

OS_Lab_05 Report

PB21111686_赵卓

实验题目

- FCFS调度。

实验目的

- 在前四个实验的基础上，添加FCFS进程调度功能。

实验内容

- 补全task.c中的缺失函数rqFCFSInit、rqFCFSIsEmpty、nextFCFSTsk、tskEnqueueFCFS、tskDequeueFCFS、createTsk、destroyTsk。

实验模块实现

- rqFCFSInit

此函数对进程就绪队列进行初始化，只需将队列的头尾指针赋初值为0，idleTsk赋值为idle任务即可。代码如下：

```
// 初始化就绪队列（需要填写）
void rqFCFSInit(myTCB *idleTsk)
{ // 对rqFCFS进行初始化处理
    rqFCFS.head = (void *)0;
    rqFCFS.tail = (void *)0;
    rqFCFS.idleTsk = idleTsk;
}
```

- rqFCFSIsEmpty

此函数判断就绪队列是否为空，如果是空就返回TRUE，只需判断就绪队列头尾指针是否都为0即可。代码如下：

```
// 如果就绪队列为空，返回True（需要填写）
int rqFCFSIsEmpty(void)
{ // 当head和tail均为NULL时，rqFCFS为空
    return (rqFCFS.head == (void *)0 && rqFCFS.tail == (void *)0);
}
```

- nextFCFSTsk

此函数得到下一个就绪的任务，即当前就绪队列的头指针对应进程。如果当前队列不是空的，直接返回头指针即可；如果队列为空，就返回idle任务。代码如下：

```
// 获取就绪队列的头结点信息，并返回（需要填写）
myTCB *nextFCFSTsk(void)
{ // 获取下一个Tsk
    if (rqFCFSIsEmpty())
        return rqFCFS.idleTsk;
    else
        return rqFCFS.head;
}
```

- tskEnqueueFCFS

此函数将一个新的就绪进程加入到就绪队列之中。如果队列为空，那么将头尾指针都指向该进程；如果队列为空，则将当前尾指针对应进程的下个进程指向该进程，然后更新当前尾指针指向的进程为该进程。代码如下：

```
// 将一个未在就绪队列中的TCB加入到就绪队列中（需要填写）
void tskEnqueueFCFS(myTCB *tsk)
{ // 将tsk入队rqFCFS
    if (rqFCFSIsEmpty())
    {
        rqFCFS.head = tsk;
        rqFCFS.tail = tsk;
    }
    else
    {
        rqFCFS.tail->nextTCB = tsk;
        rqFCFS.tail = tsk;
    }
}
```

- tskDequeueFCFS

此函数将当前头进程移出就绪队列。如果队列为空，操作无意义，直接返回；如果队列不为空，

长度为1时，出队后队列为空，所以直接将头尾指针都赋0；长度大于1时，将当前头指针指向的进程设为头指针的下一个进程即可。代码如下：

```
// 将就绪队列中的TCB移除（需要填写）
void tskDequeueFCFS(myTCB *tsk)
{ // rqFCFS出队
    if (rqFCFSIsEmpty())
        return;
    else
    {
        if (rqFCFS.head == rqFCFS.tail)
        {
            rqFCFS.head = (void *)0;
            rqFCFS.tail = (void *)0;
        }
        else
            rqFCFS.head = rqFCFS.head->nextTCB;
    }
}
```

- createTsk

createTsk创建一个新进程并将其加入进程池之中。如果当前没有空闲的TCB，无法创建，返回-1表示错误；如果有空闲TCB，将当前的空闲TCB给新进程使用，然后更新空闲TCB为下一个空闲TCB，然后将新进程的入口设为skBody，然后调用stack_init初始化栈空间，然后调用tskStart启动，并返回进程状态。代码如下：

```
// 以tskBody为参数在进程池中创建一个进程，并调用tskStart函数，将其加入就绪队列（需要填写）
int createTsk(void (*tskBody)(void))
{ // 在进程池中创建一个进程，并把该进程加入到rqFCFS队列中
    if (!firstFreeTsk)
        return -1;
    else
    {
        myTCB *new = firstFreeTsk;
        firstFreeTsk = firstFreeTsk->nextTCB;
        new->task_entrance = tskBody;
        stack_init(&new->stkTop, tskBody);
        tskStart(new);
        return new->TSK_ID;
    }
}
```

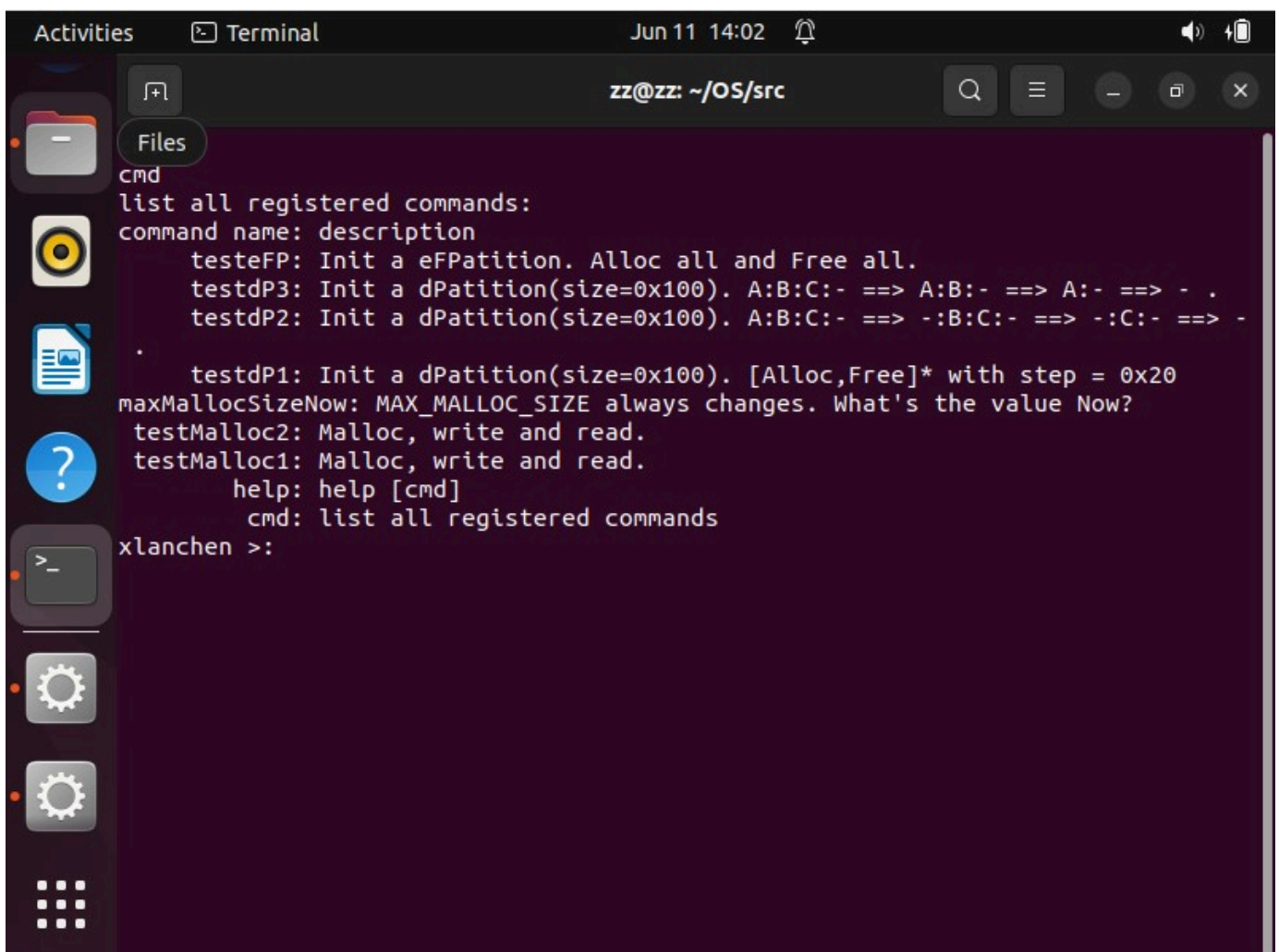
- destroyTsk

destroyTsk销毁给定序号的进程，实际上进行TCB的回收。只需要在进程池中，找到对应标号的进程TCB，然后将其入口和状态都置0，并将其下一个TCB设为当前的空闲TCB，然后更新当前TCB为该TCB。代码如下：

```
// 以takIndex为关键字，在进程池中寻找并销毁takIndex对应的进程（需要填写）
void destroyTsk(int takIndex)
{ // 在进程中寻找TSK_ID为takIndex的进程，并销毁该进程
    tcbPool[takIndex].task_entrance = (void *)0;
    tcbPool[takIndex].TSK_State = TSK_NONE;
    tcbPool[takIndex].nextTCB = firstFreeTsk;
    firstFreeTsk = &tcbPool[takIndex];
}
```

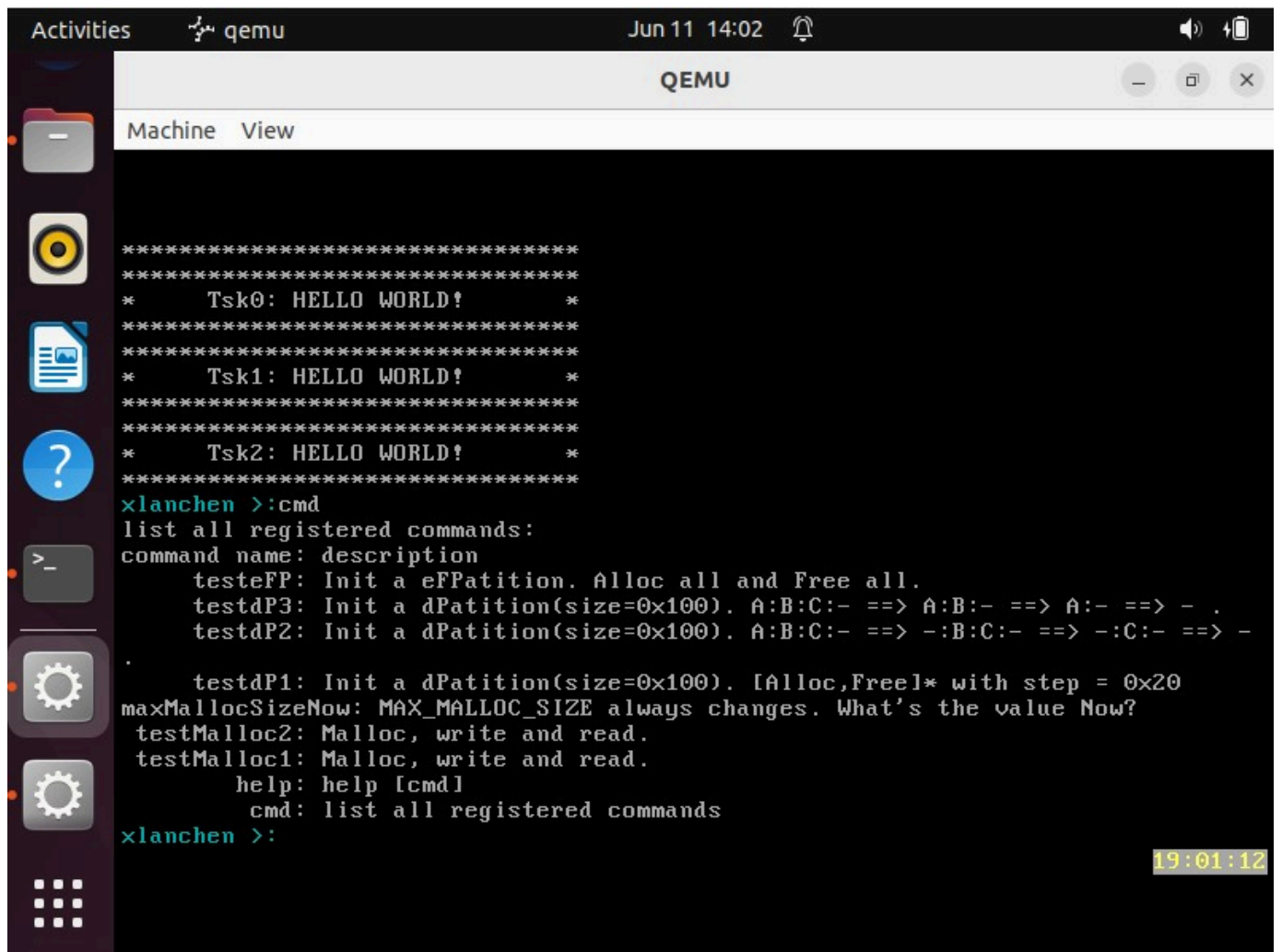
实验结果

- 运行脚本，结果如下：



The screenshot shows a terminal window titled "Terminal" with the date and time "Jun 11 14:02". The user is logged in as "zz@zz" and the current directory is "~/OS/src". The terminal displays the output of a command, which lists all registered commands and their descriptions. The output is as follows:

```
cmd
list all registered commands:
command name: description
testeFP: Init a eFPatition. Alloc all and Free all.
testdP3: Init a dPatition(size=0x100). A:B:C:- ==> A:B:- ==> A:- ==> - .
testdP2: Init a dPatition(size=0x100). A:B:C:- ==> -:B:C:- ==> -:C:- ==> - .
.
testdP1: Init a dPatition(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 >:
```



```
*****
*****
*      Tsk0: HELLO WORLD!      *
*****
*****
*      Tsk1: HELLO WORLD!      *
*****
*****
*      Tsk2: HELLO WORLD!      *
*****
xlanchen >:cmd
list all registered commands:
command name: description
    testeFP: Init a eFPatition. Alloc all and Free all.
    testdP3: Init a dPatition(size=0x100). A:B:C:- ==> A:B:- ==> A:- ==> - .
    testdP2: Init a dPatition(size=0x100). A:B:C:- ==> -:B:C:- ==> -:C:- ==> -
    .
    testdP1: Init a dPatition(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 >
```

可见结果符合预期。

思考题回答

- 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？
我们分析CTX_SW函数，根据讲义，各指令含义如下：
 - pushf: 旧进程的标志寄存器入栈。
 - pusha: 旧进程的通用寄存器入栈，此条指令和上一条指令一并，起到了保护现场的作用。
 - movl prevTSK_StackPtr, %eax: prevTSK_StackPtrAddr是指针的指针，指将其存入eax寄存器。
 - movl %esp, (%eax): ()是访存的标志，该语句的目的是存储任务的栈空间。
 - movl nextTSK_StackPtr, %esp: 该语句的目的是通过改变esp来切换栈。
 - popa: 旧进程的通用寄存器出栈。
 - popf: 旧进程的标志寄存器出栈。
 - ret: 返回指令，从栈中取出返回地址，存入eip寄存器。

```

CTX_SW:
pushf
pusha
movl prevTSK_StackPtr , %eax
movl %esp, (%eax)
movl nextTSK_StackPtr , %esp
popa
popf
ret

```

因此，CTS_SW函数中只有ret而没有call的原因即：外部调用该函数时PC寄存器的值已经被自动压栈了，而最后用ret是为了把该值出栈。

- 谈一谈你对 stack_init 函数的理解。

stack_init如下：

```

// 初始化栈空间（不需要填写）
void stack_init(unsigned long **stk, void (*task)(void))
{
    (*stk)-- = (unsigned long)0x08;    // 高地址
    (*stk)-- = (unsigned long)task;    // EIP
    (*stk)-- = (unsigned long)0x0202;  // FLAG寄存器

    (*stk)-- = (unsigned long)0xAAAAAAA; // EAX
    (*stk)-- = (unsigned long)0xCCCCCCC; // ECX
    (*stk)-- = (unsigned long)0xDDDDDDD; // EDX
    (*stk)-- = (unsigned long)0BBBBBBB; // EBX

    (*stk)-- = (unsigned long)0x44444444; // ESP
    (*stk)-- = (unsigned long)0x55555555; // EBP
    (*stk)-- = (unsigned long)0x66666666; // ESI
    (*stk) = (unsigned long)0x77777777;  // EDI
}

```

该函数用于创建进程时初始化其栈空间，从而进行上下文切换操作。同时由于栈空间从高到低地址，所以指针要一直减少。

- 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]的作用即开辟一个栈空间，stack为栈的起始地址。
 - BspContextBase[STACK SIZE]是系统栈，所有调度操作都是在这个栈里面完成的。
- prevTSK StackPtr是一级指针还是二级指针？为什么？

prevTSK StackPtr是二级指针，是指针的指针，它保存前一个进程栈顶指针的地址，即指向前一

个进程的栈顶指针。根据其定义也可以知道：

```
unsigned long **prevTSK_StackPtrAddr;
```

实验问题

- 此次实验比较简单，主要问题和上次一样，几个变量定义问题，在头文件添加extern然后将变量定义在主文件中解决。