Lab3 调度器

地理与海洋科学学院 191830173 徐嵩

Exercise

1：请把上面的程序，用 gcc 编译，在你的 Linux 上面运行，看看真实结果是啥（有一些细节需要改，请根据实际情况进行更改）

答：运行结果如下，子进程 fork 返回 0，父进程 fork 返回子进程 pid，修改程序后输出如下：

```
Father Process: Ping 943, 7;
Child Process: Pong 0, 7;
Father Process: Ping 943, 6;
Child Process: Pong 0, 6;
Father Process: Ping 943, 5;
Child Process: Pong 0, 5;
Father Process: Ping 943, 4;
Child Process: Pong 0, 4;
Child Process: Pong 0, 3;
Father Process: Ping 943, 3;
Father Process: Ping 943, 2;
Child Process: Pong 0, 2;
Child Process: Pong 0, 1;
Father Process: Ping 943, 1;
Child Process: Pong 0, 0;
Father Process: Ping 943, 0;
```

2：请简单说说，如果我们想做虚拟内存管理，可以如何进行设计？（比如哪些数据结构，如何管理内存）

答：采用请求分页的方式实现虚拟内存管理。操作系统维护一张记录内存中空闲物理块号的表，一个物理块被使用后就从表中删除。每创建一个进程就构建一个以页号为键的页表，每个页号对应一个内存物理块号、是否在内存中的标记位 P、访问次数 A、脏位 M 和外存物理块号。

当 CPU 在页表中访问页面不存在时，就产生缺页中断，操作系统为该页表项分配一个物理块号，将该页装入内存。如果内存不足则先释放个内存块。并且，每过一段时间对所有进程的页表进行检查，将未访问的内存块释放，减小内存占用。在释放一个物理块时，根据其脏位判断是否写回外存。

3：我们考虑这样一个问题：假如我们的系统中预留了 100 个进程，系统中运行了 50 个进程，其中某些结束了运行。这时我们又有一个进程想要开始运行（即需要分配给它一个空闲的 PCB），那么如何能够以 O(1)的时间和 O(n)的空间快速地找到这样一个空闲 PCB 呢？

答：创建进程调度队列，将不同状态的进程分别加入不同的队列中(RUNNABLE/BLOCKED/DEAD)需要给一个新进程分配 PCB 时，只需要从 DEAD 队列中取一个即可，并更改该 PCB 状态标识。同时，当一个进程运行结束时，将对应的 PCB 从其他队列删除，加入 DEAD 队列。

4：请你说说，为什么不同用户进程需要对应不同的内核堆栈？

答：因为 CPU 只有一个，但现代操作系统需要面对多进程并发的情况，在中断到来时，需要通过进程的内核堆栈保存当前处理器上下文；当进程再次获得 CPU 时，需要通过内核堆栈恢复切换前保存的上下文。

5：stackTop 有什么用？为什么一些地方要取地址赋值给 stackTop？

答：stackTop 存放内核栈的栈顶指针，在创建进程时，需要根据进程分配到的 PCB 来指定对应的内核栈，此时需要取地址赋值给 stackTop。切换进程时会将对应 PCB 的 stackTop 赋值给 esp 作为内核栈栈顶。同时 stackTop 本身的地址是内核栈的栈底，进程切换时需要将 stackTop 的地址赋给 tss 段的 esp0。

6：请说说在中断嵌套发生时，系统是如何运行的？

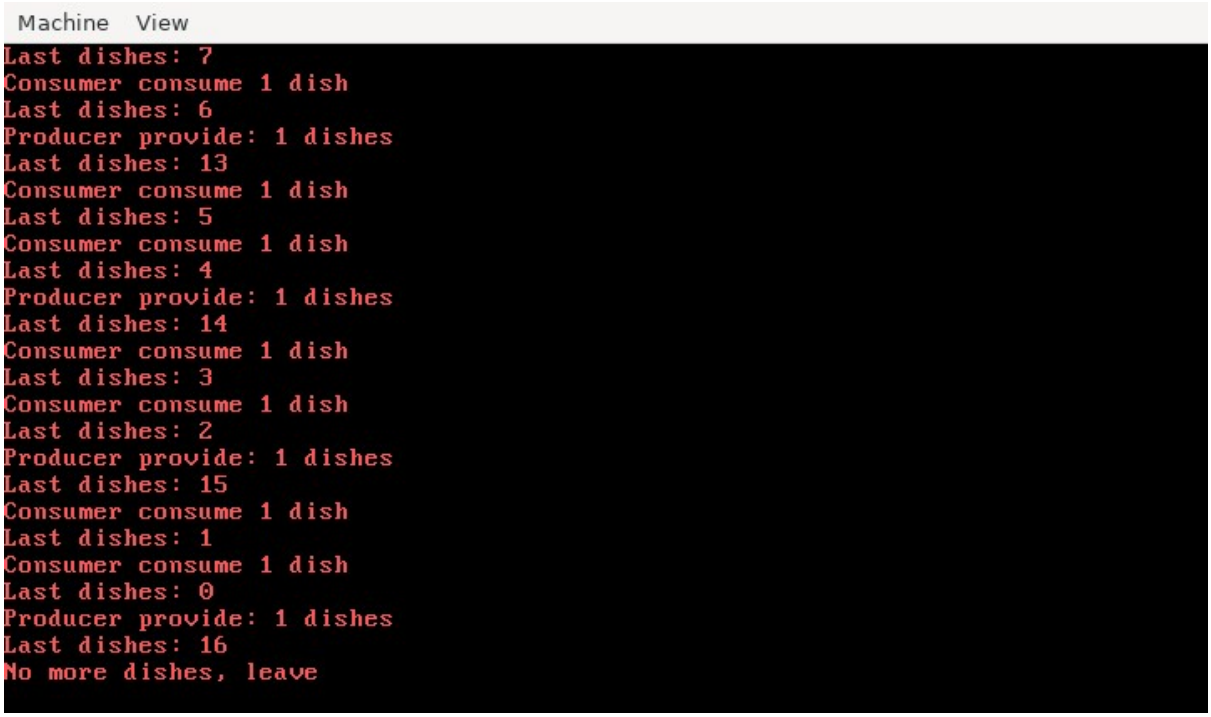
答：发生中断嵌套时，系统首先将 eflags、cs、eip、出错码、中断向量号、其他寄存器等入内核栈，然后将当前进程 PCB 对应的 stackTop 赋值为当前内核栈的栈顶。进程切换时，通过 stackTop 找到最内层中断内核栈的栈顶，然后层层退出返回用户态。

7：线程为什么要以函数为粒度来执行？

答：更大的粒度是进程，进程是处理器分配资源和调度的基本单位；更小的力度是单条语句，用于完成某一步特定的操作。线程有独立的栈区、寄存器，共享全局变量和堆区，线程之间的切换代价很小。

8：请用 fork，sleep，exit 自行编写一些并发程序，运行一下，贴上你的截图。

答：生产者每 100 个时钟周期生产 1 dish，消费者每 50 个时钟周期消耗 1 dish，当所有的 dish 被消耗完，消费者进程就会结束运行。下图可以看出，消费者每被调用两次，生产者被调用一次，由于没有实现进程共享变量所以消费者空间的 dish 和生产者空间的 dish 不同。



9：请问，我使用 loadelf 把程序装载到当前用户程序的地址空间，那不会把我 loadelf 函数的代码覆盖掉吗？

答：loadelf 是在内核栈中运行的，并且把保存在内核栈中的 eip 指向了新加载程序的入口地址，这样从内核态返回时，将到新程序的入口地址处运行。

Challenge

1：请发挥你的编码能力，在实验框架代码上设计一款虚拟内存管理机制。并说明你的设计（可以和上面的 exercise 结合）

2：请说说内核级线程库中的 pthread_create 是如何实现即可。

答：首先分配栈空间，初始化 pthread 结构体和 pthread_attr 结构体，内容从父线程拷贝，然后设置一些必要的标志。然后再将设置好的子线程加入调度器管理调度。

3：你是否能够在框架代码中实现一个简易内核级线程库，只需要实现 create，destroy 函数即可，并仍然通过时钟中断进行调度，并编写简易程序测试。

4： A. 在 create_thread 函数里面，我们要用线性时间搜索空闲 tcb，你是否有更好的办法让它更快进行线程创建？

B. 我们的这个线程库，父子线程之间关联度太大，有没有可能进行修改，并实现一个调度器，进行线程间自由切换？（这时 tcb 的结构可能需要更改，一些函数功能也可以变化，你可以说说思路，也可以编码实现）

C.修改这个线程库或者自由设计一个线程库。

答：在程序初始化时创建一个空线程的链表，链表保存了空线程的 id 和下一个空线程的地址。创建线程时取链表头节点保存的线程 id 即可；线程销毁时将该线程 id 加入到空线程链表中即可。代码如下：

```
typedef struct Node {
	int id;
	struct Node *next;
} Node;

int empty_size = maxThread;
Node *empty_tcb_list;

int get_empty_Id(){
    if (empty_size <= 0) return -1;
    int id = empty_tcb_list->id;
    int del_tcb = empty_tcb_list;
    empty_tcb_list = empty_tcb_list->next;
    empty_size--;
    free(del_tcb);
    printf("Aquire id: %d, empty size: %d\n", id, empty_size);
    return id;
}

void release_Id(int id){
    Node *node = (Node *)malloc(sizeof(Node));
    node->id = id;
    node->next = empty_tcb_list;
    empty_tcb_list = node;
    empty_size++;
    printf("Release id: %d, empty size: %d\n", id, empty_size);
}
```

```
Aquire id: 0, empty size: 255
***thread(id 0) create***
Aquire id: 1, empty size: 254
***thread(id 1) create***
***thread(id 0) begin***
ping 1
***thread(id 1) begin***
pong 1
…
ping 6
***thread(id 1) finish***
Release id: 1, empty size: 255
  0
…
ping 10
***thread(id 0) finish***
Release id: 0, empty size: 256
  0
```

5：你是否可以完善你的 exec，第三个参数接受变长参数，用来模拟带有参数程序执行。举个例子，在 shell 里面输入 cat a.txt b.txt，实际上 shell 会 fork 出一个新的 shell（假设称为 shell0），并执行 exec("cat", "a.txt","b.txt")运行 cat 程序，覆盖 shell0 的进程。不必完全参照 Unix，可以有自己的思考与设计。