Unix 实验报告

实验：	实验 5 哲学家进餐-进程版
专业：	计算机科学与技术
班级：	1 班
姓名：	姚怀聿
学号：	22920202204632

2022 年 12 月 2 日

目　　录

一、 实验内容描述

本实验的目的是学习和掌握并发进程同步的概念和方法。

实验要求编写程序 philosopher；

命令语法为:

./philosopher [-t <time>]

具体要实现的功能如下：

1.<time>是哲学界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.利用课堂已教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。

二、 实验构思

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，因此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，使用文件是可行的。

由于程序要生成多个进程，因此还需要使用到作业控制的机制。

如果哲学家中同时存在左撇子和右撇子，则哲学家问题有解。

具体实现如下:

(1) 定义 5 个文件，分别表示 5 个叉子:

```
static char *forks[N] = {"fork0", "fork1", "fork2", "fork3", "fork4"};
```

(2) 哲学家的行为用如下函数描述:

```
static void philosopher(int i, int time) {
    while(1) {
        thinking(i, time); // philosopher i think for second
        takefork(i); // philosopher i take the fork
        eating(i, time); // philosopher i eat for seconds
        putfork(i); // philosopher i put the fork
    }
}
```

(3) 在主程序里，用下面的程序段生成 5 个哲学家进程:

```
#define N 5
```

```
for(i = 0; i < N; i++) {
    fflush(stdout); // 每次fork前都需要清空缓冲区
    pid = fork();
    if(pid < 0) {
        perror("fork error: ");
        exit(1);
    } else if(pid == 0) { // child
        philosopher(i, seconds);
        exit(0);
    }
}


for(i = 0; i < N; i++) wait(NULL);
```

注意，最后父进程需要等待子进程的结束，为它“收尸”，否则它子进程将成为“僵尸进程”。

(4) 拿起叉子如下定义:

```
static void takefork(int i) {
    if(i == N - 1) { // if the last one, first take the left, then take the right
        lock(forks[N - 1]);
        lock(forks[0]);
    } else { // or not, first take the right, then take the left
        lock(forks[i + 1]);
        lock(forks[i]);
    }
}
```

(5) 放下叉子如下定义:

```
static void putfork(int i) {
    if(i == N - 1) {
        unlock(forks[N - 1]);
        unlock(forks[0]);
    } else {
        unlock(forks[i + 1]);
        unlock(forks[i]);
    }
}
```

完整代码如下：

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>
#include <time.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <string.h>
#include <sys/wait.h>
#define N 5

/*
 *  * default file access permissions for new files.
 *   * */
#define FILE_MODE (S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH)

static char *forks[N] = {"fork0", "fork1", "fork2", "fork3",
"fork4"};

static void initlock() {
    int i;
```

```
    for(i = 0; i < N; i++) unlink(forks[i]);
}

static void lock(const char *lockfile) {
    int fd;
    while((fd = open(lockfile, O_RDONLY|O_CREAT|O_EXCL, FILE_MODE)) <
0) sleep(1); // 每隔 1s 尝试一次，直到打开为止
    close(fd);
}

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


char *getTime() {
    char *time_now;
    time_now = (char*)malloc(sizeof(char) * 64);
    memset(time_now, 0, sizeof time_now);
    time_t tloc;
    struct tm *tm;
    time(&tloc);
    tm = localtime(&tloc);
    strftime(time_now, 63, "%H:%M:%S", tm);
    return time_now;
}

static void thinking(int i, int time) {
    fprintf(stdout, "Philosopher %d is thinking   %s\n", i,
getTime());
    sleep(time);
```

```
}

static void eating(int i, int time) {
    fprintf(stdout, "Philosopher %d is eating     %s\n", i,
getTime());
    sleep(time);
}

static void takefork(int i) {
    if(i == N - 1) { // if the last one, first take the left, then
take the right
        lock(forks[N - 1]);
        lock(forks[0]);
    } else { // or not, first take the right, then take the left
        lock(forks[i + 1]);
        lock(forks[i]);
    }
}

static void putfork(int i) {
    if(i == N - 1) {
        unlock(forks[N - 1]);
        unlock(forks[0]);
    } else {
        unlock(forks[i + 1]);
        unlock(forks[i]);
    }
}

static void philosopher(int i, int time) {
    while(1) {
```

```
        thinking(i, time); // philosopher i think for second
        takefork(i); // philosopher i take the fork
        eating(i, time); // philosopher i eat for seconds
        putfork(i); // philosopher i put the fork
    }
}

int main(int argc, char* argv[]) {
    initlock();
    int seconds, i;
    pid_t pid;
    if(argc == 1) {
        seconds = 2;
    } else if(argc == 3 && (strcmp(argv[1], "-t") == 0)) {
        seconds = atoi(argv[2]);
    } else {
        fprintf(stderr, "Usage: ./philosopher [-t <time>]\n");
        exit(1);
    }

    for(i = 0; i < N; i++) {
        fflush(stdout); // 每次 fork 前都需要清空缓冲区
        pid = fork();
        if(pid < 0) {
            perror("fork error: ");
            exit(1);
        } else if(pid == 0) { // child
            philosopher(i, seconds);
            exit(0);
        }
    }
```

```
    for(i = 0; i < N; i++) wait(NULL);

    exit(0);
}
```

char *getTime()函数用于打印当前时间，在程序中使用该函数能够更加直观的看到实验结果:

```
char *getTime() {
    char *time_now;
    time_now = (char*)malloc(sizeof(char) * 64);
    memset(time_now, 0, sizeof time_now);
    time_t tloc;
    struct tm *tm;
    time(&tloc);
    tm = localtime(&tloc);
    strftime(time_now, 63, "%H:%M:%S", tm);
    return time_now;
}
```

三、 实验结果

源程序名	可执行程序名
philosopher.c	philosopher

编译生成可执行文件：

使用如下指令：

`make philosopher`，即可得到可执行文件`philosopher`

```
rongrong@rongrong-virtual-machine:~/Desktop/huaiyuyao/unix/homework/homework_5$ make philosopher
cc     philosopher.c   -o philosopher
rongrong@rongrong-virtual-machine:~/Desktop/huaiyuyao/unix/homework/homework_5$ ls
philosopher  philosopher.c
```

运行程序：

```
rongrong@rongrong-virtual-machine:~/Desktop/huaiyuyao/unix/homework/homework_5$ ./philosopher
Philosopher 2 is thinking   15:22:58
Philosopher 1 is thinking   15:22:58
Philosopher 3 is thinking   15:22:58
Philosopher 0 is thinking   15:22:58
Philosopher 4 is thinking   15:22:58
Philosopher 2 is eating     15:23:00
Philosopher 0 is eating     15:23:00
Philosopher 2 is thinking   15:23:02
Philosopher 0 is thinking   15:23:02
Philosopher 1 is eating     15:23:02
Philosopher 3 is eating     15:23:02
Philosopher 1 is thinking   15:23:04
Philosopher 3 is thinking   15:23:04
Philosopher 4 is eating     15:23:04
Philosopher 2 is eating     15:23:05
Philosopher 4 is thinking   15:23:06
Philosopher 2 is thinking   15:23:07
Philosopher 3 is eating     15:23:07
Philosopher 0 is eating     15:23:07
^C
rongrong@rongrong-virtual-machine:~/Desktop/huaiyuyao/unix/homework/homework_5$
```

可以看到，该程序成功模拟了“五个哲学家问题”，解决了并发环境下，多进程之间的同步与互斥问题。

由于运行时使用的是时间的缺省值，故“哲学家们”每两秒中更换一个状态。可以将时间值更改再次尝试：

```
rongrong@rongrong-virtual-machine:~/Desktop/huaiyuyao/unix/homework/homework_5$ ./philosopher -t 10
Philosopher 1 is thinking   15:23:24
Philosopher 3 is thinking   15:23:24
Philosopher 4 is thinking   15:23:24
Philosopher 0 is thinking   15:23:24
Philosopher 2 is thinking   15:23:24
Philosopher 1 is eating     15:23:34
Philosopher 3 is eating     15:23:34
Philosopher 3 is thinking   15:23:44
Philosopher 1 is thinking   15:23:44
Philosopher 0 is eating     15:23:44
Philosopher 2 is eating     15:23:44
Philosopher 0 is thinking   15:23:54
Philosopher 2 is thinking   15:23:54
Philosopher 4 is eating     15:23:54
Philosopher 1 is eating     15:23:55
Philosopher 4 is thinking   15:24:04
Philosopher 1 is thinking   15:24:05
Philosopher 0 is eating     15:24:05
Philosopher 2 is eating     15:24:05
^C
rongrong@rongrong-virtual-machine:~/Desktop/huaiyuyao/unix/homework/homework_5$
```

可以看到，结果与预期相符。

当程序运行的时候，我们可以使用命令：

ps -auf 列出当前终端上启动的所有进程：

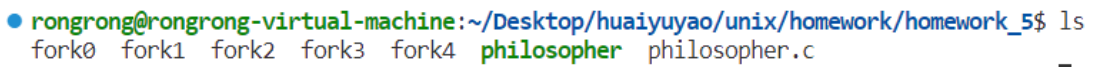
可以看到一个父进程同时产生了 5 个子进程。

使用 Ctrl + c 终止程序后，再次使用命令 ps -auf 列出当前终端上启动的所有进程：

可以看到，没有出现僵尸进程。

实验任务得以解决。

程序运行结束时，会在当前工作目录下间歇产生五个文件：



这是我们使用文件的方式实现该任务所产生的附加物。

四、 体会和建议

体会：通过本次实验，我对多进程之间的同步与互斥问题有了更深的了解。“纸上得来终觉浅，绝知此事要躬行”。上理论课时总觉得这一块理解得不透彻，只有当真正做实验的时候才感觉到自己有些理解多进程这一块的内容了。

建议：希望以后上机课之前，老师能提前说明实验课需要做什么，比如要用到哪些知识点，以及书上哪些程序对本次实验是比较有帮助的，我们可以通过看书本上的例程去更好的实现实验要求。以及可能要用到的不熟悉的函数能为我们做详细的解释。

五、 完成人姓名及完成时间

完成人姓名	完成时间
姚怀聿	2022 年 12 月 2 日