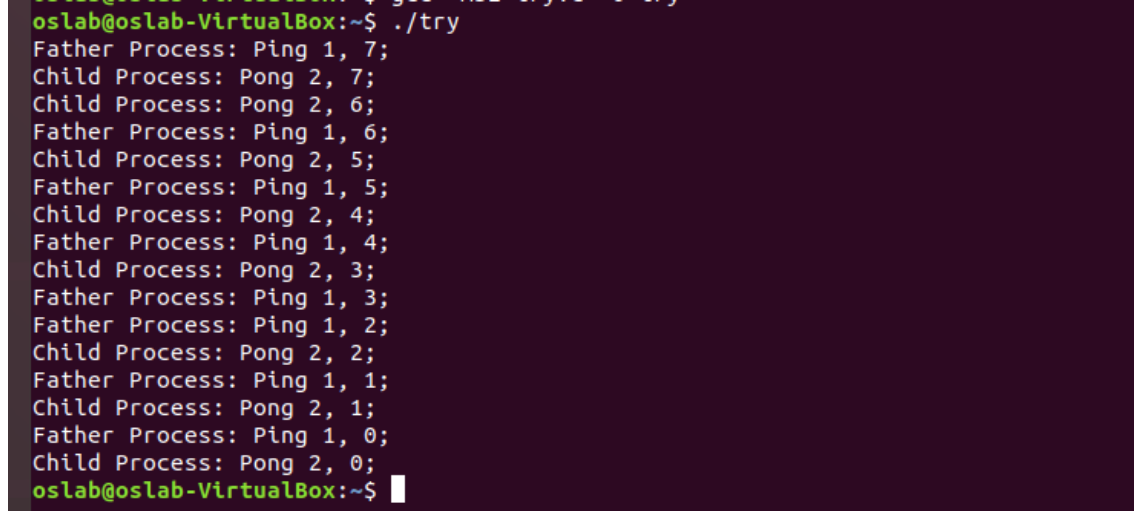


1. exercises

- **exercise 1**

如下为手册中代码的执行结果：



```
oslab@oslab-VirtualBox:~$ ./try
Father Process: Ping 1, 7;
Child Process: Pong 2, 7;
Child Process: Pong 2, 6;
Father Process: Ping 1, 6;
Child Process: Pong 2, 5;
Father Process: Ping 1, 5;
Child Process: Pong 2, 4;
Father Process: Ping 1, 4;
Child Process: Pong 2, 3;
Father Process: Ping 1, 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;
oslab@oslab-VirtualBox:~$
```

- **exercise 2**

采用分段机制和分页机制对内存地址进行转换。地址转化过程包括两个阶段：第一阶段，通过分段机制把程序的逻辑地址转换为处理器可寻址空间（即线性地址）；第二阶段，通过分页机制将线性地址转换为物理地址。

分段机制：逻辑地址包含一个段选择符和一个偏移量，段选择符提供了段描述符表中的一个数据结构（段描述符）的偏移量。段描述符中的信息包括段的大小，访问权限和段的特权级、段类型以及段基址。逻辑地址的偏移量加上段描述符中的段基址，就形成了处理器线性地址空间中的地址。

分页机制：将每个段划分为固定大小的页面（通常为4KB），通过两级页表可以将线性地址转换为物理地址。第一级表为目录，第二级表为页表，线性地址的高10位为页目录索引，用来查询页目录找到对应页表的地址，中间10位为页表索引，用来查询页表获得物理地址的高20位，将20位物理基地址和低12位偏移量组合即可获得物理地址。

- **exercise 3**

建立一个队列，将当前处于空闲的PCB的PID放入队列，每次有新的进程需要分配PID时，就从队列中取出一个PID，将其分配给该进程。

- **exercise 4**

因为不同的进程之间是“隔离”的，它们的执行互不相干，它们不共享内存，所以需要各自单独的内存空间。

• exercise 5

stackTop是taskFrame结构体的首地址。

• exercise 6

保存当前中断处理过程的现场信息，跳转至irqhandle进行处理，当前中断处理执行完毕要返回至上一级中断时，不断通过pop操作恢复寄存器的值，再通过iret指令返回到上一级中断处理过程被打断处继续执行。

• exercise 7

一个函数是用来完成一个特定任务的代码模块，其中包含多条指令。相比于进程，函数所包含的指令的规模更小，很多时候我们并不需要一个进程中的所有指令都能够并发执行，可能只是需要某几个代码段能够并发执行，如果此时以进程作为粒度来并发执行，那么会造成不必要的空间和时间开销。

但如果以函数作为粒度来进行并发，就可以把需要进行并发的几个代码段抽象为几个函数，在创建线程时，只需要对这几个代码段进行空间分配与拷贝，而一些无关代码就不需要再为其分配内存与拷贝了，这大大节约了空间和时间成本。

同时，一条指令所能完成的任务实在有限，即使是一个很简单的任务也需要多条指令组合在一起才能实现，如果以指令作为粒度进行线程的创建和执行，那需要创建非常多的线程，这会造成比较大的时间开销。

综上，我认为线程以函数作为粒度来执行是最合适的。

• exercise 8

代码：

```
#define NUM 5000
int count = 0;
int main(){
    int ret = fork();
    for(int i = 0; i < NUM; i++){
        count++;
        if(ret == 0)
        {
            printf("child process: count = %d\n", count);
            exit(1);
        }
        else if(ret != -1)
        {
            sleep(2);
            printf("parent process: count = %d\n", count);
            exit(1);
        }
    }
    return 0;
}
```

运行结果:

```
oslab@oslab-VirtualBox:~$ gcc -ms2 try.c -o try
oslab@oslab-VirtualBox:~$ ./try
child process: count = 5000
parent process: count = 5000
```