**电子科技大学**



**嵌入式系统实验**

**实验二**

**学号：202222080625**

**姓名：乔翱**

**学院：计算机科学与工程学院**

1. **实验名称**

事件管理和通信的学习

1. **实验时间**

2023年5月16日

**三、实验内容和目的**

1、学习事件管理，通过事件实现任务的同步。

2、学习任务之间的通信，学习如何在任务之间接收信息和传递信息。

3、完成实验二的要求。

**四、实验设计**

实验要求总共有14个任务，每个任务有独一无二的优先级，其中七个任务通过事件机制进行同步，七个任务之间相互通信。

前七个任务，也就是task1到task7都会等待event，只有等到相应的event才能执行后面的功能，否则阻塞。并且这七个任务都为之后的任务设置了其需要的event。

后七个任务也就是任务6到任务14接收MSG1到MSG7，任务7到任务13发送MSG1到MSG7，完成通信要求。

**五、任务配置**

按照实验设计对总共14个任务进行配置，项目中任务的相关配置如下。需要注意的是，使用事件相关服务，任务需要配置成是扩展任务。

|  |
| --- |
| /\* task configurations \*/  const T\_OSEK\_TASK\_ConfigTable\_Struct osekConfig\_TaskTable[OCC\_NTSKS]=  {  /\*Task1 ID:0 priority:CONFIG\_OSEK\_TASK\_PRIORITY\_NUMBER-0-1 \*/  {  (T\_OSEK\_TASK\_Entry)FuncTask1, //task function  Task1, //task IDs  //task initial attributes  0 | OSEK\_TASK\_ACTIVE | OSEK\_TASK\_EXTENDED | OSEK\_TASK\_NONPREEMPT,  &taskStack[Task1][TASK\_STACK\_SIZE-1], //top of task stack  &taskStack[Task1][0] //bottom of task stack    },  /\*Task2 ID:1 priority:CONFIG\_OSEK\_TASK\_PRIORITY\_NUMBER-1-1 \*/  {  (T\_OSEK\_TASK\_Entry)FuncTask2,  Task2,  0 | OSEK\_TASK\_EXTENDED | OSEK\_TASK\_NONPREEMPT | OSEK\_TASK\_ACTIVE,  &taskStack[Task2][TASK\_STACK\_SIZE-1],  &taskStack[Task2][0]  },  /\*Task3 ID:2 priority:CONFIG\_OSEK\_TASK\_PRIORITY\_NUMBER-2-1 \*/  {  (T\_OSEK\_TASK\_Entry)FuncTask3,  Task3,  0 | OSEK\_TASK\_EXTENDED | OSEK\_TASK\_NONPREEMPT | OSEK\_TASK\_ACTIVE,  &taskStack[Task3][TASK\_STACK\_SIZE-1],  &taskStack[Task3][0]  },  /\*Task4 ID:3 priority:CONFIG\_OSEK\_TASK\_PRIORITY\_NUMBER-3-1 \*/  {  (T\_OSEK\_TASK\_Entry)FuncTask4,  Task4,  0 | OSEK\_TASK\_EXTENDED | OSEK\_TASK\_NONPREEMPT | OSEK\_TASK\_ACTIVE,  &taskStack[Task4][TASK\_STACK\_SIZE-1],  &taskStack[Task4][0]  },  /\*Task5 ID:4 priority:CONFIG\_OSEK\_TASK\_PRIORITY\_NUMBER-4-1 \*/  {  (T\_OSEK\_TASK\_Entry)FuncTask5,  Task5,  0 | OSEK\_TASK\_EXTENDED | OSEK\_TASK\_NONPREEMPT | OSEK\_TASK\_ACTIVE,  &taskStack[Task5][TASK\_STACK\_SIZE-1],  &taskStack[Task5][0]  },  /\*Task6 ID:5 priority:CONFIG\_OSEK\_TASK\_PRIORITY\_NUMBER-5-1 \*/  {  (T\_OSEK\_TASK\_Entry)FuncTask6,  Task6,  0 | OSEK\_TASK\_EXTENDED | OSEK\_TASK\_NONPREEMPT | OSEK\_TASK\_ACTIVE,  &taskStack[Task6][TASK\_STACK\_SIZE-1],  &taskStack[Task6][0]  },  /\*Task7 ID:6 priority:CONFIG\_OSEK\_TASK\_PRIORITY\_NUMBER-6-1 \*/  {  (T\_OSEK\_TASK\_Entry)FuncTask7,  Task7,  0 | OSEK\_TASK\_EXTENDED | OSEK\_TASK\_NONPREEMPT | OSEK\_TASK\_ACTIVE,  &taskStack[Task7][TASK\_STACK\_SIZE-1],  &taskStack[Task7][0]  },  /\*Task8 ID:7 priority:CONFIG\_OSEK\_TASK\_PRIORITY\_NUMBER-7-1 \*/  {  (T\_OSEK\_TASK\_Entry)FuncTask8,  Task8,  0 | OSEK\_TASK\_EXTENDED | OSEK\_TASK\_NONPREEMPT | OSEK\_TASK\_ACTIVE,  &taskStack[Task8][TASK\_STACK\_SIZE-1],  &taskStack[Task8][0]  },  /\*Task9 ID:8 priority:CONFIG\_OSEK\_TASK\_PRIORITY\_NUMBER-8-1 \*/  {  (T\_OSEK\_TASK\_Entry)FuncTask9,  Task9,  0 | OSEK\_TASK\_EXTENDED | OSEK\_TASK\_NONPREEMPT | OSEK\_TASK\_ACTIVE,  &taskStack[Task9][TASK\_STACK\_SIZE-1],  &taskStack[Task9][0]  },  /\*Task10 ID:9 priority:CONFIG\_OSEK\_TASK\_PRIORITY\_NUMBER-9-1 \*/  {  (T\_OSEK\_TASK\_Entry)FuncTask10,  Task10,  0 | OSEK\_TASK\_EXTENDED | OSEK\_TASK\_NONPREEMPT | OSEK\_TASK\_ACTIVE,  &taskStack[Task10][TASK\_STACK\_SIZE-1],  &taskStack[Task10][0]  },  /\*Task11 ID:10 priority:CONFIG\_OSEK\_TASK\_PRIORITY\_NUMBER-10-1 \*/  {  (T\_OSEK\_TASK\_Entry)FuncTask11,  Task11,  0 | OSEK\_TASK\_EXTENDED | OSEK\_TASK\_NONPREEMPT | OSEK\_TASK\_ACTIVE,  &taskStack[Task11][TASK\_STACK\_SIZE-1],  &taskStack[Task11][0]  },  /\*Task12 ID:11 priority:CONFIG\_OSEK\_TASK\_PRIORITY\_NUMBER-11-1 \*/  {  (T\_OSEK\_TASK\_Entry)FuncTask12,  Task12,  0 | OSEK\_TASK\_EXTENDED | OSEK\_TASK\_NONPREEMPT | OSEK\_TASK\_ACTIVE,  &taskStack[Task12][TASK\_STACK\_SIZE-1],  &taskStack[Task12][0]  },  /\*Task13 ID:12 priority:CONFIG\_OSEK\_TASK\_PRIORITY\_NUMBER-12-1 \*/  {  (T\_OSEK\_TASK\_Entry)FuncTask13,  Task13,  0 | OSEK\_TASK\_EXTENDED | OSEK\_TASK\_NONPREEMPT | OSEK\_TASK\_ACTIVE,  &taskStack[Task13][TASK\_STACK\_SIZE-1],  &taskStack[Task13][0]  },  /\*Task14 ID:13 priority:CONFIG\_OSEK\_TASK\_PRIORITY\_NUMBER-6-1 \*/  {  (T\_OSEK\_TASK\_Entry)FuncTask14,  Task14,  0 | OSEK\_TASK\_EXTENDED | OSEK\_TASK\_NONPREEMPT | OSEK\_TASK\_ACTIVE,  &taskStack[Task14][TASK\_STACK\_SIZE-1],  &taskStack[Task14][0]  },  /\*Task Idle \*/  {  0,  OSEK\_TASK\_IDLE\_ID,  OSEK\_TASK\_ACTIVE,  },  }; |

**六、实验步骤**

1、修改osprop.h中关于任务数以及消息数的配置。

2、在cfg.h和cfh.c配置相关任务，包括任务的定义声明以及任务相关属性的配置，重点配置任务的初始状态。并且配置message，七个message的配置如下。

|  |
| --- |
| T\_OSEK\_MESSAGE\_ConfigTable\_Struct osekConfig\_MsgTable[OCC\_NMSGS] =  {  {  0, //ID  &messageBuf[0], //message butter  sizeof(messageBuf[0]), //size of the message buffer  &messageQueue[0], //message queue  MESSAGE\_BUF\_SIZE //size of the queue  },  {  1, //ID  &messageBuf[1],  sizeof(messageBuf[1]),  &messageQueue[1],  MESSAGE\_BUF\_SIZE  },  {  2, //ID  &messageBuf[2],  sizeof(messageBuf[2]),  &messageQueue[2],  MESSAGE\_BUF\_SIZE  },  {  3, //ID  &messageBuf[3],  sizeof(messageBuf[3]),  &messageQueue[3],  MESSAGE\_BUF\_SIZE  },  {  4, //ID  &messageBuf[4],  sizeof(messageBuf[4]),  &messageQueue[4],  MESSAGE\_BUF\_SIZE  },  {  5, //ID  &messageBuf[5],  sizeof(messageBuf[5]),  &messageQueue[5],  MESSAGE\_BUF\_SIZE  },  {  6, //ID  &messageBuf[6],  sizeof(messageBuf[6]),  &messageQueue[6],  MESSAGE\_BUF\_SIZE  }  }; |

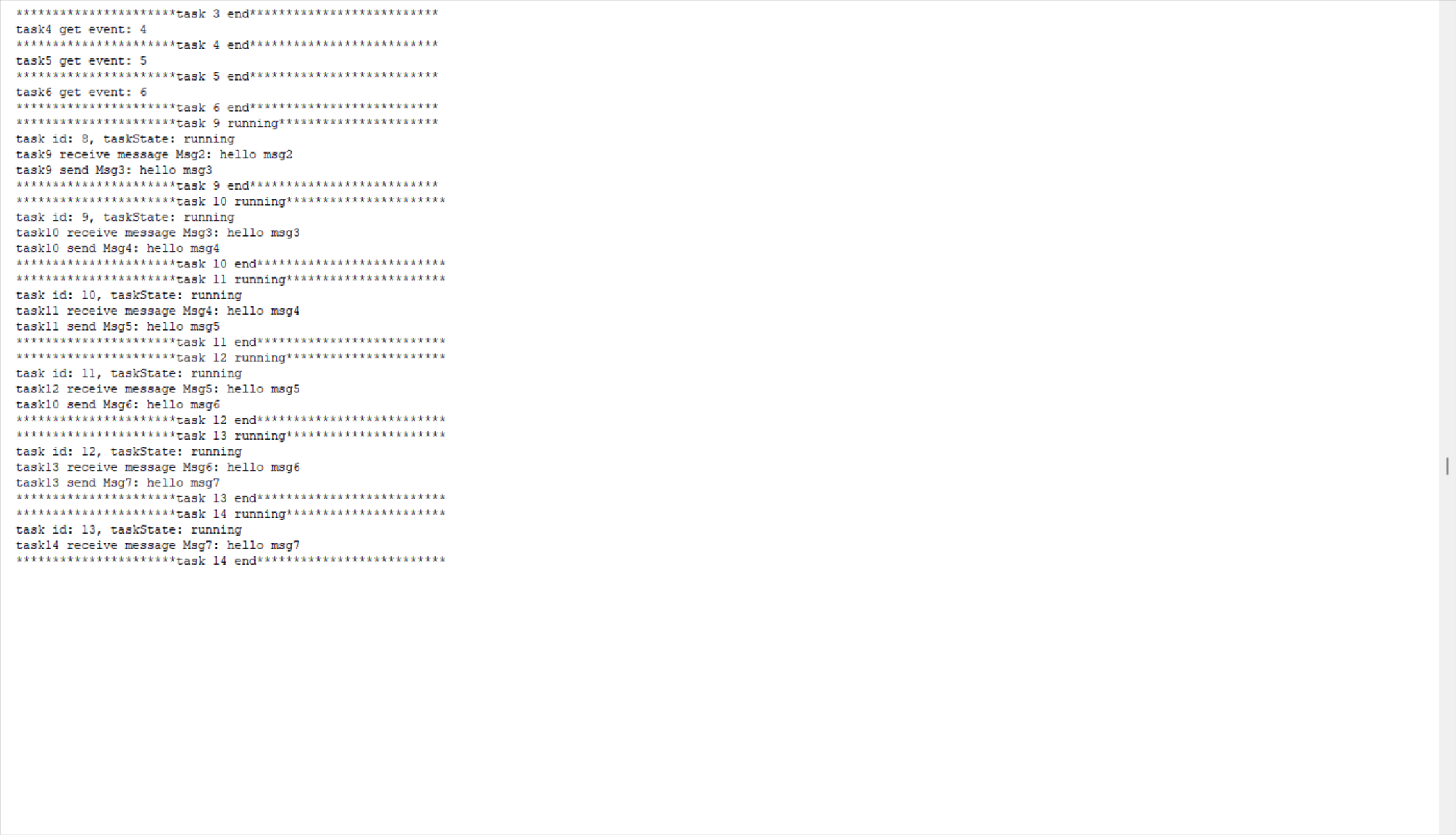
3、在init.c中编写每个任务函数，实现每个任务的相关功能。注意通信需要确保启动了StartCom服务，所以在任务一种启动了StartCom服务。下面是任务7和任务8的函数，可以体现出事件管理以及通信主要的调用。

|  |
| --- |
| TASK(Task7)  {  TaskType taskId;  EventMaskType mask;  OSBYTE buf1[] = "hello msg1";  GetTaskID(&taskId);    printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*task 7 running\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");  showTaskInfo(taskId);    //task7 waits for event 0x07  printf("task7 wait event: 0x07\n");  WaitEvent(0x07);  //get event maskcode  GetEvent(taskId, &mask);  printf("task7 get event: %d\n", (int)mask);    //set event 0x03 for task3  printf("task7 set event 0x03 for task3\n");  SetEvent(Task3, 0x03);  //set event 0x04 for task4  printf("task7 set event 0x04 for task4\n");  SetEvent(Task4, 0x04);  //set event 0x05 for task5  printf("task7 set event 0x05 for task4\n");  SetEvent(Task5, 0x05);  //set event 0x06 for task6  printf("task7 set event 0x06 for task4\n");  SetEvent(Task6, 0x06);  //task7 send message Msg1  printf("task7 send Msg1: %s\n", buf1);  SendMessage(Msg1, buf1);    printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*task 7 end\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");  TerminateTask();  }  TASK(Task8)  {  TaskType taskId;  StatusType status;  OSBYTE buf[20];  OSBYTE buf1[] = "hello msg2";  EventMaskType mask;    GetTaskID(&taskId);    printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*task 8 running\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");    //show task information  showTaskInfo(taskId);    //set event 0x01 for task1  printf("task7 set event 0x01 for task1\n");  SetEvent(Task1, 0x01);  //task8 tries to receive message Msg1  status = ReceiveMessage(Msg1, buf);  if(status == E\_OK)  {  printf("task8 receive message Msg1: %s\n", buf);  } else  {  printf("task8 receive message error: %d\n", status);  }    //task8 send message Msg2  printf("task8 send Msg2: %s\n", buf1);  SendMessage(Msg2, buf1);    printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*task 8 end\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");  TerminateTask();    } |

**七、实验结果及分析**

在IDE中编译，并运行，实验结果如下：





分析实验结果，可以看到实验正常按照所设计的进行，任务一到七执行后由于需要等待相应的事件都被阻塞，只有在后续任务设置了相应的事件，任务一到七得到相应的任务后才继续运行。

任务八到任务十四都各自接收了消息并输出，由于每个任务接收的消息都是前一个任务发送的消息，所以每个任务都正常接收到了消息并且成功输出了消息。

可以看到，整个实验，完成了实验要求，七个任务通过事件机制进行了同步，七个任务之间进行了通信。

**八、实验结论及心得体会**

通过本次实验，学习了事件管理机制的使用，学习了如何使用事件管理机制，实现任务的同步。并且学习了通信机制，在任务之间如何进行通信，如何发送消息和接收消息。