

OS Lab5 实验报告

实验目标

1. 在Lab4的基础上，实现上下文切换和 FCFS 算法进程调度，并通过进程实现 shell

源代码说明

由于助教的命名规则太过混乱，为了方便，我预先将大部分的变量和函数命名中的缩写改为了完整拼写
说明位于每个函数最后

```
1 //初始化就绪队列（需要填写）
2 void rqFCFSInit(myTCB* idleTask) { //对rqFCFS进行初始化处理
3     rqFCFS.idleTask = idleTask;
4 } //idleTask不入队
5
6 //如果就绪队列为空，返回True（需要填写）
7 int rqFCFSIsEmpty(void) { //当head和tail均为NULL时，rqFCFS为空
8     if (rqFCFS.head == NULL && rqFCFS.tail == NULL) {
9         return 1;
10    }
11    return 0;
12 }
13
14 //获取就绪队列的头结点信息，并返回（需要填写）
15 myTCB* nextFCFSTask(void) { //获取下一个Task
16     if (!rqFCFS.head) {
17         return &tcbPool[1];
18     }
19     return rqFCFS.head;
20 } //若队列为空（如运行0号进程时），返回1号进程的地址
21
22 //将一个未在就绪队列中的TCB加入到就绪队列中（需要填写）
23 void taskEnqueueFCFS(myTCB* task) { //将task入队rqFCFS
24     if (rqFCFSIsEmpty()) {
25         rqFCFS.head = task;
26         rqFCFS.tail = task;
27     }
28     else {
29         rqFCFS.tail->nextTCB = task;
30         rqFCFS.tail = task;
31     }
32 }
33
34 //将就绪队列中的TCB移除（需要填写）
35 void taskDequeueFCFS(myTCB* task) { //rqFCFS出队
36     if (rqFCFS.head == rqFCFS.tail) {
37         rqFCFS.head = NULL;
38         rqFCFS.tail = NULL;
39     }
40     rqFCFS.head = rqFCFS.head->nextTCB;
```

```

41 }
42
43 //以taskBody为参数在进程池中创建一个进程，并调用taskStart函数，将其加入就绪队列（需要填写）
44 int createTask(void (*taskBody)(void)) { //在进程池中创建一个进程，并把该进程加入到rqFCFS队列中
45     myTCB* task = firstFreeTask;
46     task->task_entrance = taskBody;
47     stack_init(&(task->stackTop), taskBody);
48     task->TASK_State = TASK_WAIT;
49     taskStart(task);
50     for (int i = 0; i < TASK_NUM; ++i) {
51         if (tcbPool[i].TASK_State == TASK_NONE) {
52             firstFreeTask = &tcbPool[i];
53             break;
54         }
55     }
56     return task->TASK_ID;
57 } //在firstFreeTask地址创建新进程，初始化栈空间，编辑进入函数，并入队
58 //遍历地址空间，找到第一个空闲的进程块，以其地址更新firstFreeTask
59 //（返回值没有用到？）
60
61 //以taskIndex为关键字，在进程池中寻找并销毁taskIndex对应的进程（需要填写）
62 void destroyTask(unsigned long taskIndex) { //在进程中寻找TASK_ID为taskIndex的进程，并销毁该进程
63     int i = 0;
64     for (i = 0; i < TASK_NUM; ++i) {
65         if (tcbPool[i].TASK_ID == taskIndex) {
66             break;
67         }
68     }
69     myTCB* task = &tcbPool[i];
70     task->TASK_State = TASK_NONE;
71 } //遍历地址空间，找到第一个ID相同的进程块，将其状态设为空闲

```

除此之外，我将 `startMultitask()` 函数中的 `currentTask = nextFCFSTask();` 改为了 `currentTask = &tcbPool[0];`，因为此时系统将要进入0号进程，即 `idleTask`，而如果 `nextFCFSTask()` 返回的结果是0号进程的地址，则进入0号进程后，由于0号进程没有出队，系统将循环调度进入0号进程。因此 `nextFCFSTask()` 返回的结果应为1号进程的地址

（在在报告的时候想到，也许助教的意思是，0号进程仅被创建，而没有进入队列，系统也不会进入0号进程，而是直接从1号进程开始运行，那逻辑就说得通了）

```

1 void startMultitask(void) {
2     BspContext = BspContextBase + STACK_SIZE - 1;
3     prevTASK_StackPtr = &BspContext;
4     //currentTask = nextFCFSTask();
5     currentTask = &tcbPool[0];
6     nextTASK_StackPtr = currentTask->stackTop;
7     CTX_SW(prevTASK_StackPtr, nextTASK_StackPtr);
8 }

```

思考题

在上下文切换的现场维护中，`pushf` 和 `popf` 对应，`pusha` 和 `popa` 对应，`call` 和 `ret` 对应，但是为什么 `CTS_SW()` 函数中只有 `ret` 而没有 `call` 呢？

因为系统内核在调用 `CTS_SW()` 函数的时候，实际上编译器编译出的代码中使用的是 `call` 指令，因此 `CTS_SW` 函数中需要一个 `ret` 指令来返回调用的位置，而 `CTS_SW()` 函数本身不需要使用 `call` 指令

谈一谈你对 `stack_init()` 函数的理解。

`myTCB` 结构体创建时即创建了一个大小为 `STACK_SIZE` 的栈空间，在 `TaskManagerInit()` 函数中，结构体中的 `stackTop` 的值被初始化为该栈空间的最高地址。`stack_init()` 函数从该栈空间最高地址开始赋初值，同时减小 `stackTop` 的值，形成一个栈结构，内容包括进程入口函数地址、寄存器初值等。在 `CTS_SW()` 函数中，寄存器内容被压栈到该栈空间中，栈指针被修改为下一个进程的 `stackTop`，并将新的进程的栈空间中的内容（若为第一次运行该进程，则内容为 `stack_init()` 函数中的值）赋给寄存器

`myTCB` 结构体定义中的 `stack[STACK_SIZE]` 的作用是什么？`BspContextBase[STACK_SIZE]` 的作用又是什么？

`stack[STACK_SIZE]` 即为该进程对应的栈空间，其中存储进程的入口函数地址、寄存器内容等；

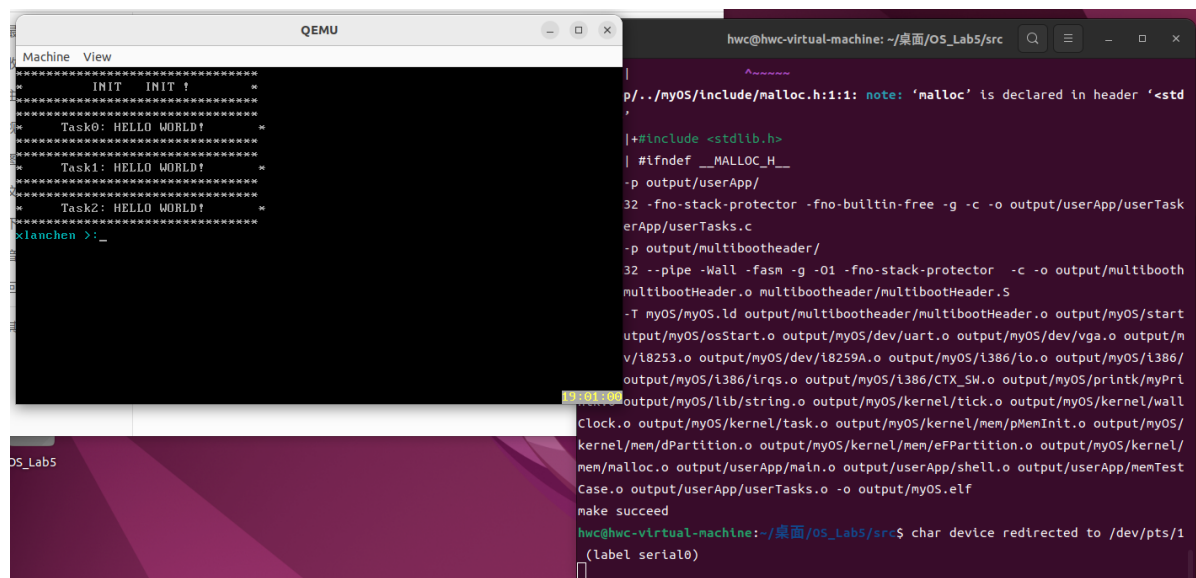
`BspContextBase[STACK_SIZE]` 用于存储进入多任务调度模式前的寄存器内容

`prevTSK_StackPtr` 是一级指针还是二级指针？为什么？

是二级指针，因为 `CTS_SW()` 函数中，需要将当前 `esp` 寄存器中的值（栈指针）保存当前进程的结构体的 `stackTop` 中，所以需要使用二级指针

实验运行结果

编译运行



```
Machine View
=====
INIT INIT !
=====
Task0: HELLO WORLD!
=====
Task1: HELLO WORLD!
=====
Task2: HELLO WORLD!
=====
xlanchen >:

hwc@hwc-virtual-machine: ~/桌面/OS_Lab5/src
p/./myOS/include/malloc.h:1:1: note: 'malloc' is declared in header '<std
|+#include <stdlib.h>
| #ifndef __MALLOC_H__
-p output/userApp/
32 -fno-stack-protector -fno-builtin-free -g -c -o output/userApp/userTask
erApp/userTasks.c
-p output/multibootheader/
32 -pipe -Wall -fasm -g -O1 -fno-stack-protector -c -o output/multibooth
multibootHeader.o multibootheader/multibootHeader.S
-T myOS/myOS.ld output/multibootheader/multibootHeader.o output/myOS/start
utput/myOS/osStart.o output/myOS/dev/uart.o output/myOS/dev/vga.o output/m
v/i8253.o output/myOS/dev/i8259A.o output/myOS/i386/io.o output/myOS/i386/
output/myOS/i386/irqs.o output/myOS/i386/CTX_SW.o output/myOS/printk/myPri
output/myOS/lib/string.o output/myOS/kernel/tick.o output/myOS/kernel/wall
Clock.o output/myOS/kernel/task.o output/myOS/kernel/men/pMemInit.o output/myOS/
kernel/men/dPartition.o output/myOS/kernel/men/eFPartition.o output/myOS/kernel/
men/malloc.o output/userApp/main.o output/userApp/shell.o output/userApp/memTest
Case.o output/userApp/userTasks.o -o output/myOS.elf
make succeed
hwc@hwc-virtual-machine:~/桌面/OS_Lab5/src$ char device redirected to /dev/pts/1
(label serial0)
```

进入screen使用 `shell`，输入 `cmd`

```
Machine View
=====
Task0: HELLO WORLD!
=====
Task1: HELLO WORLD!
=====
Task2: HELLO WORLD!
=====
xlanchen >:cmd
list all registered commands:
command name: description
testeFP: Init a ePatiition. Alloc all and Free all.
testdP3: Init a dPatiition(size=0x100). A:B:C:- ==> A:B:- ==> A:- ==> - .
testdP2: Init a dPatiition(size=0x100). A:B:C:- ==> -:B:C:- ==> -:C:- ==> - .
testdP1: Init a dPatiition(size=0x100). [Alloc,Free]* with step = 0x20
maxMallocSizeNow: MAX_MALLOC_SIZE always changes. What's the value Now?
testMalloc2: Malloc, write and read.
testMalloc1: Malloc, write and read.
help: help [cmd]
cmd: list all registered commands
xlanchen >:
```

输入 testdP2

```
Machine View
=====
EMB(start=0x109d94, size=0x14, nextStart=0xaaaaaaaa)
EMB(start=0x109da8, size=0x24, nextStart=0xbbbbbbbb)
EMB(start=0x109dcc, size=0xc0, nextStart=0x0)
Alloc memBlock C with size 0x30: success(addr=0x109dd0)!
dPartition(start=0x109d8c, size=0x100, firstFreeStart=0x109e00)
EMB(start=0x109d94, size=0x14, nextStart=0xaaaaaaaa)
EMB(start=0x109da8, size=0x24, nextStart=0xbbbbbbbb)
EMB(start=0x109dcc, size=0x34, nextStart=0xc0000000)
EMB(start=0x109e00, size=0x8c, nextStart=0x0)
Now, release A.
dPartition(start=0x109d8c, size=0x100, firstFreeStart=0x109d94)
EMB(start=0x109d94, size=0x14, nextStart=0x109e00)
EMB(start=0x109da8, size=0x24, nextStart=0xbbbbbbbb)
EMB(start=0x109dcc, size=0x34, nextStart=0xc0000000)
EMB(start=0x109e00, size=0x8c, nextStart=0x0)
Now, release B.
dPartition(start=0x109d8c, size=0x100, firstFreeStart=0x109d94)
EMB(start=0x109d94, size=0x38, nextStart=0x109e00)
EMB(start=0x109dcc, size=0x34, nextStart=0xc0000000)
EMB(start=0x109e00, size=0x8c, nextStart=0x0)
At last, release C.
dPartition(start=0x109d8c, size=0x100, firstFreeStart=0x109d94)
EMB(start=0x109d94, size=0xf8, nextStart=0x0)
xlanchen >:
```

```
hwc@hwc-virtual-machine: ~/桌面/OS_Lab5/src
cmd
cmd
list all registered commands:
command name: description
testeFP: Init a ePatiition. Alloc all and Free all.
testdP3: Init a dPatiition(size=0x100). A:B:C:- ==> A:B:- ==> A:- ==> - .
testdP2: Init a dPatiition(size=0x100). A:B:C:- ==> -:B:C:- ==> -:C:- ==> - .
testdP1: Init a dPatiition(size=0x100). [Alloc,Free]* with step = 0x20
maxMallocSizeNow: MAX_MALLOC_SIZE always changes. What's the value Now?
testMalloc2: Malloc, write and read.
testMalloc1: Malloc, write and read.
help: help [cmd]
cmd: list all registered commands
xlanchen >:
```

遇到的问题解决方法

刚开始没能完全理解上下文切换的原理，创建进程时还忘记了初始化栈空间，导致没能进入1号进程，初始化栈空间得以解决