

---

# LAB3 实验报告

张运吉 (211300063、211300063@smail.nju.edu.cn)

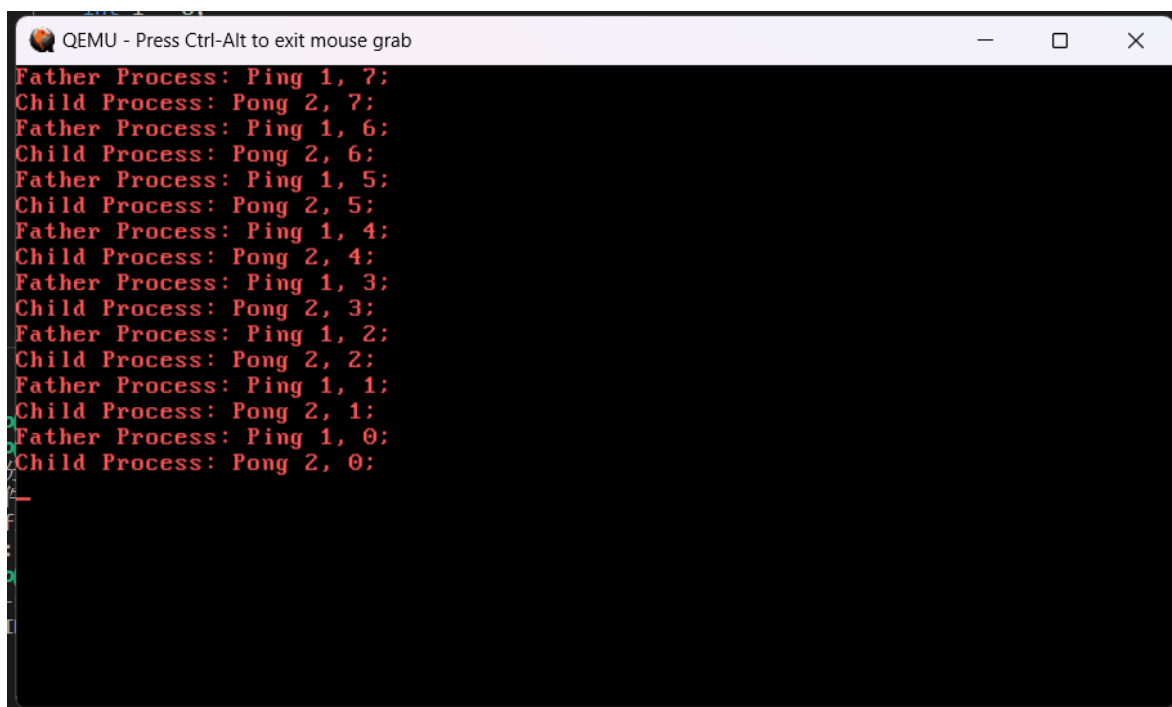
(南京大学人工智能学院, 南京 210093)

## 1 实验进度

我已经完成 **LAB3** 全部必做内容，并添加了中断嵌套的代码。

## 2 实验结果

正确通过样例:



```
QEMU - Press Ctrl-Alt to exit mouse grab
Father Process: Ping 1, 7;
Child Process: Pong 2, 7;
Father Process: Ping 1, 6;
Child Process: Pong 2, 6;
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;
```

## 3 TASK 关键代码实现

### 3.1 完成库函数

这一部分是对 lab2 的复习，比较简单：根据 lib.h 中给出的系统调用号调用 syscall 函数即可。

```
#define SYS_WRITE 0
#define SYS_FORK 1
#define SYS_EXEC 2
#define SYS_SLEEP 3
#define SYS_EXIT 4
```

### 3.2 时钟中断处理

根据讲义的提示：需要遍历 pcb, 将状态为 STATE\_BLOCK 的进程的 sleepTime 减一，如果进程的 sleepTime 变为 0，重新设为 STATE\_RUNNABLE。我用一个 for 循环实现这个功能。

```
for (int i = 0; i < MAX_PCB_NUM; i++) {
    if (pcb[i].state == STATE_BLOCKED) {
        if (--pcb[i].sleepTime == 0)
            pcb[i].state = STATE_RUNNABLE;
    }
}
```

将当前进程的 timeCount 加一，如果时间片用完且有其他状态为 STATE\_RUNNABLE 的进程，则切换，否则继续执行当前进程。这里涉及到调度的问题，根据讲义内容使用简单的轮转调度即可，这里我使用一个环形数组的思想实现。

```
int next = current, i = 1; // next指向下一个要调度的进程
for (; i < MAX_PCB_NUM; i++) { // 从current进程开始，向后循环遍历pcb一遍，找到处于STATE_RUNNABLE的进程
    if (pcb[(current + i) % MAX_PCB_NUM].state == STATE_RUNNABLE) {
        next = (current + i) % MAX_PCB_NUM;
        break;
    }
}
```

接下来就是进程切换的具体过程了，讲义以及给出这部分的代码，于是不再赘述。

### 3.3 系统调用例程

#### 3.3.1 syscallFork

基本思想是先找到一个空闲的进程控制块(即状态为 STATE\_DEAD 的)，然后将父进程的代码段和数据段完全拷贝，将父进程的 pcb 完全拷贝，然后再考虑子进程 pcb 中和父进程无关的内容，最后在父进程和子进程 pcb.reg.eax 存放返回值。

这里，我参考了 initProc 中初始化 pcb[0] 和 pcb[1] 的部分代码。

拷贝代码段和数据段：

```
for (int j = 0; j < 0x100000; j++)
    *(uint8_t *) (j + (child_id + 1) * 0x100000) = *(uint8_t *) (j + (current + 1) * 0x100000);
```

拷贝 pcb:

```
for (int j = 0; j < sizeof(ProcessTable); ++j)
    *((uint8_t *) (&pcb[child_id]) + j) = *((uint8_t *) (&pcb[current]) + j);
```

修改 pcb:

```

pcb[child_id].stackTop = (uint32_t) & (pcb[child_id].regs);
pcb[child_id].prevStackTop = (uint32_t) & (pcb[child_id].stackTop);
pcb[child_id].state = STATE_RUNNABLE;
pcb[child_id].timeCount = 0;
pcb[child_id].sleepTime = 0;
pcb[child_id].pid = child_id;

pcb[child_id].regs.ss = USEL(2 + 2 * child_id);
pcb[child_id].regs.cs = USEL(1 + 2 * child_id);
pcb[child_id].regs.ds = USEL(2 + 2 * child_id);
pcb[child_id].regs.es = USEL(2 + 2 * child_id);
pcb[child_id].regs.fs = USEL(2 + 2 * child_id);
pcb[child_id].regs.gs = USEL(2 + 2 * child_id);

pcb[child_id].regs.eax = 0;
pcb[current].regs.eax = child_id;

```

### 3.3.2 syscallSleep

将进程的状态设置为 STATE\_BLOCKED,将当前进程的 sleepTime 设为传入的参数, 存在 ecx 寄存器中, 然后模拟时钟中断进行进程切换。

### 3.3.3 syscallExit

将当前进程的状态设置为 STATE\_DEAD,然后模拟时钟中断进行进程切换。

## 4 思考题

### 4.1 Exercise1

区别:

1. fork()用于创建新进程, 父进程和子进程是独立的进程, 它们的 pid 不同, 而 execve 用于加载执行程序, 在该进程的地址空间加载一个程序并执行;
2. 两者的返回值也不同, 调用 fork()若成功, 父进程返回子进程的 pid, 子进程返回 0, 若失败父进程返回-1, 调用 execve()返回执行该程序的返回值。

相关伪代码:

```

1  pid_t fork(void) {
2      child = create_process(); // 创建一个新的进程
3      copy_memory(parent, child); // 复制父进程的内存空间到子进程
4      add_to_schedule_queue(child); // 将子进程加入进程调度队列
5      return child.pid; // 返回子进程ID给父进程, 父进程可以使用这个ID来识别子进程, 并做出相应的操作。
6  }
7
8  int execve(const char *filename, char *const argv[], char *const envp[]) {
9      load_program(filename); // 加载可执行文件到内存中, 并设为当前进程的代码段
10     setup_arguments(argv); // 将参数放入堆栈, 以便程序访问
11     setup_envp(envp); // 设置环境变量
12     jump_to_entry_point(); // 跳转到程序入口点, 开始执行
13 }

```

## 4.2 Exercise2

**fork:** 创建新进程，子进程是对父进程的复制，但子进程也只是选择性的继承父进程的数据，调用 **fork** 之后父进程的状态无影响，子进程的状态为就绪态。

**execve:** 改变进程的执行状态，销毁原来进程的代码和数据，加载新程序的代码和数据，并跳转新程序的入口执行。

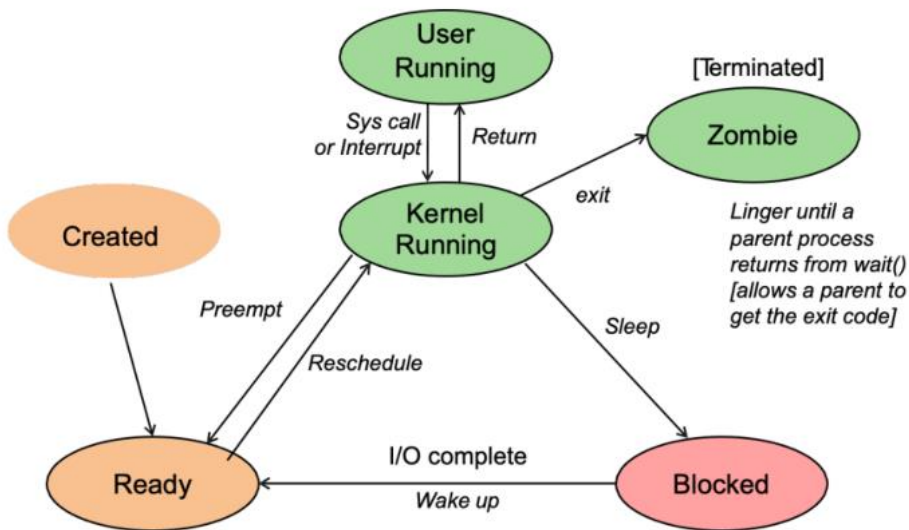
**wait:** 等待子进程结束，获取子进程的退出状态，它对进程状态的影响是阻塞父进程直到子进程结束退出。如果子进程退出时，父进程没有对其回收就会造成内存泄露。

**exit:** 结束进程，回收资源，进程的状态变为终止态。

## 4.3 Exercise3

操作系统首先将该用户态进程设置为就绪态，并将用户进程加入到就绪队列中等待 CPU 分配时间片。当调度器选择该进程执行时，CPU 会将该进程的程序计数器、栈指针等寄存器设置为进程上一次执行时，恢复进程上下文，若是初次创建的进程，则把程序计数器设置为进程入口地址，并初始化栈指针等系统寄存器。最后 CPU 从进程的开始地址（程序计数器）执行第一条指令。

## 4.4 Exercise4



## 5 收获与感想

了解了基于时间中断进行进程切换和任务调度的全过程。加深了对 **fork**、**execve**、**wait**、**exit** 等系统调用函数的理解。