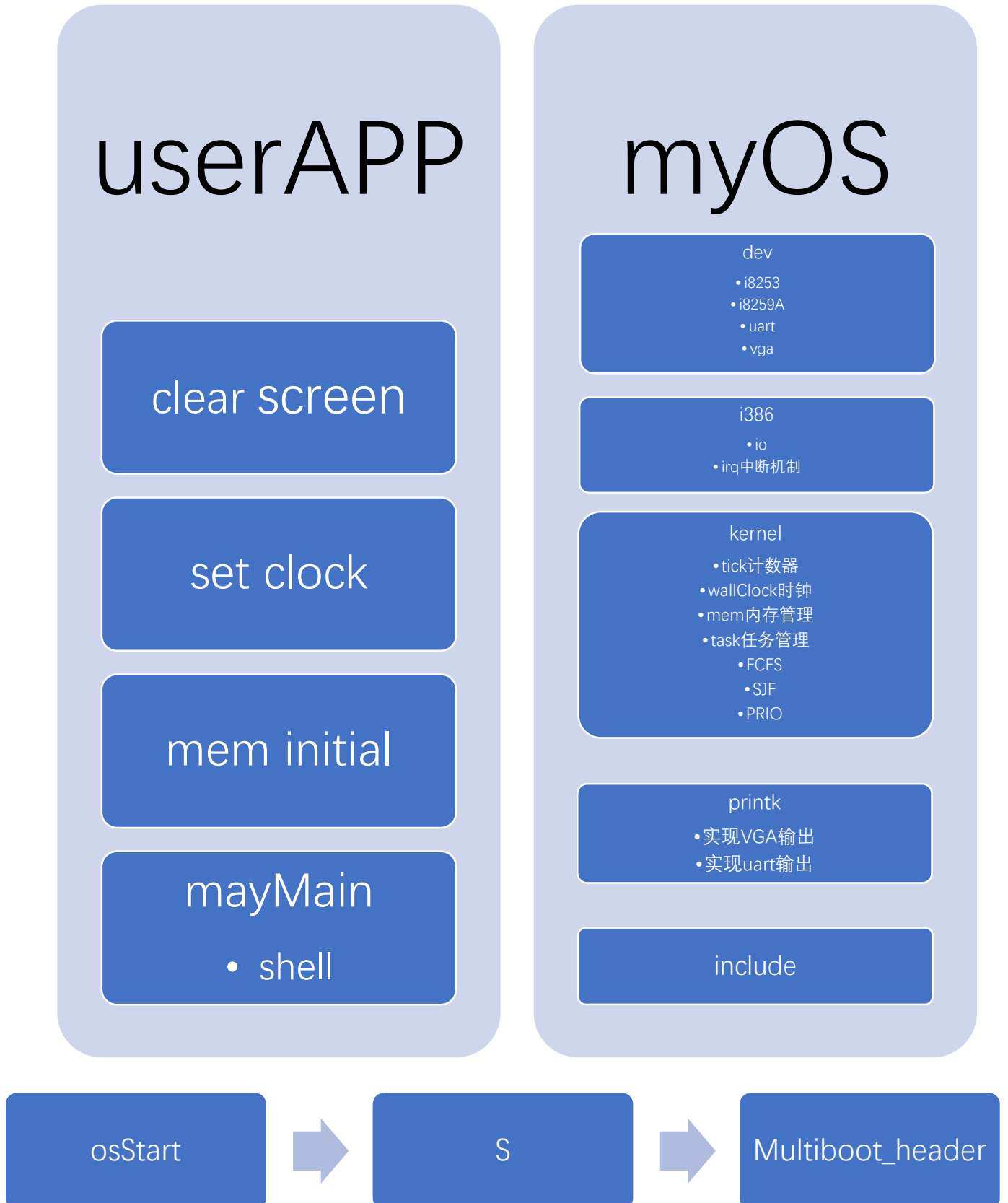


操作系统实验 5 报告

软件框图



主流程及其实现

在上次实验的基础上，增加下列内容。



主要功能模块及其实现

本次实验中要实现的功能模块分别按五个部分实现

任务参数设置

- 数据结构
- 初始值
- 整体赋值
- 成员赋值
- 初始化
- 获取



待到达任务管理

- 链表数据结构
- 初始化
- 入队
- 出队



调度算法实现

- 参考FCFS
- 完成SJF
- 完成PRIO



调度算法管理

- 调度算法选择
- 封装调度相关函数



测试任务管理

- FCFS测试
- SJF测试
- PRIO测试

源代码说明

在本次实验的根目录下，除了配置了 makefile 与 DS 文件，还有 4 个文件夹：multibootheader、myOS、output、userApp。multibootheader 中放置了有关 multibootheader 协议的 S 文件。在 output 中，原本为空文件夹，用来输出编译后生成的文件。在该文件夹中的目录结构与根目录相似，主要是为了对每个文件输出时不产生混淆。在 userApp 中存放的是 main 文件、memTestCase 文件、userTasks 文件与该文件夹下的 makefile 文件，本次实验中的 userTasks 是提供的测试 Task 相关功能文件。在 myOS 文件夹中存放了本次实验的主要源文件。直接存放在此该目录下的文件有 DS 文件、该文件夹下的 makefile 文件、链接器文件、一个配置地址的 S 文件和一个包含了 main 函数的与 multibootheader 中的 S 文件相关联的 C 文件。这个文件也是从汇编到 C 的转变。在该目录下则有 6 个实现相关功能的文件夹：dev、i386、printk、kernel、lib、include。每个文件夹下都有一个相应的 makefile 文件以及实现功能的 C 语言源文件。本次实验中在 include 中添加了 task_arr.h、task_sched.h、taskPara.h 文件，主要功能是对应的 C 文件中的函数声明与相关宏定义。此外，我在 include 中自行添加了颜色宏的头文件，方便用宏定义写入 myPrint 的颜色参数。kernel 文件夹中增加 task_arr.c、task_sched.c、taskPara.c 源代码文件以实现任务调度算法的相关功能，以及 task_sched 文件夹存放 FCFS、SJF、PRIO 等调度算法的源代码文件。每个子目录下的 makefile 文件的输出目录都是在 output 中的同名目录。下面展示相关源代码。

下面展示本次实验的框架中所要求实现的部分。

```

1  #include "../include/tcb.h"
2  #include "../include/taskPara.h"
3  #include "../include/task_sched.h"
4
5  //extern void initLeftExeTime_sjf(myTCB* tsk);
6
7  // may modified during scheduler_init
8  tskPara defaultTskPara = {
9      .priority = 0,
10     .exeTime = MAX_EXETIME,
11     .arrTime = 0,
12     .schedPolicy = SCHED_UNDEF
13 }; //task设计调度的一些参数的默认值
14
15 void _setTskPara(myTCB* task, tskPara* para)
16 {
17     if (para == NULL)
18         task->para = defaultTskPara;
19     else
20     {
21         task->para = *para;
22         switch (para->schedPolicy)
23         {
24             case SCHEDULER_FCFS:
25                 task->para.schedPolicy = SCHED_FCFS;
26                 break;
27             case SCHEDULER_SJF:
28                 task->para.schedPolicy = SCHED_SJF;
29                 break;
30             case SCHEDULER_PRIORITY:
31                 task->para.schedPolicy = SCHED_PRIO;
32                 break;
33             default:
34                 task->para.schedPolicy = SCHED_UNDEF;
35                 break;
36         }
37     }
38 }
39

```

task_para.c 代码部分 1, 关于参数的默认值以及对参数的赋值实现

```

40 void initTskPara(tskPara* buffer)
41 {
42     buffer->arrTime = 0;
43     buffer->exeTime = MAX_EXETIME;
44     buffer->priority = 0;
45     buffer->schedPolicy = SCHED_UNDEF;
46 }
47
48 void setTskPara(unsigned int option, unsigned int value, tskPara* buffer)
49 {
50     switch (option)
51     {
52     case ARRTIME:
53         buffer->arrTime = value;
54         break;
55     case EXETIME:
56         buffer->exeTime = value;
57         break;
58     case PRIORITY:
59         buffer->priority = value;
60         break;
61     case SCHED_POLICY:
62         buffer->schedPolicy = value;
63         break;
64     default:
65         break;
66     }
67 }

```

task_para.c 代码部分 2，主要实现初始化参数和对参数成员的指定赋值。

```

69 void getTskPara(unsigned option, unsigned int* para, tskPara* buffer)
70 {
71     switch (option)
72     {
73     case ARRTIME:
74         *para = currentTsk->para.arrTime;
75         break;
76     case EXETIME:
77         *para = currentTsk->para.exeTime;
78         break;
79     case PRIORITY:
80         *para = currentTsk->para.priority;
81         break;
82     case SCHED_POLICY:
83         *para = currentTsk->para.schedPolicy;
84         break;
85     default:
86         break;
87     }
88 }

```

task_para.c 代码部分 3，获得指定参数功能的实现

```

41  /* arrTime: small --> big */
42  void ArrListEnqueue(myTCB* tsk)
43  {
44      arrNode* arrnode = tcb2Arr(tsk);
45      arrnode->theTCB = tsk;
46      arrnode->arrTime = tsk->para.arrTime;
47      dLinkedList* head = &arrList;
48      dLinkedList* plist = &arrList;
49      arrNode* parr;
50      while (1)
51      {
52          //移动节点
53          plist = dLinkListFirstNode(plist);
54          parr = (arrNode*)plist;
55          //当p回到队首/到达时间小于当前节点时
56          if (plist == head || arrnode->arrTime <= parr->arrTime)
57          {
58              dLinkInsertBefore(head, plist, (dLinkedList*)arrnode);
59              break;
60          }
61      }
62  }

```

task_arr.c 代码，实现创建节点并按到达时间加入队列

```

28  void setSysScheduler(unsigned int what)
29  {
30      switch (what)
31      {
32          case SCHEDULER_FCFS:
33              sysScheduler = &scheduler_FCFS;
34              break;
35          case SCHEDULER_PRIORITY:
36              sysScheduler = &scheduler_PRIO;
37              break;
38          //case SCHEDULER_PRIORITY0:
39          // sysScheduler = &scheduler_PRIO0;
40          // break;
41          case SCHEDULER_SJF:
42              sysScheduler = &scheduler_SJF;
43              break;
44          //case SCHEDULER_RR:
45          // sysScheduler = &scheduler_RR;
46          // break;
47          //case SCHEDULER_MQ:
48          // sysScheduler = &scheduler_MQ;
49          // break;
50          //case SCHEDULER_FMQ:
51          // sysScheduler = &scheduler_FMQ;
52          // break;
53          default:
54              sysScheduler = &scheduler_FCFS;
55              break;
56      }
57  }
58

```

task_sched.c 代码部分 1，主要实现调度器的选择。未实现的调度算法已经被注释。

```

59 //由于RR涉及到设置时间片大小 所以需要如下的两个函数getSysSchedulerPara setSysSchedulerPar
60 //其他调度器则不需要
61 void getSysSchedulerPara(unsigned int who, unsigned int* para)
62 {
63     switch (who)
64     {
65     case SCHED_RR_SLICE:
66         *para = defaultSlice;
67         break;
68     case SCHED_RT_RR_SLICE:
69         *para = defaultRtSlice;
70         break;
71     default:;
72     }
73 }
74
75 void setSysSchedulerPara(unsigned int who, unsigned int para)
76 {
77     switch (who)
78     {
79     case SCHED_RR_SLICE:
80         defaultSlice = para;
81         break;
82     case SCHED_RT_RR_SLICE:
83         defaultRtSlice = para;
84         break;
85     default:;
86     }
87 }

```

task_sched.c 代码部分 2，主要实现了调度器参数获取和赋值


```

89 //每一个调度器中集成了几个函数 参考./task_sched/task_fifo.c中的scheduler scheduler_FCFS的实
90 //实现以下的nextTsk enqueueTsk dequeueTsk schedulerInit scheduler_tick
91 myTCB* nextTsk(void)
92 {
93     if (sysScheduler->nextTsk_func)
94         return sysScheduler->nextTsk_func();
95     else
96         return (myTCB*)NULL;
97 }
98
99 void enqueueTsk(myTCB* tsk)
100 {
101     if (sysScheduler->enqueueTsk_func)
102         sysScheduler->enqueueTsk_func(tsk);
103 }
104
105 void dequeueTsk(myTCB* tsk)
106 {
107     if (sysScheduler->dequeueTsk_func)
108         sysScheduler->dequeueTsk_func(tsk);
109 }
110
111 void createTsk_hook(myTCB* created)
112 {
113     if (sysScheduler->createTsk_hook)
114         sysScheduler->createTsk_hook(created);
115 }
116
117 extern void scheduler_hook_main(void);
118
119 void schedulerInit(void)
120 {
121     scheduler_hook_main();
122     if (sysScheduler->schedulerInit_func)
123         sysScheduler->schedulerInit_func();
124 }
125

```

task_sched.c 代码部分 3， 主要实现调度器的内部函数的封装。这有利于对不同算法的统一管理

```

126 void scheduler_tick(void)
127 {
128     if (sysScheduler->tick_hook)
129         sysScheduler->createTsk_hook(currentTsk);
130 }
131
132 void schedule(void)
133 {
134     static int idle_times = 0;
135     myTCB* prevTsk, * nextTsk;
136     disable_interrupt();
137     prevTsk = currentTsk;
138     nextTsk = sysScheduler->nextTsk_func();
139     currentTsk = nextTsk;
140     context_switch(prevTsk, nextTsk);
141     enable_interrupt();
142 }

```

task_sched.c 代码部分 4， 主要实现调度过程。

```

1
2 myTCB* nextSJFTsk(void)
3 {
4     dLinkedList* head = (dLinkedList*)rqSJF;
5     dLinkedList* plist = head;
6     myTCB* ptcb = (myTCB*)head;
7     do
8     {
9         //节点移动
10        plist = dLinkedListFirstNode(plist);
11        //检测到更短执行时间
12        if (((myTCB*)plist)->para.exeTime < ptcb->para.exeTime)
13        {
14            //更新最短执行时间
15            ptcb = (myTCB*)plist;
16        }
17    } while (plist != head);
18    return ptcb;
19 }

```

task_SJF 代码部分，由于大多数代码与 FCFS 代码文件所给相同，故主要展示不同的 next 函数的实现，PRIO 算法同。不同调度算法主要是对 next 函数的不同实现。主要是检测当前任务队列并依次执行时间并进行比较，最终找到最短执行时间

```

11 myTCB* nextPRIOTsk(void)
12 {
13     dLinkedList* head = (dLinkedList*)rqPRIO;
14     dLinkedList* plist = head;
15     myTCB* ptcb = (myTCB*)initTskPara;
16     int i = 0;
17     do
18     {
19         //节点移动
20        plist = dLinkedListFirstNode(plist);
21        //首次移动则立即修改ptcb
22        if (i < 1)
23        {
24            i++;
25            ptcb = (myTCB*)plist;
26            continue;
27        }
28        //检测到更高优先级
29        if (((myTCB*)plist)->para.priority > ptcb->para.priority)
30        {
31            //更新最高优先级
32            ptcb = (myTCB*)plist;
33        }
34    } while (plist != head);
35    return ptcb;
36 }

```

原理同上一算法，检测的成员不同。但是此算法需要避开 initsk 和 idletsk 的优先级对判断的干扰，故需要绕开此节点的 prio 检测。

```

1  #include "../myOS/userInterface.h" //interface from kernel
2  #include "shell.h"
3  #include "memTestCase.h"
4  #define SCHED_INSTANCE SCHEDULER_FCFS //改变宏定义实现算法选择
5

```

main 代码部分 1，本次实验为了任意在测试用例间的切换，我选择用宏定义的方法来产生对应不同的编译内容。

```

29  void myTSK0(void)
30  {
31  #if(SCHED_INSTANCE == SCHEDULER_FCFS) //FCFS
32      int j = 1;
33      while (j <= 4)
34      {
35          myPrintf(0x7, "myTSK0::%d \n", j);
36          busy_n_ms(120);
37          j++;
38      }
39  #elif(SCHED_INSTANCE == SCHEDULER_SJF) //SJF
40      int j = 1;
41      while (j <= 4)
42      {
43          myPrintf(0x7, "myTSK0::%d \n", j);
44          busy_n_ms(120);
45          j++;
46      }
47  #elif(SCHED_INSTANCE == SCHEDULER_PRIORITY) //PRIO
48      int j = 1;
49      myPrintf(0x7, "priority of myTSK0 : 3\n");
50      while (j <= 4)
51      {
52          myPrintf(0x7, "myTSK0::%d \n", j);
53          busy_n_ms(120);
54          j++;
55      }
56  #endif
57      tskEnd(); //the task is end
58  }

```

main 代码部分 2，此部分为 task0 在不同调度算法下的内容

```

60 void myTSK1(void)
61 {
62     #if(SCHED_INSTANCE == SCHEDULER_FCFS)    //FCFS
63         int j = 1;
64         while (j <= 4)
65         {
66             myPrintf(0x7, "myTSK1::%d    \n", j);
67             busy_n_ms(120);
68             j++;
69         }
70     #elif(SCHED_INSTANCE == SCHEDULER_SJF)    //SJF
71         int j = 1;
72         while (j <= 8)
73         {
74             myPrintf(0x7, "myTSK1::%d    \n", j);
75             busy_n_ms(120);
76             j++;
77         }
78     #elif(SCHED_INSTANCE == SCHEDULER_PRIORITY)    //PRIO
79         int j = 1;
80         myPrintf(0x7, "priority of myTSK0 : 5\n");
81         while (j <= 4)
82         {
83             myPrintf(0x7, "myTSK1::%d    \n", j);
84             busy_n_ms(120);
85             j++;
86         }
87     #endif
88     tskEnd();    //the task is end
89 }

```

main 代码部分 3，此部分为 task1 在不同调度算法下的内容

```

91 void myTSK2(void)
92 {
93     #if(SCHED_INSTANCE == SCHEDULER_FCFS)    //FCFS
94         int j = 1;
95         while (j <= 4)
96         {
97             myPrintf(0x7, "myTSK2::%d    \n", j);
98             busy_n_ms(120);
99             j++;
100         }
101     #elif(SCHED_INSTANCE == SCHEDULER_SJF)    //SJF
102         int j = 1;
103         while (j <= 6)
104         {
105             myPrintf(0x7, "myTSK2::%d    \n", j);
106             busy_n_ms(120);
107             j++;
108         }
109     #elif(SCHED_INSTANCE == SCHEDULER_PRIORITY)    //PRIO
110         int j = 1;
111         myPrintf(0x7, "priority of myTSK0 : 4\n");
112         while (j <= 4)
113         {
114             myPrintf(0x7, "myTSK2::%d    \n", j);
115             busy_n_ms(120);
116             j++;
117         }
118     #endif
119     tskEnd();    //the task is end
120 }

```

main 代码部分 4，此部分为 task2 在不同调度算法下的内容

```

122 void testScheduler(void)
123 {
124     //FCFS or RR or SJF or PRIORITY0
125     tskPara tskParas[4];
126     int i;
127     for (i = 0; i < 4; i++)
128         initTskPara(&tskParas[i]);
129     //创建task 并且根据调度算法 设置task的参数
130     #if (SCHED_INSTANCE == SCHEDULER_FCFS) //FCFS
131         setTskPara(ARRTIME, 50, &tskParas[0]);
132         createTsk(myTSK0, &tskParas[0]);
133         setTskPara(ARRTIME, 100, &tskParas[1]);
134         createTsk(myTSK1, &tskParas[1]);
135         setTskPara(ARRTIME, 0, &tskParas[2]);
136         createTsk(myTSK2, &tskParas[2]);
137     #elif (SCHED_INSTANCE == SCHEDULER_SJF) //SJF
138         setTskPara(EXETIME, 100, &tskParas[0]);
139         setTskPara(ARRTIME, 0, &tskParas[0]);
140         createTsk(myTSK0, &tskParas[0]);
141         setTskPara(EXETIME, 200, &tskParas[1]);
142         setTskPara(ARRTIME, 0, &tskParas[1]);
143         createTsk(myTSK1, &tskParas[1]);
144         setTskPara(EXETIME, 150, &tskParas[2]);
145         setTskPara(ARRTIME, 0, &tskParas[2]);
146         createTsk(myTSK2, &tskParas[2]);
147     #elif (SCHED_INSTANCE == SCHEDULER_PRIORITY) //PRIO
148         setTskPara(PRIORITY, 3, &tskParas[0]);
149         setTskPara(ARRTIME, 0, &tskParas[0]);
150         createTsk(myTSK0, &tskParas[0]);
151         setTskPara(PRIORITY, 5, &tskParas[1]);
152         setTskPara(ARRTIME, 0, &tskParas[1]);
153         createTsk(myTSK1, &tskParas[1]);
154         setTskPara(PRIORITY, 4, &tskParas[2]);
155         setTskPara(ARRTIME, 0, &tskParas[2]);
156         createTsk(myTSK2, &tskParas[2]);
157     #elif //ELSE
158         return;
159     #endif

```

main 代码部分 5，此部分为不同调度算法下的各个 task 参数设置

```

160     initShell();
161     memTestCaseInit();
162     setTskPara(ARRTIME, 120, &tskParas[3]);
163     setTskPara(EXETIME, 1000, &tskParas[3]);
164     setTskPara(PRIORITY, 2, &tskParas[3]);
165     createTsk(startShell, &tskParas[3]); // startShell();
166 }
167

```

main 代码部分 4，此部分为 Shell 的参数设置与任务创建

代码布局说明

所有的引导模块将按页（4KB）边界对齐，物理内存地址从 1M 处开始。

编译过程说明：

通过指令 make 完成编译，在 output 目录中的对应目录中分别输出了与根目录下对应文件相同文件。随后我将三种调度算法下编译的可执行文件分别再次命名，并且删除了中间文件，得到了三个 elf 文件。

运行和运行结果说明：

在 Ubuntu 中通过 QEMU 启动已经编译生成的 bin 文件，得到 Linux 的图形化界面运行结果，然后再通过 Ubuntu 启动一个交互界面，用于输入与输出。

```

MemStart: 100000
MemSize: 7f00000
_end: 10c910
dPartition(start=0x10c910, size=0x7ef36f0, firstFreeStart=0x10c980)
EMB(start=0x10c918, size=0x68, nextStart=0x0)
EMB(start=0x10c980, size=0x7ef3680, nextStart=0x0)
dPartition(start=0x10c910, size=0x7ef36f0, firstFreeStart=0x10c918)
EMB(start=0x10c918, size=0x7ef36e8, nextStart=0x0)
START MULTITASKING.....
*****INIT START

*****INIT END

myTSK2::1
myTSK2::2
myTSK2::3
myTSK2::4
*****IDLE LOOP.....0.
myTSK0::1
myTSK0::2
myTSK0::3
myTSK0::4
myTSK1::1
myTSK1::2
myTSK1::3
myTSK1::4
waiting for your command >:

```

如图是 FCFS 算法在 screen 上的运行结果。为了便于其他算法的实现，我先实现了 FCFS 算法验证非调度器文件的正确性。在上面的 main 代码中我已经修改了 tsk 的相关内容以避免过长的内容不便于查看完整结果。可以看到执行顺序是优先执行已经就绪的任务 2，然后执行完毕时判断任务 0 与任务 1 的到达顺序，便优先执行任务 0，符合目标。

```

MemStart: 100000
MemSize: 7f00000
_end: 10c950
dPartition(start=0x10c950, size=0x7ef36b0, firstFreeStart=0x10c9c0)
EMB(start=0x10c958, size=0x68, nextStart=0x0)
EMB(start=0x10c9c0, size=0x7ef3640, nextStart=0x0)
dPartition(start=0x10c950, size=0x7ef36b0, firstFreeStart=0x10c958)
EMB(start=0x10c958, size=0x7ef36a8, nextStart=0x0)
START MULTITASKING.....
*****INIT START

*****INIT END

myTSK0::1
myTSK0::2
myTSK0::3
myTSK0::4
myTSK2::1
myTSK2::2
myTSK2::3
myTSK2::4
myTSK2::5
myTSK2::6
myTSK1::1
myTSK1::2
myTSK1::3
myTSK1::4
myTSK1::5
myTSK1::6
myTSK1::7
myTSK1::8
*****IDLE LOOP.....0.
waiting for your command >:

```

如图是 SJF 算法在 screen 上的运行结果。在 main 中，为了显著体现出 next 的实现，我将其他算法中的任务到达时间统一修改为了 0，以便同时比较。除此之外，每个任务的执行时间也与其执行打印循环次数成正相关，可以看到，依次执行了打印次数递增的任务。


```

MemStart: 100000
MemSize: 7f00000
_end: 10c9d0
dPartition(start=0x10c9d0, size=0x7ef3630, firstFreeStart=0x10ca40)
EMB(start=0x10c9d8, size=0x68, nextStart=0x0)
EMB(start=0x10ca40, size=0x7ef35c0, nextStart=0x0)
dPartition(start=0x10c9d0, size=0x7ef3630, firstFreeStart=0x10c9d8)
EMB(start=0x10c9d8, size=0x7ef3628, nextStart=0x0)
START MULTITASKING.....
*****INIT START

*****INIT END

priority of myTSK0 : 5
myTSK1::1
myTSK1::2
myTSK1::3
myTSK1::4
priority of myTSK0 : 4
myTSK2::1
myTSK2::2
myTSK2::3
myTSK2::4
priority of myTSK0 : 3
myTSK0::1
myTSK0::2
myTSK0::3
myTSK0::4
*****IDLE LOOP.....0.
waiting for your command >:

```

如图是 PRIO 算法在 screen 上的运行结果。每个任务的优先级在任务开头先打印一次，可以看到，依次执行了打印优先级递减的任务。

遇到的问题和解决方案：

1. 不能理解调度器的具体实现方式

网络查询并了解相关信息。

2. 不理解链表头文件的用法

咨询朋友后自行了解了指针的强行转换原理并成功应用。