Journal of Engineering and Applied Sciences 11 (6): 1168-1171, 2016
ISSN: 1816-949X

Real-Time Operating Systems for Wireless Modules

[1]Danil Aleksandrovich Yaroslavsky, [1]Dmitry Alekseevich Ivanov, [1]Marat Ferdinantovich Sadykov,
[1]Mikhail Petrovich Goryachev, [2]Oleg Gennadievich Savelyev and [3]Rustam Shaukatovich Misbakhov
[1]Federal State Budgetary Educational Institution of Higher Education,
"Kazan State Power Engineering University", 51 Krasnoselskaya St., Kazan, Russian Federation
[2]PJSC TATNEFT, 75, Lenin St., Almetyevsk, Russian Federation
[3]Federal State Budgetary Educational Institution of Higher Education, A.N. Tupolev
Kazan State Technical Research University KNRTU-KAI, 10, K.Marx St.,
420111, Kazan, Republic of Tatarstan, Russian Federation

Abstract: We develop Process Automation Wireless (PAW) modules to interact with the communication protocols Bluetooth, Wi-Fi, PLC, Ethernet, USB, RS-485, IrDA designed to monitor the status of objects, enterprises and public buildings automation through information collection and transmission from external sensors, automatic retransmission of data transmitted, etc. The study describes how to choose an operating system for the wireless single-chip microcontrollers of lowest price range.

Key words: Wireless interface module, line of modules, wireless network, automation system, a set of parameters, method of transmitting information, control protocol

INTRODUCTION

Each year, the share of automated systems in various industries in the world increases. Process automation can reduce power consumption, increase resource efficiency and productivity as well as significantly expand the functionality and comfort (Kopylov *et al*., 2015; Reshetnikov *et al*., 2015; Safin *et al*., 2015; Misbakhov *et al*., 2015a, b; Misbakhov and Moskalenko, 2015. Laptev *et al*., 2015; Gortyshov *et al*., 2009; Burganov *et al*., 2016). However, implementation of automated systems in some cases is limited by their high cost and low profitability as a result. In particular, the use of such modules as for example, ZigBee XBee, etc., in automated systems is limited due to their high cost. The high cost is caused by import origin of the modules, redundant hardware and software in most cases, i.e., there is an inefficient price from the point of view of customer's needs.

Experimental PAW module prototypes having a possibility to interact with the communication protocols Bluetooth, Wi-Fi, PLC, Ethernet, USB, RS-485, IrDA are designed for monitoring object statuses, automation of enterprises and public buildings by collecting and transmitting information from external sensors, automatic retransmission of data transmitted; creation of large area continuous information coverage with a broadcast access to all nodes on the network. For PAW modules there will be developed special software realizing the following functions:

- Local and remote object management
- Self-diagnostics of PAW modules and their communication channels
- Ability to customize by an operating personnel
- Network self-healing function upon loss of several components
- Network self-organization
- Ability to configure manually a network topology

MATERIALS AND METHODS

Methods of choosing a real-time operating system for wireless modules: For this purpose, let's consider which operating systems are suitable for single-chip wireless microcontrollers of the lowest price range. The following architectures ATmega128, CC 25xx, CC26xx, STM32W, ESP82xx were selected. Of many operating systems with open source, Contiki, FreeRTOS and TinyOS are best suited for our purposes. We excluded operating system TinyOS from the list since it has the only supported architecture that satisfies the above requirements, namely ATmega128 what can make it difficult to port to another architecture.

Corresponding Author: Danil Aleksandrovich Yaroslavsky, Federal State Budgetary Educational Institution of Higher Education, "Kazan State Power Engineering University", 51 Krasnoselskaya St., Kazan, Russian Federation

Operating system Contiki was designed for systems with limited memory. The source code for the operating system is written in C. The processes in Contiki are event driven. Reenterability extras were excluded by placing the process manager (scheduler) at the application level (Bourdonov *et al*., 2006). The control transfer process takes place when an event occurs addressed to the process, or in the case of a broadcast event when all processes alternately receive control. Parallel execution of several processes and the process preemption are not possible; the process runs until it returns control to the scheduler: multitasking in Contiki is cooperative. Processes can generate any events. All processes in this operating system are organized in a form of conventional programming language functions, thus the obligation to preserve process context is completely entrusted to the compiler. A finite-state-machine model of the program is organized through the use of special macros in processes.

Skeleton of a typical program in Contiki:

```
#include "contiki.h"
PROCESS(example_process, "Example process")
 AUTOSTART_PROCESSES(&example_process)
 PROCESS_THREAD(example_process, ev, data)
 {
  PROCESS_BEGIN()
  while(1) {
   PROCESS_WAIT_EVENT()
   printf("Got event number %d\n", ev)
      ...
  }

  PROCESS_END()
}
```

Here contiki.h file includes the necessary OS definitions. PROCESS macro creates a control structure of a process that will store its state. Macro AUTOSTART_PROCESSES put the process in startup procedure. Macro PROCESS_THREAD creates a function (process) which will be called by the scheduler.

Macros in a function organize state switching. This is possible due to the fact that the block "case" of the control structure "switch" in C language can be placed inside other control structures such as "if" and "while".

Macro PROCESS_BEGIN sets the selector:

char YIELD = 1; switch (state) {case 0:

Macro PROCESS_WAIT_EVENT accommodates the block "case":

YIELD = 0; state = __LINE__; case __LINE__: if ((!YIELD)) return PT_WAITING

Macro PROCESS_END closes a selector

Between macros PROCESS_BEGIN and PROCESS_END there also can be placed other interrupting (returning control to the operating system) macros

PROCESS_EXIT (); // End of the process

PROCESS_WAIT_EVENT_UNTIL (); // Waiting for events with a certain condition

PROCESS_WAIT_UNTIL (); // Wait for the condition

PROCESS_PAUSE (); // Temporary suspension of the process

RESULTS AND DISCUSSION

Adverse event of this macros is the inability to use the statement "switch" the body of which has the above macros. Multiple interfaces for working with a timer are implemented in the operating system: clock, timer, stimer, etimer, ctimer, rtimer. Interface "clock" allows obtaining the system time values in ticks (clock_time function) or seconds (clock_seconds) and also to make CPU blocking at a certain time (clock_wait, clock_delay).

The interface "timer" is based on the function clock_time of the interface "clock". Setting the timer is carried out by function timer_set. Check for excess of an interval can be done only manually by function timer_expired.

The interface "stimer" is similar to "timer" except for that the intervals are specified in seconds. The interface "etimer" is also based on clock_time function, difference from the above timers is that the timer sends PROCESS_EVENT_TIMER event after expiration of a preset time interval. The interface "ctimer" after expiration of a preset time interval calls the specified function "callback". The interface "rtimer" uses a hardware real time timer with high resolution. It is used in time-critical tasks. This timer is architecture-dependent and does not use real-time system clock. After expiration of a preset time interval a timer interrupt handler runs the specified function "callback".

The OS Contiki has a light-weighted TCP/IP stack called uIP and compatible with IPv4 and 6. There is also a simple proprietary protocol named Rime (Dunkels, 2016). To work with a radio module, IPv6 protocol is encapsulated by 6loWPAN protocol and IPv4 protocol can be encapsulated in Rime.

UDP core of IPv6 protocol server:

```
PROCESS_THREAD(udp_server_process, ev, data)
{
 PROCESS_BEGIN()
 server_conn = udp_new(NULL, UIP_HTONS(0), NULL)
 udp_bind(server_conn, UIP_HTONS(3000))
 while(1) {
  PROCESS_WAIT_EVENT()
  if(ev == tcpip_event) {
    memset(buf, 0, MAX_PAYLOAD_LEN)
    if(uip_newdata()) {
      len = uip_datalen()
      memcpy(buf, uip_appdata, len)
            uip_ipaddr_copy(&server_conn->ripaddr, &UIP_IP_BUF-
>srcipaddr)
      server_conn->rport = UIP_UDP_BUF->srcport
      uip_udp_packet_send(server_conn, buf, len)
      uip_create_unspecified(&server_conn->ripaddr)
      server_conn->rport = 0
   }
  }
 PROCESS_END()
}
```

Here function udp_new creates a connection descriptor. Function udp_bind opens a listening port. Function uip_newdata returns "true" if a message is received. Function uip_udp_packet_send sends the response to the client.

Operating system FreeRTOS is designed for portable devices with a "stringent" real time. The source code for the operating system is written in C. The OS kernel can be flexibly configured so, the operating system can have preemptive or cooperative multitasking. The use of a cooperative multitasking saves memory but imposes stringent timing requirements on the application code. Priorities are assigned to processes in the operating system, so the task with the highest priority in a queue will be executed first.

Data exchange between tasks is performed through queues, binary and counting semaphores. The core of the program (task) in the case of cooperative multitasking is as follows:

```
void main( void )
{
	// Initialize the necessary equipment
	.	.	.
	// Create a task
	xTaskCreate (vTask, "Task", configMINIMAL_STACK_SIZE,
NULL, tskIDLE_PRIORITY, NULL)
	 // Start the Task Scheduler
	vTaskStartScheduler ()
}

static void vTask( void *pvParameters )
{
	for( ;; )
	{
		if ( nothingToDo ) taskYIELD()
		// Do something that longs not more than a few system ticks
		.	.	.
	}
}
```

In fact, the only difference in the case of preemptive multitasking is that it is permissible not to call taskYIELD. There are two types of timers in the operating system FreeRTOS, the first one is program with low accuracy and the second one is high precision hardware being more resource-intensive. Both timers call the function specified when creating the timer after expiration of a specified time interval. In addition to the timers in the case of preemptive multitasking it is allowed to use delays implemented by function vTaskDelay. Program timers come in two types, a one-shot timer and timer with auto-reload. Working with a one-shot program timer is as follows:

```
staticvoidvTask( void *pvParameters )
{
	xOneShotTimer =xTimerCreate("Timer1",		/* Name for
debugging * /
		ONE_SHOT_TIMER_PERIOD, / * Response period* /
		pdFALSE, / * Without autoreload * /
		TIMER_ID, / * Timer identifier * /
		prvOneShotTimerCallback); / * Called function * /
	if (xOneShotTimer) xTimerStart (xOneShotTimer, DONT_BLOCK);
	for (;;)
	{
		// Do anything
		.	.	.
	}
}

static void prvOneShotTimerCallback (TimerHandle_t pxExpiredTimer)
{
	// Do anything
	if (Allrigth) xTimerReset (xOneShotTimer, DONT_BLOCK)
}
```

Hungarian notation of the source code adopted in FreeRTOS may be considered among the features of the operating system. Stack uIP is ported in FreeRTOS and work with it has no fundamental differences from work with Contiki. In new FreeRTOS versions stack uIP is replaced by a more "heavy" lwIP. In this operating system there are no radio module drivers for "small" crystals in particular for the controllers on the core 8051.

CONCLUSION

The most suitable operating system for this problem is Contiki. Its distribution kit includes all the necessary components. The system takes about a hundred bytes of RAM for its own needs with minimal configuration. For most of the studied modules exactly this operating system was used. The only exception were the modules based on ESP8266 crystal for which the manufacturer has successfully ported FreeRTOS.

ACKNOWLEDGEMENTS

Work on creation of software for process automation wireless modules is performed with the financial support of applied research and experimental developments (PNIER) project of the Ministry of Education and Science of the Russian Federation under the Agreement No. 14.577.21.0168 dd. 27 October 2015, the unique identifier PNIER RFMEFI57715X0168.

REFERENCES

Bourdonov, I.B., A.S. Kosachev and V.N. Ponomarenko, 2006. Real-time operating systems (Electronic resource). Preprint of the Institute for System Programming, Russian Academy of Sciences.

Burganov, R.A., R.S. Misbakhov, V.M. Gureev and L.R. Mukhametova, 2016. Methodological aspects of the driver of economic growth and energy. Int. Sci. Res. J., 72: 189-195.

Dunkels, A., 2016. Rime-A lightweight layered communication stack for sensor networks (Electronic resource). Proceedings of the European Conference on Wireless Sensor Networks, January 2007, The Netherlands.

Gortyshov, Y.F., V.M. Gureev, R.S. Misbakhov, I.F. Gumerov and A.P. Shaikin, 2009. Influence of fuel hydrogen additives on the characteristics of a gas-piston engine under changes of an ignition advance angle. Russian Aeronautics (Iz VUZ), 52: 488-490.

Kopylov, A.M., I.V. Ivshin, A.R. Safin, R.S. Miesbachov and R.R. Gibadullin, 2015. Assessment, calculation and choice of design data for reversible reciprocating electric machine. Int. J. Applied Eng. Res., 10: 31449-31462.

Laptev, A.G., R.S. Misbakhov and E.A. Lapteva, 2015. Numerical simulation of mass transfer in the liquid phase of the bubble layer of a thermal deaerator. Thermal Eng., 62: 911-915.

Misbakhov, R. and N. Moskalenko, 2015. Simulation of heat transfer and fluid dynamics processes in shell-and-pipe heat exchange devices with segmental and helix baffles in a casing. Biosci. Biotechnol. Res. Asia, 12: 563-569.

Misbakhov, R.S., V.M. Gureev, N.I. Moskalenko, A.M. Ermakov and I.Z. Bagautdinov, 2015. Simulation of surface intensification of heat exchange in shell-and-pipe and heat exchanging devices. Biosci. Biotechnol. Res. Asia, 12: 517-525.

Misbakhov, R.S., V.M. Gureev, N.I. Moskalenko, A.M. Ermakov, N.I. Moskalenko and I.Z. Bagautdinov, 2015. Numerical studies into hydrodynamics and heat exchange in heat exchangers using helical square and oval tubes. Biosci. Biotechnol. Res. Asia, 12: 719-724.

Reshetnikov, A.P., I.V. Ivshin, N.V. Denisova, A.R. Safin, R.S. Misbakhov and A.M. Kopylov, 2015. Optimization of reciprocating linear generator parameters. Int. J. Applied Eng. Res., 10: 31403-31414.

Safin, A.R., I.V. Ivshin, A.M. Kopylov, R. Misbakhov and A.N. Tsvetkov, 2015. Selection and justification of design parameters for reversible reciprocating electric machine. Int. J. Applied Eng. Res., 10: 31427-31440.