实验11~12：FreeRTOS 与 生产者-消费者模型 实验

11610101 韦青茂

实验器材

- 硬件：ARM-STM32开发板，ST-Link。
- 软件：Win10, CubeMX, PlatformIO via VSCode

实验要求

1. Finish the practice of the last lab: Using counting semaphore to solve the producer-consumer problem
2. Using mail queues to solve the producer-consumer problem
3. The buffer size of the producer-consumer problem is 4

实验过程

1.使用`Mail Queue`解决生产者-消费者问题

配置

1. `Producer`优先级为`Normal`，`Consumer`优先级为`BelowNormal`.
2. `MailQueue`大小为4:

```
osMailQDef(mail01, 4, mailStruct);
mail01Handle = osMailCreate(osMailQ(mail01), NULL);
```

代码

1. Producer部分:

```
void MsgProducerTask(void const *argument)
{
/* USER CODE BEGIN MsgProducerTask */
mailStruct *mail;
u_int8_t i = 0;
/* Infinite loop */
for (;; ++i)
{
    while (!(mail = (mailStruct *)osMailAlloc(mail01Handle, osWaitForever))
    {
    // 如果mail queue已满, 则此时mail为空指针
    // 等待500ms, 此时系统调度会切换线程
    printf("[P]Full! Wait.\n");
    osDelay(500);
    }
    // 向mail queue中发送一则消息
    mail->var = i;
    printf("[P]>>%d\n", mail->var);
    osMailPut(mail01Handle, mail);
}
/* USER CODE END MsgProducerTask */
}
```

2. Consumer部分:

```
void MsgConsumerTask(void const *argument)
{
/* USER CODE BEGIN MsgConsumerTask */
osEvent event;
mailStruct *pMail;

/* Infinite loop */
for (;;)
{
    // 消费者每隔100ms尝试获取一次消息
    osDelay(100);
    event = osMailGet(mail01Handle, osWaitForever);
    if (event.status == osEventMail)
    {
    if (!(pMail = event.value.p))
    {
        // 如果mail queue为空, 则此时pMail为空指针
        // 等待500ms, 此时系统调度会切换线程
        printf("[C]Empty!.\n");
        osDelay(500);
        continue;
    }
    printf("[C]%d<<\n", pMail->var);
    osMailFree(mail01Handle, pMail);
    }
}
/* USER CODE END MsgConsumerTask */
}
```

结果(串口终端)

1. Producer每次塞入四条消息后, 被阻塞, 此时调度器切换到Consumer, 接收四条消息.
2. 当Consumer消化完4条信息后,再一次调用osMailGet(mail01Handle, osWaitForever)时,调度器会切换线程到Producer.

```
...
[C]120<<
[C]121<<
[C]122<<
[C]123<<
[P]>>124
[P]>>125
[P]>>126
[P]>>127
[P]Full! Wait.
[C]124<<
[C]125<<
[C]126<<
[C]127<<
[P]>>128
```

```
[P]>>129
[P]>>130
[P]>>131
[P]Full! Wait.
[C]128<<
[C]129<<
[C]130<<
[C]131<<
[P]>>132
[P]>>133
[P]>>134
[P]>>135
[P]Full! Wait.
...
```

2.使用信号量解决生产者-消费者问题

配置

1. 新建一个`CountingSema01`的计数信号量,初始`Count`值设为4.

代码

1. `Producer`部分:

```
void MsgProducerTask(void const * argument)
{
  /* USER CODE BEGIN MsgProducerTask */
  u_int8_t i=0;
  /* Infinite loop */
  for (;;)
  {
    // 不断尝试生产
    printf("[Producer>] Try produce %d.\n",i);
    osSemaphoreWait(CountingSem01Handle, osWaitForever);
    printf("[Producer>] %d produced.\n",i++);
  }
  /* USER CODE END MsgProducerTask */
}
```

2. `Consumer`部分:

```
void MsgConsumerTask(void const * argument)
{
  /* USER CODE BEGIN MsgConsumerTask */
  /* Infinite loop */
  for (;;)
  {
    // 每隔1s尝试消费一次
    osDelay(1000);
    printf("[>Consumer] Cosumes.\n");
    osSemaphoreRelease(CountingSem01Handle);
  }
  /* USER CODE END MsgConsumerTask */
}
```

结果(串口终端)

1. `Producer`不断尝试生产, `Consumer`每1s尝试消费一次.
2. `Producer`尝试生产第5个时, 调用`osSemaphoreWait()`被阻塞.
3. 调度器切换线程到`Consumer`.
4. `Consumer`每消费一次后进入`osDelay()`,调度器切换线程.
5. `Producer`生产一次后又被`osSemaphoreWait()`阻塞,调度器切换线程.
6. 步骤4和步骤5循环

```
[Producer>] Try produce 0.
[Producer>] 0 produced.
[Producer>] Try produce 1.
[Producer>] 1 produced.
[Producer>] Try produce 2.
[Producer>] 2 produced.
[Producer>] Try produce 3.
[Producer>] 3 produced.
[Producer>] Try produce 4.
[>Consumer] Cosumes.
[Producer>] 4 produced.
[Producer>] Try produce 5.
[>Consumer] Cosumes.
[Producer>] 5 produced.
[Producer>] Try produce 6.
[>Consumer] Cosumes.
[Producer>] 6 produced.
[Producer>] Try produce 7.
...
```

遇到的问题及解决方法

1. 尝试使用`printf()`进行串口打印,无输出.

上次看门狗实验中,只要重载`__io_putchar()`函数即可使用`printf()`在串口打印,但这次串口无输出.
打开上次的看门狗实验, 在`__io_putchar()`函数内打断点, 使用st-link进行Debug. 程序运行时于该处暂停, 查看此时的调用堆栈:

- __io_putchar@0x08000e8e (Src\main.c:78)
- **_write@0x080010fa (Src\syscalls.c:112)**
- _write_r@0x08003938 (_write_r.dbgasm:10)
- __sflush_r@0x08002bb4 (__sflush_r.dbgasm:111)
- __swbuf_r@0x08001d2a (__swbuf_r.dbgasm:51)
- __sfputs_r@0x0800364c (__sfputs_r.dbgasm:14)
- _vfprintf_r@0x080036b4 (_vfprintf_r.dbgasm:41)
- printf ...

对比后发现,如果使用FreeRTOS,那么`Src\syscalls.c`文件并不存在, 故`_wrtie`函数没有实现. 所以需要在这次实验的代码中自己手动实现`_write`函数.
观察看门狗实验里`Src\syscalls.c`中的`_write`函数:

```c
__attribute__((weak)) int _write(int file, char *ptr, int len)
{
    int DataIdx;

    for (DataIdx = 0; DataIdx < len; DataIdx++)
    {
        __io_putchar(*ptr++);
    }
    return len;
}
```

将其复制到`freertos.c`下,修改修饰符,编译代码并下载到板子上运行,成功在串口输出.