|  |
| --- |
| 1. 实验目的及实验内容   （本次实验所涉及并要求掌握的知识；实验内容；必要的原理分析） |
| 在已有实验代码基础上，将1-6章节进行功能综合，采用汇编和C  语言，实现一个mini-os  – 功能要求：  • 用软盘启动该mini-OS，如果能够硬盘启动更好。  • 能够实现内存的分配与释放  • 能够实现多进程管理与调度  • 能够提供一个定时触发的系统中断调用，完成对指定进程内存结构的完整性度量，  并打印结果  • 所有代码需用目录树结构管理，并添加完整的makefile编译，以及文档  学期综合实验二  （线下自学部分，三选一）  2.内核模块的编写：完成一个Linux/Windows内核/驱动模块的编写，  能够实现对文件访问的监控、或者对键盘设备、USB设备、网络设备、  蓝牙设备等的监控。 |
| 1. 实验环境及实验步骤   （本次实验所使用的器件、仪器设备等的情况；具体的实验步骤） |
| **1. 内存的分配与释放**  **参照资料为：**  **<https://www.jianshu.com/p/49cbaccd38c5>**  （1）先看能用的内存空间。  在Loader.asm中，作者给出的orange’s内存结构图    可以看到，我们可以使用的空间为7E00H~30000H。  为方便进行取整，使用从10000H开始到30000H的地址空间。  （2）初始化位图bitmap  10000H~30000H大小为20000H，为128K，一页大小为4k，相除正好得到32。故用32位的位图即可记录。  Kernel.asm中，数据段初始化位图。    （3）增加分配和释放内存函数  Kernel.asm中。Global声明进行导出    二者实现函数：    alloc\_page:      xor eax,eax  .1: bts [bitmap],eax      jnc .2      inc eax      cmp eax,bitmap\_len      jl .1  .2:      shl eax,12      add eax,0x10000      ret  free\_page:      push    ebp      mov ebp, esp      xor eax,eax      mov eax,[ebp+8]      sub eax,10000h      shr eax,12      btr [bitmap],eax        pop ebp      ret  proto.h中声明参数.    （3）加入测试条件，编译测试结果。    连续分配两页，再释放一页，再分配一页，再全部释放。      可以看到首先连续分配两页，释放第一页后，再分配又使用了第一页，说明分配和释放都是成功的。  **2.能够实现多进程管理与调度**  **这里模拟实现一个多级反馈队列调度算法，并用其尝试调度5个任务，输出性能评价信息。**   1. 添加到5个任务。       （2）添加进程结构成员和变量  添加指针，方便创建链表队列。  每个进程完成所需的tick。  开始时间。  是否完成的指示位。    加入全局变量    用来做队列。  （3）初始化工作    （4）调度算法  注：实现了抢占和非抢占的两种算法。  PUBLIC void schedule()  {      PROCESS\* p;      PROCESS\* t;      int time;      int  greatest\_ticks = 0;      if(p1!=NULL)    //队列非空      {          if (p1->ticks<=0 || p1->rq\_ticks<=0)   //时间片到了或者已经跑完          {              if (p1->rq\_ticks>0)  //如果没有跑完,入队列2              {                  t=p1;                  if(p2==NULL)                  {                      p2=t;                      p1=p1->next;                      t->next=NULL;                  }                  else                  {                      p=p2;                      while(p->next!=NULL)                      {                          p=p->next;                      }                      p1=p1->next;                      p->next=t;                      t->next=NULL;                  }                  t->ticks=50;              }              else//打印跑完              {                  disp\_str(p1->p\_name);                  disp\_str(" is done,starts at ");                  disp\_int(p1->start\_ticks);                  disp\_str(" finished at ");                  disp\_int(ticks);                  time=ticks-p1->start\_ticks;                  disp\_str(" time all :");                  disp\_int(time);                  p1=p1->next;              }              if(p1!=NULL)              {                  p\_proc\_ready = p1;              }              else //如果是最后一个,开始队列2              {                  disp\_str("p1finished!");                  p\_proc\_ready = p2;              }          }      }      else if (p2!=NULL)      {          if (p2->ticks<=0 || p2->rq\_ticks<=0)   //时间片到了或者已经跑完          {              if (p2->rq\_ticks>0)  //如果没有跑完,入队列3              {                  t=p2;                  if(p3==NULL)                  {                      p3=t;                      p2=p2->next;                      t->next=NULL;                  }                  else                  {                      p=p3;                      while(p->next!=NULL)                      {                          p=p->next;                      }                      p2=p2->next;                      p->next=t;                      t->next=NULL;                  }                  t->ticks=20;              }              else//打印跑完              {                  disp\_str(p2->p\_name);                  disp\_str(" is done,starts at ");                  disp\_int(p2->start\_ticks);                  disp\_str(" finished at ");                  disp\_int(ticks);                  time=ticks-p2->start\_ticks;                  disp\_str(" time all :");                  disp\_int(time);                  p2=p2->next;              }              if(p2!=NULL)              {                  p\_proc\_ready = p2;              }              else //如果是最后一个,开始队列3              {                  disp\_str("p2finished!");                  p\_proc\_ready = p3;              }          }      }      else if (p3!=NULL)      {          if (p3->ticks<=0 || p3->rq\_ticks<=0)   //时间片到了或者已经跑完          {              if (p3->rq\_ticks>0)  //如果没有跑完,入队列3              {                  t=p3;                  if(p3==NULL)                  {                      p3=t;                      p3=p3->next;                      t->next=NULL;                  }                  else                  {                      p=p3;                      while(p->next!=NULL)                      {                          p=p->next;                      }                      p3=p3->next;                      p->next=t;                      t->next=NULL;                  }                  t->ticks=20;              }              else//打印跑完              {                  disp\_str(p3->p\_name);                  disp\_str(" is done,starts at ");                  disp\_int(p3->start\_ticks);                  disp\_str(" finished at ");                  disp\_int(ticks);                  time=ticks-p3->start\_ticks;                  disp\_str(" time all :");                  disp\_int(time);                  p3=p3->next;              }              if(p3!=NULL)              {                  p\_proc\_ready = p3;              }              else //如果是最后一个,开始队列3              {                  disp\_str("p3finished!");                  //p\_proc\_ready = p3;              }          }      }  }  PUBLIC void schedule()  {      PROCESS\* p;          PROCESS\* t;      int  time = 0;      if(p1!=NULL)//队列不为空      {          if(p1->ticks <= 0 || p1->rq\_ticks <= 0) //时间片到了或者已经跑完          {          //调度下一个进程\*/                if (p1->rq\_ticks>0) //如果未跑完              {                  t=p1;    //保存当前进程                  p=p1;   //找到队列尾部                  while(p->next!=NULL)                  {                      p=p->next;                  }                  p->next=t;          p1=t->next;                  t->next=NULL; //当前进程进入队尾                  //修改时间片和优先级                  if (t->priority==1)                  {                      t->priority=2;                      t->ticks=50;                  }                  if (t->priority==2 || t->priority==3)                  {                      t->priority=3;                      t->ticks=20;                  }              }              else //如果跑完了              {          if (p1->fin==0)                  {              disp\_str(p1->p\_name);              disp\_str(" time consumed");              time=ticks-p1->start\_ticks;              disp\_int(time);              p1->fin=1;          }          if (p1->next!=NULL)          {              p1=p1->next;          }              }              p\_proc\_ready = p1;          }      }  }  实现结果：  抢占    非抢占    **3 能够提供一个定时触发的系统中断调用，完成对指定进程内存结构的完整性度量**  （1）添加过程，用于接受参数 syscall.asm    （2）系统调用的中断门已经之前初始化不再修改 protect.c    （3）添加syscalltable成员 global.c    （4）crc校验函数。 Proc.c  参照https://www.cnblogs.com/zzdbullet/p/9580502.html  PUBLIC int sys\_get\_crc()  {      PROCESS\* p = p\_proc\_ready;  //当前进程      int num=0;      int a=0;      int b=0;      a=&proc\_table[1];      b=&proc\_table[0];   //得到进程表字节大小      num=a-b;      u8 \*addr=&p;      u16 crc=0x0000;      int i;      for(;num>0;num--){          crc=crc^(\*addr++);          for(i=0;i<8;i++){              if(crc&0x8000)                  crc=(crc<<1)^0xA001;              else                  crc<<=1;          }          crc &= 0xFFFF;      }      return (crc);  }  （5）添加函数声明 proto.h    （6）修改调用个数 const.h    （7）添加函数调用，定时调用    (8)编译运行，结果    内核模块的编写：  完成一个Linux/Windows内核/驱动模块的编写，能够实现对文件访问的监控、或者对键盘设备、USB设备、网络设备、蓝牙设备等的监控。  本次实现对键盘设备的监控。  了解linux内核模块编写的基本知识：      Ubuntu虚拟机  首先，需要了解键盘驱动的原理。（摘自书写基于Linux内核的键盘记录器）    首先，当你输入一个键盘值的时候。键盘将会  发送相应的scancodes给键盘驱动。一个独立的击键  可以产生一个六个scancode的队列  当接受到一个键盘中断时。键盘中断器会读取scancode和键盘的状态。读写键盘事件都是通过Ox60端口(键盘数据注册器)和0x64(键盘状态注册器)来实现的。  此处采用request\_irq（）函数注册中断服务，当中断产生的时候会去调用写好的中断服务。    产生中断时，从0x60键盘数据设备中得到1个字节的扫描码，为了防止在ISR中嵌套导致混乱，将处理写入记录文件的函数放入任务队列中，当不处于临界区  时再写入。        这里根据scancode表进行解析。将解析完的token先写入buf，然后再写入文件中。  这里记得要分为常用类，小键盘和附加键，常用键通常只会产生1个字节的内容。  完善了不可见字符（键盘上所有字符都能记录）；    这里对更长的就需要引入全局变量来进行控制。  这里还涉及到内核模块的读写文件。  要用filp\_open、filp\_close();    效果如图。 |
| 1. 实验过程分析   （实验分工，详细记录实验过程中发生的故障和问题，进行故障分析，说明故障排除的过程及方法。根据具体实验，记录、整理相应的数据表格等） |
| 1.网上的scancode有很多是有问题的；  https://blog.csdn.net/cosmoslife/article/details/9066735  这个没问题。  2.scancode有的不止1个字节，需要提前进行判断。 |