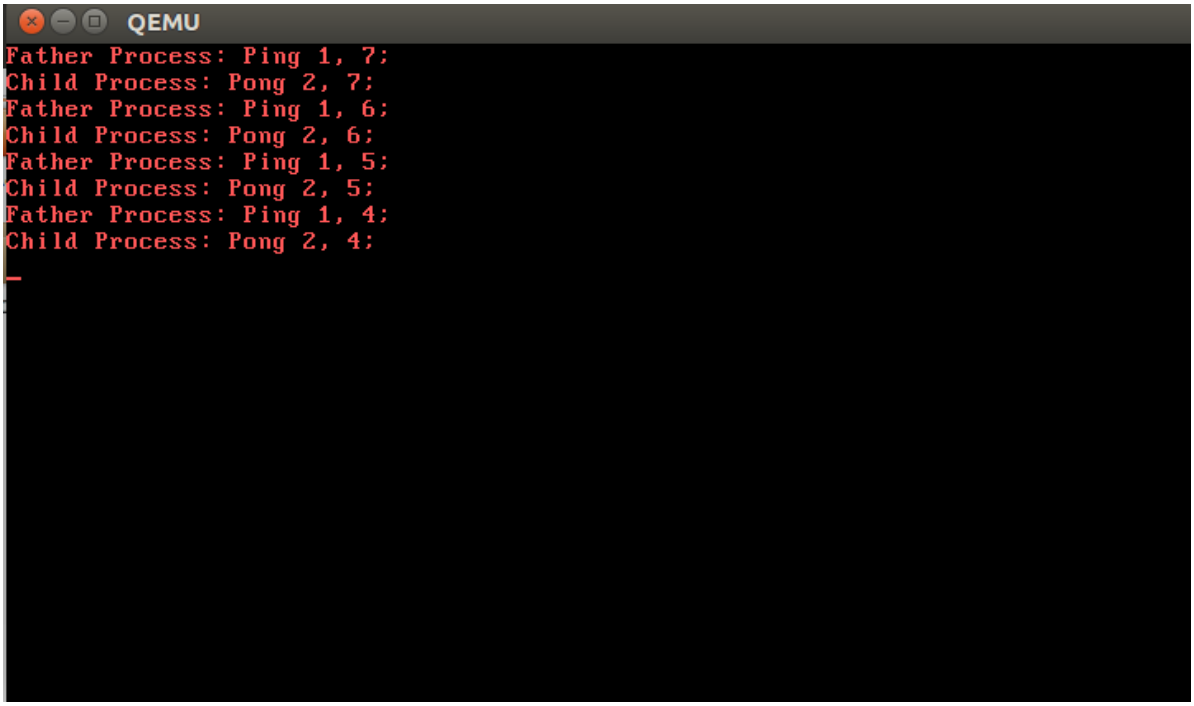
lab3 实验报告

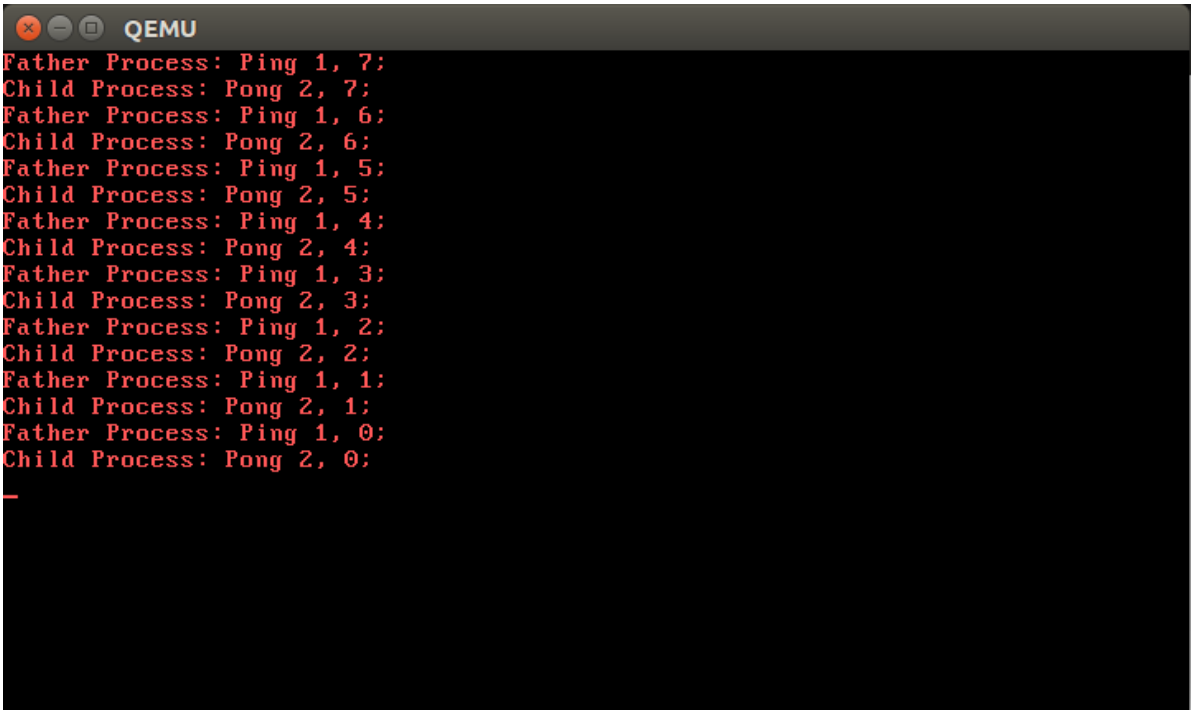
李培佳 191300029 191300029@smail.nju.edu.cn

1. 实验进度

实现了进程管理功能，基于时间中断进行进程切换完成任务调度，完成了`fork, exit, sleep`库函数和对应的处理例程实现

2. 实验结果





3. 修改的代码位置

阅读整体代码框架，本次实验和`lab2`较为接近，需要完成的函数为`fork(), sleep(), exit()`，用户程序使用上述函数的具体过程还是系统调用，只需要在`lib/syscall.c`中添加相应函数的系统调用即可。

3.1 时钟中断处理

根据讲义中的提示，我们可以大致了解`timeHandle()`的流程

- 当每次进入时钟中断时，首先遍历`pcb`，找到状态为`STATE_BLOCKED`的进程，将此进程的`sleeptime`减一，如果某个进程的`sleeptime`减一之后变成0，则将此进程的状态设为`STATE_RUNNABLE`。
- 框架代码中有全局变量`current`用于记录当前的进程在`pcb`中的索引号，将当前进程的`timeCoun`加一，如果当前进程的`timeCount`超过`MAX_PCB_NUM`，我们接下来要在`pcb`中搜索是否有`STATE_RUNNALBE`的进程，如果没有找到，则判断当前进程是否可以继续执行，如果可以，则将`timeCount`重新设置为0，并继续执行当前进程，如果当前进程无法继续执行，则返回内核代码（即设置`current = 0`）。
- 当前进程的`timeCount`没有超过`MAX_PCB_NUM`，则不用进程切换，直接继续执行当前进程即可。
- 我们通过上述的判断更新`current`，并默认切换到`current`的进程上，进程切换的代码使用了讲义上提供的代码。

3.2 系统调用例程

3.2.1 sleep

将当前进程的状态设置为`STATE_BLOCKED`，并将传入的`time`传给当前进程的`sleeptime`，如果`time`小于等于0，则不进行赋值，最后将当前进程的`timeCount`设置为`MAX_TIME_COUNT`，使用内联汇编`int 0x20`进入时钟中断即可。

3.2.2 exit

将当前进程的状态设置为`STATE_DEAED`，并将`timeCount`设置为`MAX_TIME_COUNT`，使用内联汇编`int 0x20`进入时钟中断。

3.2.3 fork

`fork`用于创建一个子进程，并返回子进程的进程号（记为`i`），如果创建失败，则返回`-1`。

注意`fork`只涉及创建子进程，不涉及进程切换。

我们需要在`pcb`中找到一个状态为`STATE_DEAD`的进程块，如果不存在，直接返回`-1`即可。找到之后，将父进程的资源复制给子进程，内存拷贝的代码使用了讲义中提供的代码。接下来对子进程的一些属性进行设置

- 子进程的`pid`即为我们一开始找到的进程号`i`，`timeCount, sleepTime`为0，`state`为`STATE_RUNNABLE`，除了`eax`以外的寄存器(包括`eflags`)全部直接拷贝`current`（即父进程）的寄存器，`eax`设置为0，（调用`fork`后父进程返回子进程的`pid`，子进程返回0），`current`的`eax`赋值为`i`。
- 子进程的段寄存器不能直接复制，需要通过计算得到，除了`cs`为`USEL(2 * i + 1)`，其余为`USEL(2 * i + 2)`
- 子进程的`preStackTop`记录目前的`stackTop`即可，随后给`stackTop`赋值为`pcb[i].regs`