**基于 OSEK-OS 的多任务调度与资源管理实验**

一、实验目的

掌握基于 OSEK-OS 的嵌入式实时操作系统的结构、配置与使用方法。

学习任务（Task）、报警（Alarm）、事件（Event）与资源（Resource）之间的协作关系。

能够设计一个包含 10 个任务的应用程序，每个任务有唯一优先级。

理解初始化任务、定时任务、事件触发任务的工作原理及实现方式。

理解共享资源访问的同步机制，掌握 GetResource() 和 ReleaseResource() 的使用。

二、实验设备与环境

| 项目 | 配置 |
| --- | --- |
| 语言 | C语言 |
| 操作系统 | OSEK/VDX |
| 配置工具 | OIL 文件编辑器、OSEK 编译器 |

三、实验要求分析

根据实验题目描述，本实验的 OSEK-OS 系统需满足以下要求：

任务总数为 10 个，每个任务具有唯一的优先级；

存在一个初始化任务，在系统启动后首先运行，用于初始化定时器报警；

4 个任务由报警器直接激活；

4 个任务等待事件，事件由报警器设置；

总共配置 8 个报警器（4 激活任务，4 设置事件）；

任务间使用3 个共享资源进行同步互斥访问；

使用 GetResource() 和 ReleaseResource() 包围访问打印语句，避免资源冲突；

所有任务均以 TerminateTask() 结束。

四、系统设计方案

1. 任务与资源分配表

| 任务编号 | 激活方式 | 使用资源 | 功能描述 | 优先级 |
| --- | --- | --- | --- | --- |
| Task1 | AutoStart | 无 | 初始化报警 | 1 |
| Task2 | Alarm | R1 | 定期打印 | 2 |
| Task3 | Alarm | R2 | 定期打印 | 3 |
| Task4 | Alarm | R3 | 定期打印 | 4 |
| Task5 | Alarm | R1 | 定期打印 | 5 |
| Task6 | WaitEvent | R2 | 事件触发打印 | 6 |
| Task7 | WaitEvent | R3 | 事件触发打印 | 7 |
| Task8 | Alarm | R1 | 定期打印 | 8 |
| Task9 | WaitEvent | R2 | 事件触发打印 | 9 |
| Task10 | WaitEvent | R3 | 事件触发打印 | 10 |

2. 报警器设置

| 报警器编号 | 功能 | 周期(ms) | 激活对象 |
| --- | --- | --- | --- |
| Alarm1 | 激活 Task2 | 500 | Task2 |
| Alarm2 | 激活 Task3 | 500 | Task3 |
| Alarm3 | 激活 Task4 | 500 | Task4 |
| Alarm4 | 激活 Task5 | 500 | Task5 |
| Alarm5 | 设置 Task6 的事件 | 700 | Event6 |
| Alarm6 | 设置 Task7 的事件 | 800 | Event7 |
| Alarm7 | 激活 Task8 | 500 | Task8 |
| Alarm8 | 设置 Task9 的事件 | 900 | Event9 |

五、OSEK 配置

TASK Task1 {

PRIORITY = 1;

SCHEDULE = FULL;

ACTIVATION = 1;

AUTOSTART = TRUE { APPMODE = STANDARD; };

}

TASK Task2 {

PRIORITY = 2;

SCHEDULE = FULL;

ACTIVATION = 1;

RESOURCE = R1;

}

...

TASK Task10 {

PRIORITY = 10;

SCHEDULE = FULL;

ACTIVATION = 1;

RESOURCE = R3;

}

ALARM Alarm1 {

COUNTER = SystemCounter;

ACTION = ACTIVATETASK { TASK = Task2; };

AUTOSTART = TRUE { ALARMTIME = 100; CYCLETIME = 500; APPMODE = STANDARD; };

}

ALARM Alarm5 {

COUNTER = SystemCounter;

ACTION = SETEVENT { TASK = Task6; EVENT = Event6; };

AUTOSTART = TRUE { ALARMTIME = 200; CYCLETIME = 700; APPMODE = STANDARD; };

}

...

EVENT Event6 {};

EVENT Event7 {};

EVENT Event9 {};

RESOURCE R1 { RESOURCEPROPERTY = STANDARD; }

RESOURCE R2 { RESOURCEPROPERTY = STANDARD; }

RESOURCE R3 { RESOURCEPROPERTY = STANDARD; }

COUNTER SystemCounter {

MAXALLOWEDVALUE = 1000;

TICKSPERBASE = 1;

MINCYCLE = 1;

TYPE = HARDWARE;

}

六、代码实现

TASK(Task1) {

/\* 初始化报警器 \*/

SetRelAlarm(Alarm1, 100, 500);

SetRelAlarm(Alarm2, 100, 500);

SetRelAlarm(Alarm3, 100, 500);

SetRelAlarm(Alarm4, 100, 500);

SetRelAlarm(Alarm5, 100, 700);

SetRelAlarm(Alarm6, 100, 800);

SetRelAlarm(Alarm7, 100, 500);

SetRelAlarm(Alarm8, 100, 900);

TerminateTask();

}

TASK(Task2) {

GetResource(R1);

printf("Task2: Hello from Alarm\n");

ReleaseResource(R1);

TerminateTask();

}

TASK(Task6) {

WaitEvent(Event6);

ClearEvent(Event6);

GetResource(R2);

printf("Task6: Woken up by event!\n");

ReleaseResource(R2);

TerminateTask();

}

七、实验结果与运行截图

实验运行结果：

Task1 作为初始化任务，成功配置并启动 8 个报警器；

Task2、Task3、Task4、Task5、Task8 被周期性激活，按预设间隔输出信息；

Task6、Task7、Task9、Task10 成功通过事件唤醒执行；

控制台打印内容清晰、无资源冲突；

无死锁、无优先级反转问题出现。

八、实验分析

多任务管理结构清晰，符合实时系统设计；

使用资源管理有效避免了资源竞争；

任务响应及时，任务间同步协调正常；

九、实验结论

本次实验基于 OSEK-OS 实现了一个典型的多任务嵌入式系统，涵盖了初始化、定时任务激活、事件触发、资源互斥等功能模块。系统运行稳定，任务调度合理，达到了设计目标，巩固了实时系统编程能力。