张慕晖的博客
LUX ET VERITAS

2018-05-23

《操作系统》2015年期末考试分析

试题来自。这个卷子实在过于长了，不仅莫名有一个完整的缓冲区问题的实现，还有一堆ucore代码需要阅读和填空。

一（10分）

在用`do_execve`启动一个用户态进程时，ucore需要完成很多准备工作，这些工作有的在内核态完成，有的在用户态完成。请判断下列事项是否是ucore在正常完成`do_execve`中所需要的，如果是，指出它完成于内核态还是用户态（通过修改`trapframe`，在`iret`时改变寄存器的过程被认为是在内核态完成）。

1. 初始化进程所使用的栈
2. 在栈上准备argc和argv的内容
3. 将argc和argv作为用户main函数的参数放到栈上
4. 设置EIP为用户main函数的地址
5. 设置系统调用的返回值

---

1. 需要；内核态
2. 需要；内核态
3. ~~需要；内核态~~不需要；用户态
4. ~~需要；内核态~~不需要；用户态
5. 不需要

这个题出的很没有意义啊，系统调用返回之后几乎就要立即跳转到用户进程指令的第一条了。

以下内容摘自ucore docs Lab5：

> 最终通过`do_execve`函数来完成用户进程的创建工作。此函数的主要工作流程如下：

- 首先为加载新的执行码做好用户态内存空间清空准备。如果mm不为NULL，则设置页表为内核空间页表，且进一步判断mm的引用计数减1后是否为0，如果为0，则表明没有进程再需要此进程所占用的内存空间，为此将根据mm中的记录，释放进程所占用户空间内存和进程页表本身所占空间。最后把当前进程的mm内存管理指针为空。由于此处的initproc是内核线程，所以mm为NULL，整个处理都不会做。
- 接下来的一步是加载应用程序执行码到当前进程的新创建的用户态虚拟空间中。这里涉及到读ELF格式的文件，申请内存空间，建立用户态虚存空间，加载应用程序执行码等。load_icode函数完成了整个复杂的工作。
  ……
  load_icode函数的工作：

1. 初始化mm
2. 分配和设置页目录表
3. 解析ELF文件，建立vma，初始化进程的用户态虚拟地址空间
4. 分配物理内存空间，建立页表映射关系，拷贝程序内容
5. 设置用户栈
6. 将页目录表基地址加载到CR3寄存器中
7. 重设进程中断帧，准备切换到用户态
   至此，用户进程的用户环境已经搭建完毕。此时initproc将按产生系统调用的函数调用路径原路返回，执行中断返回指令“iret”（位于trapentry.S的最后一句） 后，将切换到用户进程hello的第一条语句位置_start处（位于user/libs/initcode.S的第三句） 开始执行。

2018.5.25 UPD：
tsz同学指出，事实上这道题和第六大题的代码内容直接相关。从中可以看出，`user/libs/initcode.S`做的就是在用户态为`main`函数设置参数的工作。所以3和4的答案应该修改一下。事实证明，想当然是不好的。

二 VSFS（18分）

这道题和MOOC期末考试题中的第20题一模一样，所以略。

三 进程状态变化（16分）

在ucore中`enum proc_state`的定义包含以下四个值：

- `PROC_UNINIT`
- `PROC_SLEEPING`
- `PROC_RUNNABLE`
- `PROC_ZOMBIE`
  请解释每一种状态的含义，以及各状态之间可能的迁移。

- `PROC_UNINIT`：刚申请完进程控制块，进程还未被初始化
- `PROC_SLEEPING`：进程处于等待状态
- `PROC_RUNNABLE`：进程处于就绪或运行状态
- `PROC_ZOMBIE`：僵尸状态，进程已经退出，等待父进程进一步回收资源

以下内容（进程的正常生命周期）摘自ucore docs Lab6：

> 进程的正常生命周期如下：

- 进程首先在cpu初始化或者`sys_fork`的时候被创建，当为该进程分配了一个进程控制块之后，该进程进入`uninit`态(在`proc.c`中`alloc_proc`)。
- 当进程完全完成初始化之后，该进程转为`runnable`态。
- 当到达调度点时，由调度器`sched_class`根据运行队列`rq`的内容来判断一个进程是否应该被运行，即把处于`runnable`态的进程转换成`running`状态，从而占用CPU执行。
- `running`态的进程通过`wait`等系统调用被阻塞，进入`sleeping`态。
- `sleeping`态的进程被`wakeup`变成`runnable`态的进程。
- `running`态的进程主动`exit`变成`zombie`态，然后由其父进程完成对其资源的最后释放，子进程的进程控制块成为`unused`。
- 所有从`runnable`态变成其他状态的进程都要出运行队列，反之，被放入某个运行队列中。

以下内容摘自进程运行状态转变过程：

```
process state changing:

  alloc_proc                                  RUNNING
      +                                   +--<----<--+
      +                                   + proc_run +
      V                                   +-->---->--+
PROC_UNINIT -- proc_init/wakeup_proc --> PROC_RUNNABLE --
try_free_pages/do_wait/do_sleep --> PROC_SLEEPING --
                                           A      +
+
                                           |      +--- do_exit --> PROC_ZOMBIE
+
                                           +
+
                                           ----------------------wakeup_proc----------
------------------------
```

四 Stride调度算法（15分）

假设在lab6测试stride scheduling的过程中，采用如下默认配置：BigStride为0x7FFFFFFF，CPU时间片为50ms，测试过程包含五个进程，其初始~~stride~~pass均为1，优先级分别为1、2、3、4、5，测试时间为10s。下面给出了五种修改上述配置的方式，试讨论：对于~~每~~一种改动，测试结果相比改动之前是否会发生明显的变化？如果是，结果会变得更接近于理想情况，还是远离理想情况？

1. BigStride改为120
2. CPU时间片改为5ms
3. 五个进程的初始pass改为100

4. 五个进程的优先级设为2、4、6、8、10
5. 测试时间延长到20s

在测试时间10s的情况下，时间片总个数为200。

1. 如果将BigStride改为120，则stride最大为120，不会溢出，而且120能够整除1、2、3、4、5，更能够保证进程的pass按优先级推进，因此会更接近于理想情况
2. 时间片总个数变成2000，因为进程stride有偏差，因此会远离理想情况
3. 因为100这个值相比各个进程的stride太小了，所以应该不会有明显变化
4. 不会有明显变化
5. 同2，更远离理想情况

这个题目中不同学长的答案大相径庭，所以我选了一种我觉得合理的。事实上，Stride调度算法的论文中讨论了一下误差问题：在stride和优先级精确地成反比的情况下，各个线程之间按比例分配到的时间片数量的误差不超过1，也就是说，总误差是O(nc)（nc是线程数量）。所以大概stride计算不准确造成的影响是比较大的。

五 生产者-消费者问题（10分）

生产者-消费者问题是指，一组生产者进程和一组消费者进程共享一个初始为空、大小为3（不如说是BUFFER_SIZE）的缓冲区，只有缓冲区没满时，生产者才能把消息放入到缓冲区，否则必须等待；只有缓冲区不空时，消费者才能从中取出消息，否则必须等待。由于缓冲区是临界资源，它只允许一个生产者放入消息，或者一个生产者放入消息，或者一个消费者从中取出消息。

下面是生产者-消费者问题的一个实现和测试结果。请回答下面问题：

1. 请用伪码给出信号量的PV操作实现。
2. 这个实现正确吗？如果不正确，给出你的正确实现。
3. 这两个测试用例能发现该实现中的可能错误吗？如果不能，请给出你的尽可能完整的测试用例。

```
    ==== producer-consumer.cpp ====
    #include <stdio.h>
1   #include <pthread.h>
2   #include <semaphore.h>
3   #include <cstring>
4   #include <unistd.h>
5   #include <string>
6   #include <cstdlib>
7   #include <new> // ::operator new[]
8
9   using namespace std;
10
11  #define BUFFER_SIZE 3
12  #define SLEEP_SPAN 5
13  #define WORK_SPAN 4
14
```

```
#define PRODUCER 0
#define CONSUMER 1

int iflag = 0;
int oflag = 0;
sem_t empty, full, mutex;
int empty_count, full_count;
int data_num = 0;
int num = 0;

int buffer[BUFFER_SIZE] = {};

int p_task_done = -1;
int c_task_done = -1;

struct arg_struct {
    arg_struct(int _id, int _start, int _work, string _indent): id(_id),
start(_start), work(_work), indent(_indent) {}
    arg_struct(int _id): id(_id), start(0), work(0), indent(string("")) {}
    int id;
    int start;
    int work;
    string indent;
};

void* producer(void* argv){
    arg_struct arg = *(arg_struct*)argv;
    int id = arg.id;
    const char* indent = arg.indent.c_str();

    sleep(arg.start);

    printf("%sSTART\n", indent);

    sem_wait(&mutex);  /* 顺序错了 */
    printf("%saMUTEX\n", indent);

    sem_wait(&empty);  /* 顺序错了 */
    printf("%saEMPTY\n", indent);

    printf("%sENTER\n", indent);

    int time = rand() % SLEEP_SPAN;
    sleep(arg.work);

    p_task_done++;
    printf("%sProd %d\n", indent, p_task_done);
```

```
62
63      buffer[iflag] = p_task_done;
64
65      if (empty_count == 0) printf("Error: Produce while no empty\n");
66      iflag = (iflag + 1) % BUFFER_SIZE;
67      empty_count--;
68      full_count++;
69
70      printf("%sEXIT\n", indent);
71
72      sem_post(&mutex);
73      printf("%srMUTEX\n", indent);
74
75      sem_post(&full);
76      printf("%srFULL\n", indent);
77
78      return NULL;
79  }
80
81  void* consumer(void* argv){
82      arg_struct arg = *(arg_struct*)argv;
83      int id = arg.id;
84      const char* indent = arg.indent.c_str();
85
86      sleep(arg.start);
87
88      printf("%sSTART\n", indent);
89      sem_wait(&full);
90      printf("%saFULL\n", indent);
91
92      sem_wait(&mutex);
93      printf("%saMUTEX\n", indent);
94
95      printf("%sENTER\n", indent);
96
97      sleep(arg.work);
98
99      ++c_task_done;
100     if (full_count == 0) printf("Error: Consume while no full\n");
101
102     int tmp = buffer[oflag];
103     printf("%sCons %d\n", indent, tmp);
104
105     oflag = (oflag + 1) % BUFFER_SIZE;
106     if (c_task_done != tmp) printf("Error: Consume data wrong\n");
107     if (c_task_done > p_task_done) printf("Error: Over-consume!\n");
108
```

```
109     full_count--;
110     empty_count++;
111
112     printf("%sEXIT\n", indent);
113
114     sem_post(&mutex);
115     printf("%srMUTEX\n", indent);
116
117     sem_post(&empty);
118     printf("%srEMPTY\n", indent);
119
120     return NULL;
121 }
122
123 #define N 3
124 void testcase_producer_consumer(int ThreadNumber, int inst[2 * N][3]) {
125     pthread_t * p_consumer = new pthread_t[ThreadNumber];
126     pthread_t * p_producer = new pthread_t[ThreadNumber];
127
128     int c_count = 0, p_count = 0;
129
130     printf("testcase_producer_consumer:\n");
131     /* For managed creation of 'ThreadNumber' threads */
132     int st_time = 0;
133     /* Print the first line */
134     int tmp_c = 0, tmp_p = 0;
135     for (int i = 0; i < ThreadNumber; i++) {
136         if (inst[i][0] == PRODUCER) {
137             printf("P%d\t", tmp_p++);
138         } else if (inst[i][0] == CONSUMER) {
139             printf("C%d\t", tmp_c++);
140         }
141     }
142     printf("\n");
143
144     /* Create Producers and Consumers according to $inst*/
145     int rc;
146     string indent("");
147     for (int i = 0; i < ThreadNumber; i++) {
148         if (inst[i][0] == PRODUCER)  {
149             rc = pthread_create(p_producer + p_count, NULL, producer, new
150 arg_struct(p_count, inst[i][1],
151             inst[i][2], indent));
152             if (rc) printf("ERROR\n");
153             p_count++;
154         } else if (inst[i][0] == CONSUMER){
155             rc = pthread_create(p_consumer + c_count, NULL, consumer, new
```

```
156 arg_struct(c_count, inst[i][1],             inst[i][2], indent));
157             if (rc) printf("ERROR\n");
158             c_count++;
159         }
160         indent += '\t';
161     }
162 
163     /* wait until every thread finishes*/
164     for (int i = 0; i < p_count; i++) {
165         pthread_join(p_producer[i], NULL);
166     }
167 
168     for (int i = 0; i < c_count; i++) {
169         pthread_join(p_consumer[i], NULL);
170     }
171 
172     delete[] p_producer;
173     delete[] p_consumer;
174 }
175 
176 int main(int argc, char** argv) {
177     srand((unsigned)time(NULL));
178 
179     memset(buffer, 0, sizeof(int) * BUFFER_SIZE);
180 
181     sem_init(&mutex, 0, 1);
182     sem_init(&empty, 0, BUFFER_SIZE);
183     sem_init(&full, 0, 0);
184 
185     empty_count = BUFFER_SIZE;
186     full_count = 0;
187 
188     /* For managed creation of 2 * N threads */
189     int ThreadNumber = 2 * N ;
190     int st_time = 0;
191     int inst[2 * N][3] = {
192         /* { Consumer or Producer to be create?,
193         When does it start to work after being created?, st_stime += N means it starts
194 N seconcds later than the previous P/C
195         How long does it work after it enters critical zone? } */
196         {CONSUMER, st_time += 0, 2},
197         {CONSUMER, st_time += 1, 2},
198         {CONSUMER, st_time += 2, 2},
199         {PRODUCER, st_time += 3, 2},
200         {PRODUCER, st_time += 4, 2},
201         {PRODUCER, st_time += 5, 2}
202     };
```

```
    testcase_producer_consumer(ThreadNumber, inst);
    st_time = 0;

    int inst2[2 * N][3] = {
        {PRODUCER, st_time += 0, 2},
        {PRODUCER, st_time += 1, 2},
        {CONSUMER, st_time += 2, 2},
        {CONSUMER, st_time += 3, 2},
        {PRODUCER, st_time += 4, 2},
        {CONSUMER, st_time += 5, 2}
    };
    testcase_producer_consumer(ThreadNumber, inst2);
    return 0;
}
```

测试用例的执行输出结果：

```
xyong@ubuntu-xyong:~/work$ gcc producer-consumer.cpp -lpthread -lstdc++
xyong@ubuntu-xyong:~/work$ ./a.out
testcase_producer_consumer:
C0    C1    C2    P0    P1    P2
START
      START
            START
                  START
                  aMUTEX
                  aEMPTY
                  ENTER
                  Prod 0
                  EXIT
                  rMUTEX
                  rFULL
aFULL
aMUTEX
ENTER
                        START
Cons 0
EXIT
                        aMUTEX
                        aEMPTY
                        ENTER
rMUTEX
rEMPTY
                        Prod 1
                        EXIT
                        rMUTEX
```

```
                        rFULL
        aFULL
        aMUTEX
        ENTER
        Cons 1
        EXIT
        rMUTEX
        rEMPTY
                        START
                        aMUTEX
                        aEMPTY
                        ENTER
                        Prod 2
                        EXIT
                        rMUTEX
                        rFULL
              aFULL
              aMUTEX
              ENTER
              Cons 2
              EXIT
              rMUTEX
              rEMPTY

testcase_producer_consumer:
P0    P1    C0    C1    P2    C2
START
aMUTEX
aEMPTY
ENTER
      START
Prod 3
EXIT
rMUTEX
rFULL
      aMUTEX
      aEMPTY
      ENTER
            START
            aFULL
      Prod 4
      EXIT
      rMUTEX
      rFULL
            aMUTEX
            ENTER
                  START
```

```
                    aFULL
              Cons 3
              EXIT
              rMUTEX
              rEMPTY
                    aMUTEX
                    ENTER
                    Cons 4
                    EXIT
                    rMUTEX
                    rEMPTY
                          START
                          aMUTEX
                          aEMPTY
                          ENTER
                          Prod 5
                          EXIT
                          rMUTEX
                          rFULL
                                START
                                aFULL
                                aMUTEX
                                ENTER
                                Cons 5
                                EXIT
                                rMUTEX
                                rEMPTY
 xyong@ubuntu-xyong:~/work$
```

这道题真是又臭又长……

信号量PV操作的伪代码：这个是十分简单了。

```
P() {
    sem--;
    if (sem < 0) {
        Add this thread t to q;
        block(p);
    }
}

V() {
    sem++;
    if (sem <= 0) {
        Remove a thread t from q;
```

```
            wakeup(t);
        }
    }
```

这个实现是否正确？答案是不正确。producer线程的实现中获取`mutex`和`empty`信号量的顺序反了。总的来说，把这两个换一下就好了。

题目中给出的两个测试样例是这样的：

```
int inst[2 * N][3] = {
    /* { Consumer or Producer to be create?,
    When does it start to work after being created?, st_stime += N means it starts N seconcds later than the previous P/C
    How long does it work after it enters critical zone? } */
    {CONSUMER, st_time += 0, 2},
    {CONSUMER, st_time += 1, 2},
    {CONSUMER, st_time += 2, 2},
    {PRODUCER, st_time += 3, 2},
    {PRODUCER, st_time += 4, 2},
    {PRODUCER, st_time += 5, 2}
};

int inst2[2 * N][3] = {
    {PRODUCER, st_time += 0, 2},
    {PRODUCER, st_time += 1, 2},
    {CONSUMER, st_time += 2, 2},
    {CONSUMER, st_time += 3, 2},
    {PRODUCER, st_time += 4, 2},
    {CONSUMER, st_time += 5, 2}
};
```

我给出的测试样例是这样的：

```
int inst2[2 * N][3] = {
    {PRODUCER, st_time += 0, 2},
    {PRODUCER, st_time += 1, 2},
    {PRODUCER, st_time += 2, 2},
    {PRODUCER, st_time += 3, 2},
    {CONSUMER, st_time += 4, 2},
    {CONSUMER, st_time += 5, 2}
};
```

题目中并没有要求Producer和Consumer的数量必须为3个。从理论上来说，只要Producer比Consumer大的个数

在3个（也就是缓冲区的大小）以内，都能正常结束。但是在错误实现中会发生这样的问题：P1-P3生产完之后，P4获得`mutex`后开始在`empty`信号量上等待。但是，由于它占据了`mutex`，因此C1和C2无法进入临界区进行消费，于是也不会对`empty`信号量执行V操作，发生死锁。

六 ucore用户进程（16分）

下面是关于ucore中用户程序的生命历程的代码。请完成下面填空和代码补全。

（1）

在`sh`的命令行上输入`args 1`启动用户程序`args`，则`sh`会调用（**1**）创建新进程并调用（**2**）将`args`加载到该进程的地址空间中。（回答系统调用名称即可）

1. `SYS_fork`
2. `SYS_exec`

这一题使我觉得我应该复习一下ucore里的各种系统调用、实现方法及其作用。

（2）

将`args`从硬盘加载主要由`load_icode`完成，请补全以下代码。

```
    // load_icode - called by sys_exec-->do_execve
1
2   static int
3   load_icode(int fd, int argc, char **kargv) {
4       /* LAB8:EXERCISE2 YOUR CODE HINT:how to load the file with handler fd in to
5   process's memory? how to setup argc/argv?
6       * MACROs or Functions:
7       * mm_create - create a mm
8       * setup_pgdir - setup pgdir in mm
9       * load_icode_read - read raw data content of program file
10      * mm_map - build new vma
11      * pgdir_alloc_page - allocate new memory for TEXT/DATA/BSS/stack parts
12      * lcr3 - update Page Directory Addr Register -- CR3
13      */
14      /* (1) create a new mm for current process
15      * (2) create a new PDT, and mm->pgdir= kernel virtual addr of PDT
16      * (3) copy TEXT/DATA/BSS parts in binary to memory space of process
17      * (3.1) read raw data content in file and resolve elfhdr
18      * (3.2) read raw data content in file and resolve proghdr based on info in elfhdr
19      * (3.3) call mm_map to build vma related to TEXT/DATA
20      * (3.4) callpgdir_alloc_page to allocate page for TEXT/DATA, read contents in file
21      * and copy them into the new allocated pages
22      * (3.5) callpgdir_alloc_page to allocate pages for BSS, memset zero in these pages
```

```
23      * (4) call mm_map to setup user stack, and put parameters into user stack
24      * (5) setup current process's mm, cr3, reset pgidr (using lcr3 MARCO)
25      * (6) setup uargc and uargv in user stacks
26      * (7) setup trapframe for user environment
27      * (8) if up steps failed, you should cleanup the env.
28      */
29      assert(argc >= 0 && argc <= EXEC_MAX_ARG_NUM);
30
31      if (current->mm != NULL) {
32          panic("load_icode: current->mm must be empty.\n");
33      }
34
35      int ret = -E_NO_MEM;
36      struct mm_struct *mm;
37      if ((mm = mm_create()) == NULL) {
38          goto bad_mm;
39      }
40      if (setup_pgdir(mm) != 0) {
41          goto bad_pgdir_cleanup_mm;
42      }
43
44      struct Page *page;
45
46      struct elfhdr __elf, *elf = &__elf;
47      /* 2a */
48      if ((ret = load_icode_read(fd, elf, _(2a)_, 0)) != 0) {
49          goto bad_elf_cleanup_pgdir;
50      }
51
52      if (elf->e_magic != ELF_MAGIC) {
53          ret = -E_INVAL_ELF;
54          goto bad_elf_cleanup_pgdir;
55      }
56
57      struct proghdr __ph, *ph = &__ph;
58      uint32_t vm_flags, perm, phnum;
59      for (phnum = 0; phnum < elf->e_phnum; phnum ++) {
60          off_t phoff = elf->e_phoff + sizeof(struct proghdr) * phnum;
61          if ((ret = load_icode_read(fd, ph, sizeof(struct proghdr), phoff)) != 0) {
62              goto bad_cleanup_mmap;
63          }
64          if (ph->p_type != ELF_PT_LOAD) {
65              /* 2b */
66              _(2b)_
67          }
68          if (ph->p_filesz > ph->p_memsz) {
69              ret = -E_INVAL_ELF;
```

```
70              goto bad_cleanup_mmap;
71          }
72          if (ph->p_filesz == 0) {
73              continue ;
74          }
75          vm_flags = 0, perm = PTE_U;
76          if (ph->p_flags & ELF_PF_X) vm_flags |= VM_EXEC;
77          if (ph->p_flags & ELF_PF_W) vm_flags |= VM_WRITE;
78          if (ph->p_flags & ELF_PF_R) vm_flags |= VM_READ;
79          if (vm_flags & VM_WRITE) perm |= PTE_W;
80          if ((ret = mm_map(mm, ph->p_va, ph->p_memsz, vm_flags, NULL)) != 0) {
81              goto bad_cleanup_mmap;
82          }
83          off_t offset = ph->p_offset;
84          size_t off, size;
85          uintptr_t start = ph->p_va, end, la = ROUNDDOWN(start, PGSIZE);
86
87          ret = -E_NO_MEM;
88
89          end = ph->p_va + ph->p_filesz;
90          while (start < end) {
91              if ((page = pgdir_alloc_page(mm->pgdir, la, perm)) == NULL) {
92                  ret = -E_NO_MEM;
93                  goto bad_cleanup_mmap;
94              }
95              off = start - la, size = PGSIZE - off, la += PGSIZE;
96              if (end < la) {
97                  size -= la - end;
98              }
99              if ((ret = load_icode_read(fd, page2kva(page) + off, size, offset)) != 0)
100 {
101                 goto bad_cleanup_mmap;
102             }
103             start += size, offset += size;
104         }
105         end = ph->p_va + ph->p_memsz;
106
107         if (start < la) {
108             /* ph->p_memsz == ph->p_filesz */
109             if (start == end) {
110                 continue ;
111             }
112             off = start + PGSIZE - la, size = PGSIZE - off;
113             if (end < la) {
114                 size -= la - end;
115             }
116             memset(page2kva(page) + off, 0, size);
```

```
117             start += size;
118             assert((end < la && start == end) || (end >= la && start == la));
119         }
120         while (start < end) {
121             if ((page = pgdir_alloc_page(mm->pgdir, la, perm)) == NULL) {
122                 ret = -E_NO_MEM;
123                 goto bad_cleanup_mmap;
124             }
125             off = start - la, size = PGSIZE - off, la += PGSIZE;
126             if (end < la) {
127                 size -= la - end;
128             }
129             memset(page2kva(page) + off, 0, size);
130             start += size;
131         }
132     }
133     sysfile_close(fd);
134 
135     vm_flags = VM_READ | VM_WRITE | VM_STACK;
136     /* 2c */
137     if ((ret = mm_map(mm, _(2c)_, USTACKSIZE, vm_flags, NULL)) != 0) {
138         goto bad_cleanup_mmap;
139     }
140     assert(pgdir_alloc_page(mm->pgdir, USTACKTOP-PGSIZE , PTE_USER) != NULL);
141     assert(pgdir_alloc_page(mm->pgdir, USTACKTOP-2*PGSIZE , PTE_USER) != NULL);
142     assert(pgdir_alloc_page(mm->pgdir, USTACKTOP-3*PGSIZE , PTE_USER) != NULL);
143     assert(pgdir_alloc_page(mm->pgdir, USTACKTOP-4*PGSIZE , PTE_USER) != NULL);
144 
145     mm_count_inc(mm);
146     current->mm = mm;
147     current->cr3 = PADDR(mm->pgdir);
148     lcr3(PADDR(mm->pgdir));
149 
150     //setup argc, argv
151     uint32_t argv_size=0, i;
152     for (i = 0; i < argc; i ++) {
153         argv_size += strnlen(kargv[i],EXEC_MAX_ARG_LEN + 1)+1;
154     }
155 
156     uintptr_t stacktop = USTACKTOP - (argv_size/sizeof(long)+1)*sizeof(long);
157     char** uargv=(char **)(stacktop - argc * sizeof(char *));
158 
159     argv_size = 0;
160     for (i = 0; i < argc; i ++) {
161         uargv[i] = strcpy((char *)(stacktop + argv_size ), kargv[i]);
162         /* 2d */
163         _(2d)_
```

```
164         }
165
166         stacktop = (uintptr_t)uargv - sizeof(int);
167         /* 2e */
168         *(int *)stacktop = _(2e)_;
169
170         struct trapframe *tf = current->tf;
171         memset(tf, 0, sizeof(struct trapframe));
172         tf->tf_cs = USER_CS;
173         tf->tf_ds = tf->tf_es = tf->tf_ss = USER_DS;
174         tf->tf_esp = stacktop;
175         tf->tf_eip = elf->e_entry;
176         tf->tf_eflags = FL_IF;
177         ret = 0;
178     out:
179         return ret;
180     bad_cleanup_mmap:
181         exit_mmap(mm);
182     bad_elf_cleanup_pgdir:
183         put_pgdir(mm);
184     bad_pgdir_cleanup_mm:
185         mm_destroy(mm);
186     bad_mm:
187         goto out;
    }
```

1. `sizeof(struct elfhdr)`（读一个`elfhdr`大小的文件数据）
2. `goto bad_cleanup_mmap;`（这个很简单：已经设置了`pgdir`和`mm`了，因此如果失败需要清理；而且周围都是跳转到这里）
3. `USTACKTOP - USTACKSIZE`（这段大概是映射用户栈空间，不过我并不是很明白）
4. `argv_size += strnlen(kargv[i], EXEC_MAX_ARG_LEN + 1) + 1;`（得到当前的参数的长度）
5. `argc`（把argc放到栈顶；之所以是栈顶，是因为gcc是从右向左压栈的）

这种默写代码的题目实在是无聊死了。

（3）

完成加载后会从内核态回到用户态，请补全此时的用户栈图示。（假定为写入部分全部初始化为0，注意使用小尾端）

地址	内容
0xb0000000	-
0xaffffffc	00 31 00 00

0xaffffff8	61 72 67 73 // 'args'
0xaffffff4	(3a)
0xaffffff0	(3b)
0xafffffec	(3c)
0xafffffe8	(3d)

此时并不会直接进入main函数，而是执行以下代码，请简述其作用。

```
//////// user/libs/initcode.S //////////
.text
.globl _start
_start:
    movl $0x0, %ebp

    movl (%esp), %ebx
    lea 0x4(%esp), %ecx

    subl $0x20, %esp

    pushl %ecx
    pushl %ebx

    call umain
1: jmp 1b

////////// user/libs/umain.c ////////////
#include <ulib.h>
#include <unistd.h>
#include <file.h>
#include <stat.h>
int main(int argc, char *argv[]);
static int
initfd(int fd2, const char *path, uint32_t open_flags) {
    int fd1, ret;
    if ((fd1 = open(path, open_flags)) < 0) {
        return fd1;
    }
    if (fd1 != fd2) {
        close(fd2);
        ret = dup2(fd1, fd2);
        close(fd1);

```

```
        }
35      return ret;
36  }
37
38  void
39  umain(int argc, char *argv[]) {
40      int fd;
41      if ((fd = initfd(0, "stdin:", O_RDONLY)) < 0) {
42          warn("open <stdin> failed: %e.\n", fd);
43      }
44      if ((fd = initfd(1, "stdout:", O_WRONLY)) < 0) {
45          warn("open <stdout> failed: %e.\n", fd);
46      }
47      int ret = main(argc, argv);
48      exit(ret);
49  }
```

#OS

□ 评论 □ 分享到

下一篇

《操作系统》ucore实验四"内核线程管理"报告

上一篇

《操作系统》2017年期末考试分析

标签云

A.Munday Blogging C.Marlowe CSP Codeforces Codeforces Contest Counseling Cryptography D.Drayton Deep Learning Depth-first Search DigitCircuit E. Vere E.Spencer Essay Github GoldenTreasury H.Constable J.Donne J.Lyly J.Sylvester J.Webster Leetcode Leetcode Contest Lyric Machine Learning Machine Translation Natural Language Processing OS OSTEP OldBlog P.Sidney Paper Paul Simon PhysicsExperiment Psychology Quality Estimation R.Barnfield Raspberry Pi Reading Report S.Daniel SGU Sonnet Spokes SystemAnalysis&Control T.Dekker T.Heywood T.Lodge T.Nashe T.Wyatt THUMT TensorFlow Translation Tree USACO W.Alexander W.Drummond W.Shakespeare alg:Array alg:Automata alg:Backtracking alg:Binary Indexed Tree alg:Binary Search alg:Binary Search Tree alg:Bit Manipulation alg:Breadth-first Search alg:Breadth-firth Search alg:Brute Force alg:Depth-first Search alg:Divide and Conquer alg:Dynamic Programming alg:Graph alg:Greedy alg:Hash Table alg:Heap alg:In-Order Traversal alg:Linked List alg:Math alg:Meet in the Middle alg:Minimax alg:Priority Queue alg:Queue alg:Random alg:Recursion alg:Recursive alg:Rejection Sampling alg:Reservoir Sampling alg:Set

alg:Sort alg:Stack alg:String alg:Topological Sort alg:Tree alg:Trie alg:Two Pointers alg:Union-find Forest artist:Ceremony artist:Cruel Hand artist:Have Heart artist:Johnny Cash artist:Touche Amore artist:Wir Sind Helden ucore 付勇林 卞之琳 屠岸 戴镏龄 曹明伦 朱生豪 李霁野 杨熙龄 林天斗 梁宗岱 梁葆成 袁广达 郭沫若 黄新渠

归档

十二月 2018 (18)
十一月 2018 (22)
十月 2018 (31)
九月 2018 (63)
八月 2018 (69)
七月 2018 (15)
六月 2018 (3)
五月 2018 (19)
四月 2018 (27)
二月 2018 (2)
一月 2018 (7)
十二月 2017 (9)
七月 2017 (11)
四月 2017 (1)
三月 2017 (2)
一月 2017 (1)

近期文章

为什么sigmoid和softmax需要和cross entropy一起计算
Leetcode 956. Tallest Billboard（DP）
Leetcode 955. Delete Columns to Make Sorted II（贪心）
Leetcode 954. Array of Doubled Pairs（贪心）
Leetcode 953. Verifying an Alien Dictionary（hash），及周赛（114）总结

友情链接

wenj
ssh