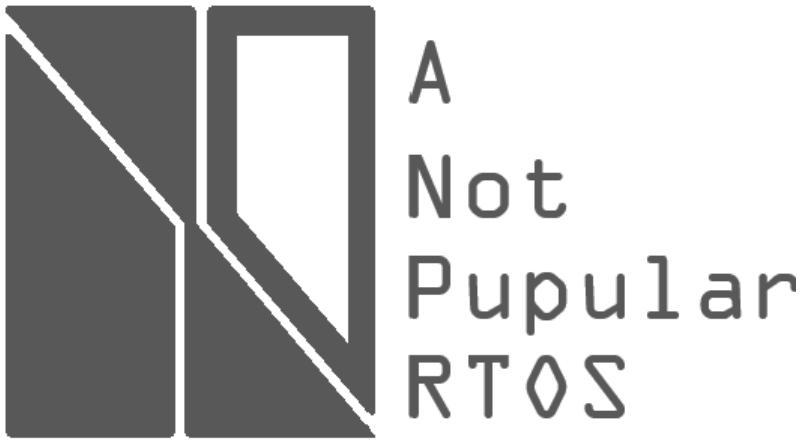

简介 Introduction

为什么有?

NpOS以运行在RISC-V架构芯片上的完整的嵌入式实时操作系统为目标。目前可以进行基于抢占优先级的调度，实现多任务。课余时间的小实践，之后有精力了会尝试将其移植到arm平台上。

这个rtos更多的是我个人对所学的实践，1.0.0版本是一个只有任务调度的非常微小迷你的内核（甚至还不能称为一个操作系统，只是一个调度内核），可以在release中或者tag中找到，也适合刚学完操作系统或者单片机，想要练手的朋友来阅读。npos使用的是GPL协议，欢迎大家在其上进行肆意猖狂地改动。

有什么用?

目前Npos能做到的:

- 创建最多64个优先级（1.5版本之后才增加到64，1.5之前的版本只有8个优先级）
- 基于优先级的抢占式调度器
- 慢慢添加的组件与内核功能...
    - message

配置 Config

所有跟给用户配置的系统相关的配置都在NpOS_config_user.h中，修改其中的宏定义即可对Npos进行裁剪修改

系统 System

系统心跳 System Tick

\#define NPOS_SchedulingInterval_MS 5 修改这个宏，即可改变系统心跳频率

> 系统心跳间隔表示调度系统多久进行一次任务调度，同时它也是时间片的最小单位，后面使用的系统级延时函数，延时的时间为系统心跳间隔的整数倍

系统信息上报 Log

`\#define NPOS_logInfoPrintf_EN  0` 0为不启用，1为启用

启用后会向串口打印一些系统运行信息

任务 Task

调度器设置 Scheduler Setting

Npos默认使用的是基于优先级的抢占调度机制，可以通过修改下列宏附加新的调度机制

`\#define NPOS_TASK_TIMESLICE_SCHEDUL_EN 0` 1为启用同任务优先级的时间片轮转调度，0为不启用

任务运行时间 task runTime

`#define NPOS_TASK_USAGERATE_EN 0` 1为启用任务运行时长的记录，0为不启用。单个任务的运行时间被保存在任务的tcb中

cpu利用率统计 cpu usage rate

`\#define NPOS_TASK_CPUUSAGE_RATE_EN 0` 1为启用，0为不启用.

该宏会在idle task中打印输出cpu的累计利用率.

> 使用*cpu*利用率统计的前提是 任务运行时间 *task runTime* 功能被启用，即*`#define NPOS_TASK_USAGERATE_EN 1`*
>
> 输出的利用率是累计利用率，而不是平均时刻利用率!

任务优先级数量 task priority number

```
// 允许的最多优先级数量
#define NPOS_TASK_PRIORITY_NUMBER_8          8
#define NPOS_TASK_PRIORITY_NUMBER_16         16
#define NPOS_TASK_PRIORITY_NUMBER_32         32
#define NPOS_TASK_PRIORITY_NUMBER_64         64
#define NPOS_TASK_PRIORITY_NUMBER_128        128
#define NPOS_TASK_PRIORITY_NUMBER_256        256

//在这里填写对应数量级的宏
#define NPOS_TASK_PRIORITY_NUMBER    NPOS_TASK_PRIORITY_NUMBER_8
```

修改`#define NPOS_TASK_PRIORITY_NUMBER  NPOS_TASK_PRIORITY_NUMBER_8`后面的`NPOS_TASK_PRIORITY_NUMBER_8`即可改变系统可以支持的优先级数

> 不论如何，系统都保留一个最低优先级，不开放给用户使用！（其实你非要用也没差，只是那个优先级我拿来放空闲任务了而已）

对象 object

通过修改以下宏的值，来进行内核功能的裁剪。 1为使用 0为不使用

```
/*            Object                */
//是否使用消息功能
#define NPOS_OBJ_MESSAGE_EN          0
```

API说明 API reference

这里面列出来的函数是可以供用户调用的，没有列出来的就是不建议单独使用的（比如有的人跑到头文件里去找函数声明...）

系统 SYSTEM

进入系统

```
void NpOS_Start();
```

作用：调用此函数后*cpu*的使用权与控制器正式交给*os*，进入系统

参数：void

返回值：void

任务 TASK

相关定义 Related definitions

- `Np_TCB` 任务控制块结构体，用来存放任务相关的信息。
- `TASK_STACK_TYPE` 这是一个宏定义，当用户要申请一块堆栈给任务使用时，请务必使用这个宏定义作为数组类型。
- `task_funcsta` 函数执行情况，是一个枚举类型
  - Exc_ERROR
  - Exc_OK

相关函数 Related functions

初始化任务链表

```
void NpOS_task_tcblistInit();
```

作用：初始化任务相关链表

作用区间：未进入系统之前，即还未调用`NpOS_Start()`之前

参数：void

返回值：void

创建任务

```
task_funcsta NpOS_task_createTask(Np_TCB* tcb,
                                  p_taskFunction taskfunc,
                                  TASK_PRIORITY_TYPE taskpri,
                                  void* stackbut,
                                  uint32_t stacksize,
                                  task_status taskstatus);
```

作用：用以创建一个任务

作用区间：任意时刻

参数：

1. *\param[in**] Np_TCB** tcb* 任务所属的任务控制块的指针*
2. *\param[in**] p_taskFunction taskfunc* 任务的函数入口指针
3. *\param[in**] TASK_PRIORITY_TYPE taskpri* 任务的优先级
4. *\param[in**] void** stackbut* 任务的堆栈的栈底指针（即申请的数组的头指针）*
5. *\param[in**] uint32_t stacksize* 任务的堆栈大小（单位 *Byte*）
6. *\param[in**] task_status taskstatus* 任务创建完后的初始状态
    1. 取值： `TASK_PEND`,`TASK_WAIT`,`TASK_READY`,`TASK_UNKNOWN`

返回结果：

1. *\retval task_funcsta* 返回任务的执行情况

使用范例：

```
#define task_StackSize 512

TASK_STACK_TYPE task_Stack[task_StackSize];

```

```
Np_TCB task_Tcb;

void task(){
    while(1){
    }
}

int main(void)
{
    usart0_config();
    eclic_global_interrupt_enable();
    eclic_set_nlbits(ECLIC_GROUP_LEVEL3_PRIO1);
    NpOS_task_tcblistInit();

    NpOS_task_createTask(
        &task_Tcb,
        task,
        7,
        task_Stack,
        task_StackSize,
        TASK_READY
        );

    NpOS_Start();

    while (1){
    }
}
```

> 如果任务创建失败，请在*config*文件中打开系统*log*功能，并通过串口查看打印信息排查错误

删除任务

```
task_funcsta NpOS_task_deleteTask(Np_TCB* tcb);
```

作用：删除一个任务

作用区间：当进入系统后，即调用了`NpOS_Start()`之后

参数：

1. *param[in**] Np_TCB** tcb 任务所属的任务控制块的指针*

返回值：

1. *retval* *task_funcsta 返回函数的执行情况

使用范例：

```
void task(){
    while(1){
        NpOS_task_deleteTask(p_anotherTask_TCB);
    }
}
```

> 使用这个函数会主动触发一次任务调度

挂起任务

```
task_funcsta NpOS_task_pendTask(Np_TCB* tcb);
```

作用：挂起一个任务，使其在被重新就绪之前都无法主动获取cpu的使用权限

作用区间：当进入系统后，即调用了`NpOS_Start()`之后

参数：

1. *param[in**] Np_TCB** tcb 任务所属的任务控制块的指针*

返回值：

1. *retval task_funcsta* 返回任务的执行情况

使用范例：

```
void task(){
    while(1){
        NpOS_task_pendTask(p_anotherTask_TCB);
    }
}
```

> 如果被挂起的任务正在*pend*延时中，会打断它的延时，并且这个任务被重新就绪时，会刷新它的*delayTicks*为*0*
>
> 使用这个函数会主动触发一次任务调度

就绪任务

```
task_funcsta NpOS_task_readyTask(Np_TCB* tcb);
```

作用：使一个任务从其它状态变为就绪态

作用区间：当进入系统后，即调用了`NpOS_Start()`之后

参数：

1. \param[in**] Np_TCB** tcb 任务所属的任务控制块的指针*

返回值：

1. \retval task_funcsta 返回任务的执行情况

使用范例：

```
void task(){
    while(1){
        NpOS_task_readyTask(p_anotherTask_TCB);
    }
}
```

> 使用这个函数会主动触发一次任务调度

任务非阻塞延时

```
void NpOS_task_pendDelayTicks(uint32_t ticks);
```

作用：将当前任务挂起，非阻塞 延时ticks个系统tick

作用区间：当进入系统后，即调用了`NpOS_Start()`之后

参数：

1. \param[in**] uint32_t ticks 任务需要等待的tick数 每个ticks的持续时间由 Npos_config.h中的 NPOS_SchedulingInterval_MS 决定
    1. 例如系统NPOS_SchedulingInterval_MS 为 5 ，那么 NpOS_task_pendDelayTicks（200)会让当前任务延时1秒

返回值：void

使用范例：

```
void task(){
    while(1){
        //延时1秒
        NpOS_task_pendDelayTicks(1000/NPOS_SchedulingInterval_MS );
    }
}
```

> 使用这个函数会主动触发一次任务调度

消息 MESSAGE

相关定义 Related definitions

- `Np_MSG` 描述消息的结构体

相关函数 Related functions

创建（初始化）一个消息体

```
task_funcsta NpOS_obj_createMsg(Np_MSG* msg);
```

作用：实际上是初始化一个`Np_MSG`结构体

作用区间：创建可以在任何时候创建，但发送与接收消息必须在进入操作系统环境之后使用

参数：

1. *\param[in] Np_MSG** msg 要初始化的消息体*

返回值：

1. *\retval task_funcsta* 返回任务的执行情况

使用范例：

```
Np_MSG messageExample;

void task1(){
    uint8_t character;
    while(1){
        character = 'A';
        NpOS_obj_createMsg(&messageExample);

 NpOS_obj_sendMsgtoTask(&messageExample,&anotherTaskTcb,&character);
    }
}
```

> 消息结构体需要在参与通信的任务都可以访问到的地方声明（比如直接声明为全局变量）

发送消息至 指定 任务

```
task_funcsta NpOS_obj_sendMsgtoTask(Np_MSG* msg,
                                    Np_TCB* receivertcb,
                                    void* p_msg);
```

作用：发送消息到指定的一个任务。

作用区间：当进入系统环境后，即调用了`NpOS_Start()`之后

参数：

1. *param[in**] Np_MSG** msg 承载消息的结构体*
2. *param[in**] Np_TCB** receivertcb消息的接收者，用以核验消息的收发正确性*
3. *param[in**] void** p_msg 消息所携带的信息的指针*

返回值：

1. *retval task_funcsta 返回任务的执行情况

使用范例：

```
Np_MSG messageExample;

void task1(){
    uint8_t character;
    while(1){
        character = 'A';
        NpOS_obj_createMsg(&messageExample);

 NpOS_obj_sendMsgtoTask(&messageExample,&anotherTaskTcb,&character);
    }
}
```

> 使用这个函数会主动触发一次任务调度

等待接收一个来自指定任务的消息

```
task_funcsta NpOS_obj_receiveMsgFromTask(Np_MSG* msg,
                                         Np_TCB* sendertcb);
```

作用：等待接收一个来自指定任务的消息

作用区间：当进入系统环境后，即调用了`NpOS_Start()`之后

参数：

1. *param[in**] Np_MSG** msg 承载消息的结构体*
2. *param[in**] Np_TCB** sendertcb 消息的发送者，用以核验消息的收发正确性*

返回值：

1. *retval task_funcsta 返回任务的执行情况

使用范例：

```
Np_MSG messageExample;

```

```
void task1(){
    uint8_t character;
    while(1){
        character = 'A';
        NpOS_obj_createMsg(&messageExample);
        NpOS_obj_sendMsgtoTask(&messageExample,&task2_tcb,&character);
    }
}


void task2(){
    while(1){
        NpOS_obj_receiveMsgFromTask(&messageExample,&task1_tcb);
        printf("you receive a character : %s \n",*(uint8_t)
(messageExample.p_message));
    }
}
```

> 使用这个函数会让当前任务持续等待消息的到来，如果没有等来消息，任务会暂时进入*wait*态，直到消息到来。