实验五 编制模拟“五个哲学家”问题的程序

姓名：刘梦杰 学号：22920212204168 完成时间：2023.12.8

一、实验目的

学习和掌握并发进程同步的概念和方法。

二、实验内容

具体要求：

1、 程序语法

philosopher [-t <time>]

<time>是哲学家进餐和沉思的持续时间值，缺省值为 2 秒。

2、五个哲学家的编号为 0~4，分别用五个进程独立模拟。

3、程序的输出要简洁，仅输出每个哲学家进餐和沉思的信息。例如，当编号为 3 的哲学家在进餐时，就打印：

philosopher 3 is eating

而当他在沉思时，则打印：

philosopher 3 is thinking

除此之外不要输出其他任何信息。

4、利用课堂已讲授的知识而不使用线程或 IPC 机制进行同步。

5、程序应该一直运行，直到人为地终止它（按 Ctrl-C 或 Ctrl-\）。不允许出现僵尸进程。

三、 实验原理及设计思路

该实验的要点是，解决并发环境下，多进程之间的同步与互斥问题。进程间的同步互斥必然涉及进程间的通信（信息交换）。但是进程的内存空间是彼此隔离的，它们之间的通信只能通过如下手段：IPC 机制、管道、信号或文件。就目前所学知识和实验要求而言，只能使用文件。

所以关键在于进程之间如何通过文件来通信？查看 lock.c 文件中 lock() 和 unluck() 函数的实现：

```c
#include    <sys/types.h>
#include    <sys/stat.h>
#include    <fcntl.h>
#include    <unistd.h>
#include    "apue.h"

void initlock(const char *lockfile)
{
    int i;
    unlink(lockfile);
}

void lock(const char *lockfile)
{
```

```
    int fd;
    while ( (fd = open(lockfile, O_RDONLY | O_CREAT | O_EXCL,
FILE_MODE)) < 0)
        sleep(1);
    close(fd);
}

void unlock(const char *lockfile)
{
    unlink(lockfile);
}
```

lock() 函数中为 open() 函数指定了参数 O_CREAT 和 O_EXCL，含义分别为：

O_CREAT　　若此文件不存在则创建它

O_EXCL　　如果同时指定了 O_CREAT，而文件已存在，则出错返回。、

据此可以测试一个文件是否存在。

当一个进程尝试 open() 一个文件时，如果该文件已经存在，则出错返回-1，延时 1s，接着继续尝试；如果该文件不存在，则创建该文件，且创建后其他进程无法再创建该文件。

而 unlock() 函数中则使用 unlink() 函数删除该文件。

所以可以这样模拟：

某一个哲学家拿起叉子（对应进程创建一个文件），放下叉子前（文件未被删除前），其他哲学家不能使用被拿起的叉子（其他进程无法创建已经存在的文件）。这样就实现了进程之间的通信。

大部分代码已经给出：

```
// 定义5 个文件，分别表示5 个叉子
static char *forks[N] = {"fork0", "fork1", "fork2", "fork3",
"fork4"};


// 哲学家的行为
void philosopher(int i)
{
    while (1)
    {
        thinking(i, nsecs); // 哲学家i 思考nsecs 秒
        takeFork(i);        // 哲学家i 拿起叉子
        eating(i, nsecs);   // 哲学家i 进餐nsecs 秒
        putFork(i);         // 哲学家i 放下叉子
    }
}
```

```
// main 函数中生成 5 个哲学家进程
for (i = 0; i < N; i++){
    if ((pid = fork()) < 0){
        err_quit("fork error");
    }
    else if (pid == 0){ // 父进程
        philosopher(i);
    }
}
wait(NULL);


// 拿起叉子
void takeFork(int i) {
    if (i == N - 1) {
        lock(forks[0]);
        lock(forks[i]);
    }
    else {
        lock(forks[i]);
        lock(forks[i + 1]);
    }
}

// 放下叉子
void putFork(int i) {
    if (i == N - 1) {
        unlock(forks[0]);
        unlock(forks[i]);
    }
    else {
        unlock(forks[i]);
        unlock(forks[i + 1]);
    }
}
```

整体框架已经给出，只需要补充以下代码

参数检查:

```
// 参数 check
if (argc != 1 && argc != 3){
    err_quit("usage: ./philosopher [ -t <time> ]");
}
else if (argc == 3){
```

```
    if (strcmp(argv[1], "-t") != 0){
        err_quit("usage: ./philosopher -t <time>");
    }
    else{
        nsecs = atoi(argv[2]);
    }
}
```

thinking() 和 eating() 函数:

```
void thinking(int i, int nsecs)
{
    printf("philosopher %d is thinking\n", i);
    sleep(nsecs);
}

void eating(int i, int nsecs)
{
    printf("philosopher %d is eating\n", i);
    sleep(nsecs);
}
```

三、实验结果

源程序名：philosopher.c

可执行程序名：philosopher

编译：

输入命令 gcc philosopher.c lock.c error.c -o philosopher 进行编译，生成可执行文件 philosopher.

```
[cs214168@mcore 5]$ gcc philosopher.c lock.c error.c -o philosopher
在包含自 lock.c：5 的文件中：
include/apue.h:7:1: 警告：“_POSIX_C_SOURCE”重定义
在包含自 /usr/include/sys/types.h：27 的文件中，
                 从 lock.c：1：
/usr/include/features.h:198:1: 警告：这是先前定义的位置
```

运行程序：

程序语法为

./philosopher [-t <time>]

其中<time>参数可以指定也可以不指定，缺省值为 2 秒。

运行结果：

不指定<time>参数:

```
[cs214168@mcore 5]$ ./philosopher
philosopher 0 is thinking
philosopher 1 is thinking
philosopher 2 is thinking
philosopher 3 is thinking
philosopher 4 is thinking
philosopher 0 is eating
philosopher 2 is eating
philosopher 0 is thinking
philosopher 2 is thinking
philosopher 1 is eating
philosopher 3 is eating
philosopher 1 is thinking
philosopher 3 is thinking
philosopher 4 is eating
philosopher 2 is eating
philosopher 4 is thinking
philosopher 2 is thinking
philosopher 1 is eating
philosopher 3 is eating

[cs214168@mcore 5]$
```

指定<time>参数:

```
[cs214168@mcore 5]$ ./philosopher -t 4
philosopher 0 is thinking
philosopher 1 is thinking
philosopher 2 is thinking
philosopher 3 is thinking
philosopher 4 is thinking
philosopher 0 is eating
philosopher 2 is eating
philosopher 0 is thinking
philosopher 2 is thinking
philosopher 1 is eating
philosopher 3 is eating
philosopher 1 is thinking
philosopher 3 is thinking
philosopher 4 is eating
philosopher 2 is eating
philosopher 4 is thinking
philosopher 2 is thinking
philosopher 1 is eating
philosopher 3 is eating
philosopher 1 is thinking
philosopher 3 is thinking
philosopher 0 is eating

[cs214168@mcore 5]$
```

实验要求不允许出现僵尸进程，这里我们使用 ps au 命令查看当前用户的进程。

运行 philosopher 时:

```
[cs214168@mcore ~]$ ps au
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root      2368  0.0  0.0   3816   508 tty1     Ss+  Dec05   0:00 /sbin/mingetty
root      3606  0.0  0.0   3816   504 tty2     Ss+  Nov30   0:00 /sbin/mingetty
root      3607  0.0  0.0   3816   504 tty3     Ss+  Nov30   0:00 /sbin/mingetty
root      3608  0.0  0.0   3816   504 tty4     Ss+  Nov30   0:00 /sbin/mingetty
root      3609  0.0  0.0   3816   508 tty5     Ss+  Nov30   0:00 /sbin/mingetty
root      3610  0.0  0.0   3816   504 tty6     Ss+  Nov30   0:00 /sbin/mingetty
cs214168  8136  0.0  0.0  66200  1692 pts/1    Ss   15:13   0:00 -bash
cs214200  8329  0.0  0.0  70476  1724 pts/2    Ss+  15:21   0:00 -bash
cs214168  8809  0.0  0.0  66096  1636 pts/3    Ss   15:50   0:00 -bash
cs214168  8871  0.0  0.0   3672   312 pts/1    S+   15:55   0:00 ./philosopher
cs214168  8872  0.0  0.0   3676   256 pts/1    S+   15:55   0:00 ./philosopher
cs214168  8873  0.0  0.0   3676   256 pts/1    S+   15:55   0:00 ./philosopher
cs214168  8874  0.0  0.0   3676   256 pts/1    S+   15:55   0:00 ./philosopher
cs214168  8875  0.0  0.0   3676   256 pts/1    S+   15:55   0:00 ./philosopher
cs214168  8876  0.0  0.0   3676   256 pts/1    S+   15:55   0:00 ./philosopher
cs214168  8878  0.0  0.0  65620  1004 pts/3    R+   15:55   0:00 ps au
[cs214168@mcore ~]$
```

可以看到有 6 个进程，包含 1 个父进程，5 个子进程。

停止 philosopher 后:

```
[cs214168@mcore ~]$ ps au
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root      2368  0.0  0.0   3816   508 tty1     Ss+  Dec05   0:00 /sbin/mingetty
root      3606  0.0  0.0   3816   504 tty2     Ss+  Nov30   0:00 /sbin/mingetty
root      3607  0.0  0.0   3816   504 tty3     Ss+  Nov30   0:00 /sbin/mingetty
root      3608  0.0  0.0   3816   504 tty4     Ss+  Nov30   0:00 /sbin/mingetty
root      3609  0.0  0.0   3816   508 tty5     Ss+  Nov30   0:00 /sbin/mingetty
root      3610  0.0  0.0   3816   504 tty6     Ss+  Nov30   0:00 /sbin/mingetty
cs214168  8136  0.0  0.0  66200  1692 pts/1    Ss+  15:13   0:00 -bash
cs214200  8329  0.0  0.0  70476  1724 pts/2    Ss+  15:21   0:00 -bash
cs214168  8809  0.0  0.0  66096  1636 pts/3    Ss   15:50   0:00 -bash
cs214168  8900  0.0  0.0  65620  1004 pts/3    R+   15:58   0:00 ps au
```

已经没有了 philosopher 进程，未出现僵尸进程。

四、体会和建议

一句话概括该实验就是利用文件来进行进程间的通信。代码量很少，重在理解。关键是理解 lock()函数的实现，该函数创建一个文件并将其“占为己有”，“告诉”其他进程不允许再创建该文件，这样便实现了进程间的通信。这次实验让我对进程同步和进程间的通信有了一定的了解，对后续进程同步异步的学习有所帮助。