

操作系统 Lab3——进程切换

151220045 蒋雨霖

一. 实验目的:

本实验通过实现一个简单的任务调度, 介绍基于时间中断进行进程切换完成任务调度的全过程

二. 实验思路

1. 实现时钟中断:

将讲义代码拷贝到 `time.c` 文件中, 并且在 `idt` 表添加相应的表项, 添加中断门的时候需要添加到第 `0x20` 个表项上。

2. 实现 PCB 表:

结构如下:

```
14
15 typedef struct ProcessTable
16 {
17     struct TrapFrame regs;
18     int state;
19     int timeCount;
20     int sleepTime;
21     unsigned int pid;
22     struct ProcessTable *next;
23 }ProcessTable;
24
25
```

定义了一个 `pcb` 数组 `pcb[]`, `state` 记录每个进程的状态 (即 `RUNNING`、`RUNNABLE`、`BLOCKED`、`DEAD` 等), `timeCount` 记录每个进程的处理 (`RUNNING`) 时间片, `sleepTime` 记录每个进程阻塞 (`BLOCKED`) 的时间片, `pid` 记录每个进程的进程号。

3. 实现调度程序:

(1): 堆栈切换:

我预先将 `tss.esp0` 设置为当前进程的 `pcb` 中的地址,

系统运行在用户态并且发生中断时, 硬件会首先将堆栈切换到 `tss.esp0` 指示的内核栈, 也就是当前进程的 `PCB` 中的地址, 在再调度的时候修改 `tss.esp0` 的值

(2): idle 线程:

当目前系统中没有进程参与调度时, 让 `idle` 线程占用 `cpu`, 由一个函数实现, 所做的就是打开中断等待中断的到来。

(3): 中断处理

只需要将当前进程的 `timeCount` 减 1, 并且将阻塞队列中所有进程 `sleepTime` 减 1, 当睡眠时间减为 0, 则将这个进程从阻塞队列中移动到就绪队列中。如果当前进程的时间片减为 0, 则为这个进程重新分配时间片并加入到就绪队列中, 然后进行调度。

(4): 调度:

调度时让就绪队列的队头进程获取 cpu, 如果队列为空, 则让 idle 线程获得 cpu。

4. 实现系统调用:

主要实现三个函数 sleep、exit、fork。

Sleep:

SLEEP 系统调用用于进程主动阻塞自身, 内核需要将该进程由 RUNNING 状态转换为 BLOCKED 状态, 设置该进程的 SLEEP 时间片, 并切换运行其他 RUNNABLE 状态的进程

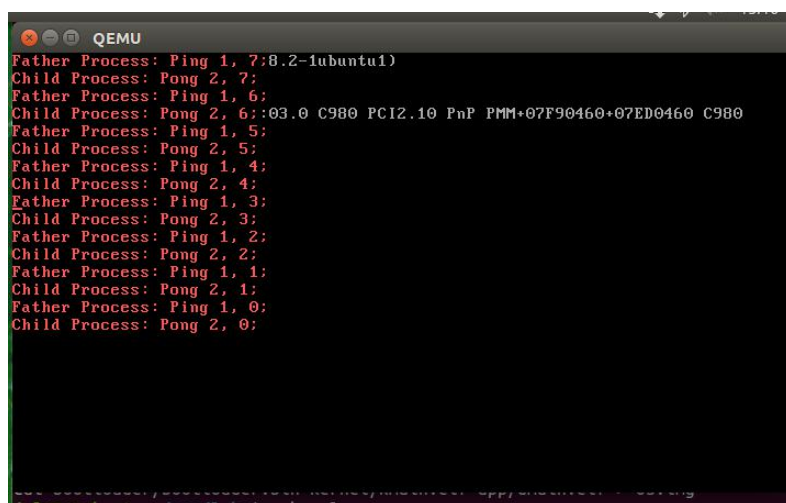
Exit:

EXIT 系统调用用于进程主动销毁自身, 内核需要将该进程由 RUNNING 状态转换为 DEAD 状态, 回收分配给该进程的内存、进程控制块等资源, 并切换运行其他 RUNNABLE 状态的进程

Fork:

FORK 系统调用用于创建子进程, 内核需要为子进程分配一块独立的内存, (实现 fork 时在 gdt 表中添加两个段, 子进程的代码段和数据段) 将父进程的地址空间、用户态堆栈完全拷贝至子进程的内存中, 并为子进程分配独立的进程控制块, 完成对子进程的进程控制块的设置若子进程创建成功, 则对于父进程, 该系统调用的返回值为子进程的 pid, 对于子进程, 其返回值为 0; 若子进程创建失败, 该系统调用的返回值为-1

三. 实验效果截图:



```
QEMU
Father Process: Ping 1, 7:8.2-1ubuntu1)
Child Process: Pong 2, 7:
Father Process: Ping 1, 6:
Child Process: Pong 2, 6::03.0 C980 PCI2.10 PnP PMM+07F90460+07ED0460 C980
Father Process: Ping 1, 5:
Child Process: Pong 2, 5:
Father Process: Ping 1, 4:
Child Process: Pong 2, 4:
Father Process: Ping 1, 3:
Child Process: Pong 2, 3:
Father Process: Ping 1, 2:
Child Process: Pong 2, 2:
Father Process: Ping 1, 1:
Child Process: Pong 2, 1:
Father Process: Ping 1, 0:
Child Process: Pong 2, 0:
```

四. 实验心得:

这次实验沿用了 lab2 的框架, 相对复杂了许多。在实验过程中, 开始时由于自己对堆栈切换理解不够清楚, 一直无法正确的产生时钟中断, 也是调了很长时间才能够改正。通过这次 lab 中遇到的大大小小的问题, 让我对计算机系统的中断机制也有了更加深刻的认识。

