# 《高级操作系统》
# 实验2

内存分配和管理

# 实验任务

- 任务一：用kmalloc 分配1KB、8KB的内存并打印指针地址
- 任务二：用vmalloc 分配8KB、1MB，64MB的内存并打印指针地址
- 任务三：申请、读写、释放I/O端口
- 任务四：申请、读写、释放I/O内存

# 实验回顾：编译运行

```
#include<linux/module.h>                    //包含了对模块的结构定义以及模块的版本控制

MODULE_LICENSE("GPL");                    //声明GPL版权
static __init module_init(void){          //加载模块
     。。。。。。
}

static __exit module_exit(void){          //卸载模块
     。。。。。。
}
module_init(module_init);
module_exit(module_exit);
```

```
kas build  common-oscourse-qemuriscv64.yml
kas shell common-qemuriscv64-core-image-minimal.yml -c "runqemu nographic"
```

# 实验回顾：运行

- 内核模块常用操作
  - 加载内核模块：insmod
    - 如： insmod hello.ko
  - 卸载内核模块：rmmod
    - 如：rmmod hello.ko
  - 查看内核模块：lsmod
    - 如：lsmod | grep hello

  - 加载/卸载内核模块后，查看模块打印信息：
    - dmesg | tail -n 2
      tail -n <行数> 显示文件的尾部 n 行内容

# 任务一

- kmalloc
  - 是个功能强大且高速(除非被阻塞)的工具，
  - 所分配到的内存在物理内存中连续且保持原有的数据（不清零）。

- 原型
  - #include <linux/slab.h>
  - void *kmalloc(size_t size, int flags);

# 任务一

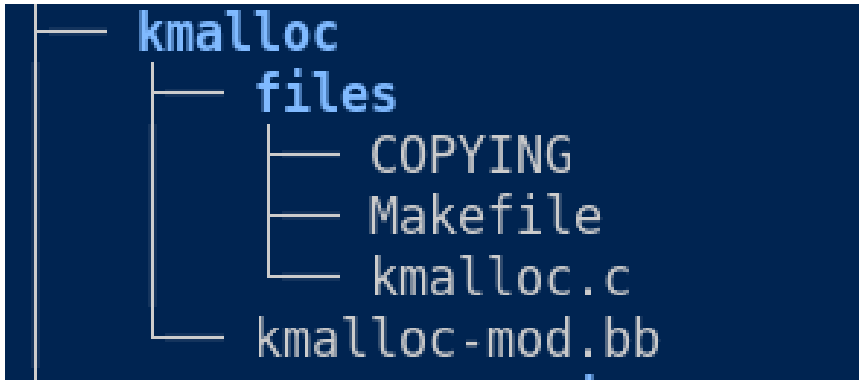
- size 参数说明
  - 内核管理系统的物理内存，物理内存只能按页面进行分配。
  - kmalloc 和典型的用户空间 malloc 在实际上有很大的差别，内核使用特别的基于页的分配技术，以最好的方式利用系统 RAM。Linux 处理内存分配的方法：创建一系列内存对象集合，每个集合内的内存块大小是固定。
  - 处理分配请求时，就直接在包含有足够大内存块的集合中传递一个整块给请求者。
- 注意
  - 内核只能分配一些预定义的、固定大小的字节数组。
  - kmalloc 能够处理的最小内存块是 32 或 64 字节（体系结构依赖），而内存块大小的上限随着体系和内核配置而变化。
  - 考虑到移植性，不应分配大于 128 KB的内存。若需多于几个 KB的内存块，最好使用其他方法。

# 任务一

- flags 参数说明
  - 内存分配最终总是调用 __get_free_pages 来进行实际的分配，这就是 GFP_ 前缀的由来。
  - 任何标志都定义在 <linux/gfp.h>，有符号代表常常使用的标志组合。
- 具体可选参数：
  - GFP_ATOMIC —— 分配内存的过程是一个原子过程，分配内存的过程不会被（高优先级进程或中断）打断；
  - GFP_KERNEL —— 正常分配内存；
  - GFP_DMA —— 给 DMA 控制器分配内存，需要使用该标志（DMA要求分配虚拟地址和物理地址连续）

# 任务一

- 具体要求
  - 编写内核模块，调用kmalloc()函数，并分配1KB、8KB内存，打印指针地址
  - 编写makefile文件，执行kas 编译模块，运行qemu 镜像
  - 加载模块，查看加载的模块内容，即打印出来的指针地址



# 任务一：运行结果演示

编译完成后环境内容 build/tmp-glibc/work/qemuriscv64-oe-linux/kmalloc-mod/1.0-r0

```
[root@openEuler ve]# cd task1.1
[root@openEuler task1.1]# ls
Makefile  Module.symvers  kmalloc.c  kmalloc.ko  kmalloc.mod.c  kmalloc.mod.o  kmalloc.o  modules.order
[root@openEuler task1.1]# pwd
/root/raspberrypi-kernel/ve/task1.1
```

1. 安装内核模块

```
sudo insmod kmalloc.ko
```

2. 查看输出

```
sudo dmesg | tail -n 2
```

```
[root@openEuler task1.1]# sudo insmod kmalloc.ko
[root@openEuler task1.1]# sudo dmesg | tail -n 2
[172277.766590] kmallocmem1 addr = ffffb140c179c400
[172277.766608] kmallocmem2 addr = ffffb140f5126000
```

# 任务二

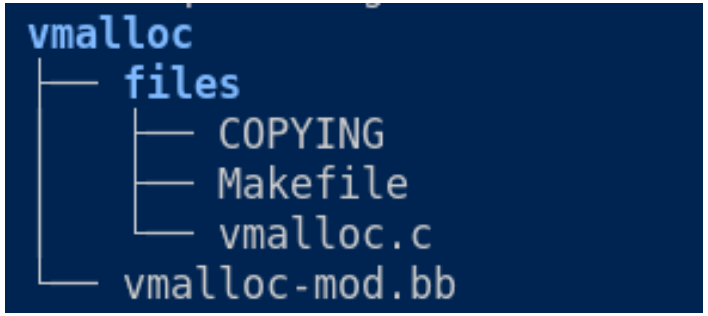
- kmalloc的限制
  - 申请的内存位于物理内存映射区域，而且在物理上也是连续的，
  - 它们与真实的物理地址只有一个固定的偏移，
  - 因为存在较简单的转换关系，所以对申请的内存大小有限制，不能超过128KB。

# 任务二

- vmalloc
  - vmalloc() 函数则会在虚拟内存空间给出一块连续的内存区，但这片连续的虚拟内存在物理内存中并不一定连续。
  - 由于 vmalloc() 没有保证申请到的是连续的物理内存，因此对申请的内存大小没有限制，如果需要申请较大的内存空间就需要用此函数了。
  - 由于直接内存映射区（3GB ~ 3GB+896MB）是直接映射到物理地址（0 ~ 896MB）的，所以内核不能通过直接内存映射区使用到超过 896MB 之外的物理内存。
  - 这时候就需要提供一个机制能够让内核使用 896MB 之外的物理内存，所以 Linux 就实现了一个 vmalloc 机制。

# 任务二：题目要求

- 编写内核模块，调用vmalloc()函数，分配8KB、1MB、64MB内存，并打印指针地址
- 编写makefile文件，执行kas 编译模块，运行qemu 镜像
- 加载模块，查看加载的模块内容，即打印出来的指针地址
- 根据机子是32位或者是64位的情况，分析地址落在的区域，并给出相应的解释.



# 任务二：运行结果演示

编译完成后环境内容 build/tmp-glibc/work/qemuriscv64-oe-linux/vmalloc-mod/1.0-r0

```
[root@openEuler task1.2]# ls
Makefile  Module.symvers  modules.order  vmalloc.c  vmalloc.ko  vmalloc.mod.c  vmalloc.mod.o  vmalloc.o
[root@openEuler task1.2]# pwd
/root/raspberrypi-kernel/ve/task1.2
```

1. 安装内核模块

```
sudo insmod vmalloc.ko
```

2. 查看输出

```
sudo dmesg | tail -n 2
```

```
[root@openEuler ve]# cd task1.2
[root@openEuler task1.2]# ls
Makefile  Module.symvers  modules.order  vmalloc.c  vmalloc.ko  vmalloc.mod.c  vmalloc.mod.o  vmalloc.o
[root@openEuler task1.2]# sudo insmod vmalloc.ko
[root@openEuler task1.2]# sudo dmesg | tail -n 2
[172750.160670] vmallocmem2 addr = ffff00000b825000
[172750.186017] vmallocmem3 addr = ffff0000151dd000
```

# 任务三

- 申请、读写、释放I/O端口

# 任务三

- I/O端口
  - I/O接口电路需要设置若干专用寄存器，缓冲输入输出数据，设定控制方式，保存输入输出状态信息等，这些寄存器可被CPU直接访问，常称为端口。
  - 根据端口传输的信息，端口可分为数据端口、状态端口和控制端口，用以传输数据信息、状态信息和控制信息。

相关函数：request_region 用于申请 IO 端口号

# 任务三

- void request_region(unsigned long from,
  unsigned long num,
  const char *name)
  - 这个函数用来申请一块输入输出区域。
    如果这段I/O端口没有被占用，在我们的驱动程序中就可以使用它。在使用之前，必须向系统登记，以防止被其他程序占用。登记后，在/proc/ioports文件中可以看到你登记的io口。
  - 参数1：io端口的基地址。
  - 参数2：io端口占用的范围。
  - 参数3：使用这段io地址的设备名。

# 任务三

- 输入端口具有的能力：
  - CPU从输入端口输入数据时，要求外部设备事先将数据准备好，当外设有数据要发给CPU时，数据应先保存在输入端口上，不能直接进入系统的数据总线上，当需要读取数据时，输入端口就把数据放到数据总线上，这就是输入端口必须具有的通断控制能力。
  - 总线结构的计算机系统，除了CPU任何其他部件都不能直接加载数据到数据总线上。
  - 设备的接口电路挂接在系统总线上，但是设备输出的数据不能直接加载到数据总线。
  - 接口上需要有一个“开关”一样的电路，当CPU读取外部设备的数据时，这个“开关”就将设备的数据加载到数据总线上，然后CPU通过数据总线接收数据。

# 任务三：题目要求

- 编写内核模块，调用内核相关接口，实现申请I/O端口、读写I/O端口、释放I/O端口；并打印输出相关信息。
- 编写makefile文件，执行kas 编译模块，运行qemu 镜像；
- 加载模块，查看加载的模块内容。

```
├── request_region
│   ├── files
│   │   ├── COPYING
│   │   ├── Makefile
│   │   └── request_region.c
│   └── request-region.bb
```

# 任务三：运行结果演示

编译完成后环境内容 build/tmp-glibc/work/qemuriscv64-oe-linux/request-region/1.0-r0

```
[root@openEuler task2.1]# ls
Makefile  Module.symvers  modules.order  request_region.c  request_region.ko  request_region.mod.c  request_region.mod.o  request_region.o
[root@openEuler task2.1]# pwd
/root/raspberrypi-kernel/ve/task2.1
```

1、安装内核模块

```
sudo insmod request_region.ko
```

2、查看输出

```
sudo dmesg | tail -n 5
```

```
/root/raspberrypi-kernel/ve/task2.1
[root@openEuler task2.1]# sudo insmod request_region.ko
[root@openEuler task2.1]# sudo dmesg | tail -n 5
[174763.660003] brcmfmac: brcmf_cfg80211_set_power_mgmt: power save enabled
[175070.980766] Start request region!
[175070.980805] it's ok for 22222.
[175079.662674] IPv6: ADDRCONF(NETDEV_UP): wlan0: link is not ready
[175079.662739] brcmfmac: brcmf_cfg80211_set_power_mgmt: power save enabled
```

ISCAS

# 任务四

- 申请、读写、释放I/O内存

# 任务四

- 根据访问外设寄存器的不同方式，可以把CPU分成两大类。
- 一类CPU（如M68KPower PC等）把这些寄存器看作内存的一部分
  - 寄存器参与内存统一编址，访问寄存器只需使用一般的内存指令进行，即这类CPU不需要有专门用于设备I/O的指令。
  - 这就是所谓的“I/O内存”方式。
- 另一类CPU（如X86)将外设的寄存器看成一个独立的地址空间。
  - 这样访问内存的指令就不能用来访问这些寄存器因此要为对外设寄存器的读/写设置专用指令，如IN和OUT指令。
  - 这就是所谓的“I/O端口”方式。



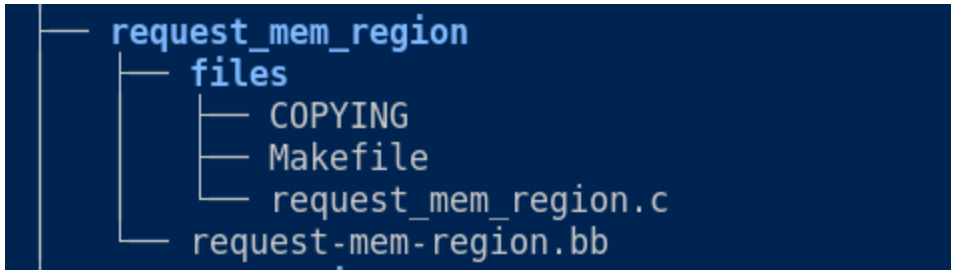
相关函数：request_mem_region 用于申请 IO 地址

# 任务四

- 独立编址主要优点
  - I/O端口地址不占用存储器空间；使用专门的I/O指令对端口进行操作，I/O指令短，执行速度快。
  - 并且由于专门I/O指令与存储器访问指令有明显的区别，使程序中I/O操作和存储器操作层次清晰，程序的可读性强。
  - 同时，由于使用专门的I/O指令访问端口，并且I/O端口地址和存储器地址是分开的，故I/O端口地址和存储器地址可以重叠，而不会相互混淆。
  - 译码电路比较简单(因为I/O端口的地址空间一般较小，所用地址线也就较少)。
- 缺点
  - 只能用专门的I/O指令，访问端口的方法不如访问存储器的方法多。

# 任务四：题目要求

- 编写内核模块，调用内核相关接口，实现申请I/O内存、释放I/O内存；并打印输出相关信息。
- 编写makefile文件，执行kas 编译模块，运行qemu 镜像；
- 加载模块，查看加载的模块内容。



# 任务四：运行结果演示

编译完成后环境内容 ucas/build/tmp-glibc/work/qemuriscv64-oe-linux/request-mem-region/1.0-r0

```
[root@openEuler task2.2]# ls
Makefile  Module.symvers  modules.order  request_mem_region.c  request_mem_region.ko  request_mem_region.mod.c  request_mem_region.mod.o  request_mem_region.o
[root@openEuler task2.2]# pwd
/root/raspberrypi-kernel/ve/task2.2
```

1. 安装内核模块

```
sudo insmod request_mem_region.ko
```

2. 查看输出

```
sudo dmesg | tail -n 5
```

```
[root@openEuler task2.2]# sudo dmesg | tail -n 5
[175395.600317] brcmfmac: brcmf_cfg80211_set_power_mgmt: power save enabled
[175496.130532] Start request mem region!
[175496.130572] it's ok for 994115584 .
[175711.605239] IPv6: ADDRCONF(NETDEV_UP): wlan0: link is not ready
[175711.605301] brcmfmac: brcmf_cfg80211_set_power_mgmt: power save enabled
```

# 作业

- 任务一：用kmalloc 分配1KB、8KB的内存并打印指针地址
- 任务二：用vmalloc 分配8KB、1MB，64MB的内存并打印指针地址
- 任务三：申请、释放I/O端口
- 任务四：申请、释放I/O内存

作业提交：
每一个任务关键操作截图，所有任务结果保存到一个Word文件中。
内核模块以自己的学号命名。

交作业截止日期：下周五 0:00之前。

# 本节完