame then the system posts tasks in timer ISR. When task is executed, the system calls data transmitting function to transmit data to the next node via CC2420, at last, data reaches gateway and then is sent to control center via Internet, as shown in Fig. 5.

B. Receiving Packet

When wireless TX/RX module receives a packet, interrupt is triggered. In ISR, TaraxOS validates the received packet. The packet will be stored into RAM’s receiving buffer, and then the system posts a task if it is valid. While task is executed, the system calls frame-processing function of the route program. Frame-processing function examines type of the frame. If it is a data frame, it will be copied into transmitting buffer. Then the system modifies address of next-hop, preparing for transmitting the packet. If it is a protocol frame, it will be processed by local node. If it is a command frame, the system will call command processing function, as shown in Fig. 6.
4. Performance Analysis

The binary code’s capacity is only 97K bytes, which makes TaraxOS saves memory adequately and is exercisable in the instruction space of less than hundreds of bytes and data space of less than a few of ten bytes\[7\]. Next, we will discuss the system’s performance in the field of data processing and power consumption.

4.1 Node’s Data Processing

In sensor networks, three typical tasks of the node are: receiving route packet, forwarding data packet from other nodes, and acquiring local data and transmitting it out. The tasks number of the node depends on how the node processing data. If the node only transmits original data to sink directly, most of the tasks are communicating route tasks; if the node samples data and sends them to sink after processing, the local processing tasks will be much more. When tasks to be processed exceed node’s capacity, over-loading will occur. As for the former, if frequency of node’s transmitting data is higher or node’s density is larger, communication tasks will be superabundance, so over loading occurs. The latter, if there is a great amount of data to be processed or local tasks’ frequency is higher, over loading will occur too. Idiographic instances are as following.

1) While interrupt frequency is high, for example, we increase the sensor’s sampling frequency. CPU is busy with processing AD interrupt with the event PRI, and has no time left to process task in the queue, namely, the speed of system’s processing task is lower than that of posting task, which leads to queue keeping on increasing until no more task can be posted, as shown in Fig. 7.

2) The second instance, while there are some tasks needing long running time, network’s throughput will also decline. This is coordinate with the second characteristic of the FIFS task scheduling\[8\]. That means a task’s running time couldn’t be too long. As shown in Fig. 8, when tasks’ running time reaches 42 ms, packet’s transmitting rate declines to one packet per second. With task’s consumption time increasing, packet’s transmitting rate toboggans. Similar instances occur while receiving packet during experiments.

4.2 Node Power Consumption

The main power consumption\[9\] elements are composed of processing module, wireless TX/RX module, and sensor module. Wireless TX/RX module’s consumption takes the largest proportion\[10\] in Fig. 9.

The node’s average power consumption is about 110mW under normal operation state. Table 1 describes each part’s average power consumption under normal operation-state.
### 5. Conclusions

This paper describes TaraxOS suitable for wireless sensor networks. TaraxOS has the following characteristics: ingenious kernel scheduling module, two-level scheduling algorithm supporting event and task, higher event’s priority (PRI) used to process interrupt. Compared with tasks, event is preemptive. Tasks are not preemptive, executed by the order of FIFO, and are used in the centralized operation comparatively. Besides, our TaraxOS supports reduced and effective communication protocol, the mode of equity communication, broadcast communication among nodes, and asynchronous communication processing mode based on event.

### References


