Unix程序设计

实验（五）哲学家就餐问题

姓　　名	熊恪峥
学　　号	22920202204622
日　　期	2022年12月10日
学　　院	信息学院
课程名称	Unix程序设计

实验（五）哲学家就餐问题

目录

1 实验内容

编制模拟“五个哲学家”问题的程序。程序语法：

```
philosopher [ -t <time >]
```

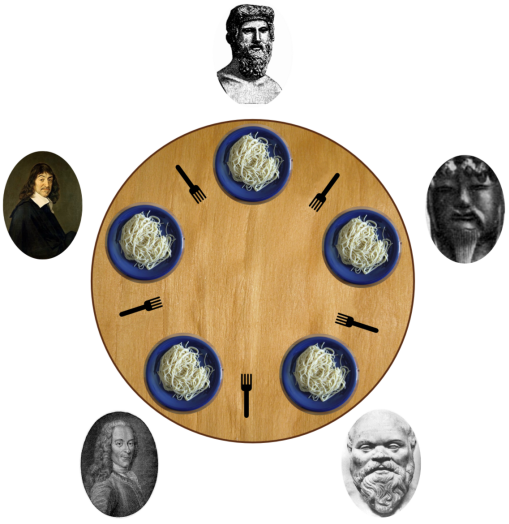
<time >是哲学家进餐和沉思的持续时间值，缺省值为2秒。

1. 五个哲学家的编号为0 · · · 4，分别用五个进程独立模拟。
2. 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。

2 使用文件同步的实现

哲学家就餐问题是一种用于描述资源同步和解决方案的问题 [?]。该问题描述了五个哲学家围坐在一张圆桌周围，每个哲学家面前都有一碗饭和一根筷子。哲学家们的生活方式是交替地思考和进餐，思考时不使用筷子，进餐时需要使用两根筷子。当一个哲学家进餐时，他需要同时拿起左右两边的筷子，如果两边的筷子都被其他哲学家拿着，那么他就需要等待，直到两边的筷子都空闲下来。当一个哲学家拿起一根筷子时，他就会把它放在桌子上，以示其他哲学家可以拿起它。当一个哲学家进餐完毕，他会把两根筷子放回桌子上，然后开始思考 [1]，如图 1所示。 [1]

图 1: 哲学家就餐问题



为了实现同步，有多种可以考虑的方法。使用文件是一种简单直接地方法。lock.c通过文件实现了一种互斥锁。如附录：代码清单中的代码 1所示。为了获取锁，程序就会使用O_EXCL标志尝试创建对应的文件。如果此时已经有进程拥有这把锁，就会导致创建失败，因此尝试获取锁的进程就会休眠一段时间再试。这样就实现了互斥锁的效果。为了释放锁，进程只需要删除对应的文件即可。

使用这种互斥锁的方法，可以实现哲学家就餐问题。如附录：代码清单中的代码 2所示。在这个程序中，每个哲学家都是一个进程，每个进程都会尝试获取左右两边的筷子，如果获取成功，就会进餐。在这个实现中，解决死锁问题的方法是“指定一个哲学家，它获取两把筷子的顺序和其他人相反。”。这种方法本质上是一种“资源分级”的解决方案。它相当于给筷子A、B、C、D、E赋予优先级，然后要求各哲学家都先获取优先级低的筷子，再获取优先级高的筷子。这样就可以避免死锁的发生。

[1]图片：https://en.wikipedia.org/wiki/Dining_philosophers_problem

这种方法能够避免死锁发生，但它不是公平的。尤其是当哲学家思考的时间不同的时候，可能思考时间较长的哲学家永远没有办法进餐。此外，利用文件实现互斥锁的方法也有一些缺点。首先，它需要创建文件，这样就会占用磁盘空间。文件I/O也是一种引入了额外开销的操作。

为了解决后者的问题，可以使用**信号量**。

3 使用信号量优化上述实现

信号量（Semaphores）是一种常见的同步原语（Primitives）。信号量是一个变量或者抽象数据类型，它的作用是用来控制多个线程或进程对共享资源的访问，避免并行系统中的关键资源被多个进程同时访问而导致的不一致性。信号量的值是一个非负整数，它可以表示共享资源的数量。POSIX API提供了创建可以在进程间共享的信号量的方法。

信号量的基本操作可以被称为P操作和V操作。P操作是“等待”操作，它会使信号量的值减1。如果信号量的值为0，那么P操作会使进程休眠，直到信号量的值大于0。V操作是“释放”操作，它会使信号量的值加1。如果有进程在等待信号量，那么V操作会唤醒一个等待进程。对应的POSIX API 如表 1所示。

表格 1: 信号量的POSIX API

函数	功能	返回值
sem_init	初始化信号量	0
sem_wait	P操作	0
sem_post	V操作	0
sem_destroy	销毁信号量	0

使用信号量的实现如附录：代码清单中的代码 3所示。

4 另一种解决方案

除了限定哲学家中同时存在左撇子和右撇子，还可以通过限制能吃饭的哲学家的数量来避免死锁 [2]。这种方法的思路是，当有哲学家想要进餐的时候，需要进入“餐厅”，餐厅仅限4个人，这样每人都有足够的叉子。如果餐厅已经满了，那么哲学家就需要等待。当有哲学家离开餐厅的时候，餐厅就会有空位，这时候可以让等待的哲学家进入餐厅。

这样，就可以用初始值为4的信号量来表示餐厅的空位数。当哲学家想要进入餐厅的时候，就需要执行P操作，如果餐厅有空位，那么P操作就会成功，哲学家就可以进入餐厅。当哲学家离开餐厅的时候，就需要执行V操作，这样餐厅的空位数就会加1，可以让等待的哲学家进入餐厅。

这样的实现如附录：代码清单中的代码 4所示。

5 运行效果

以上三种实现的运行效果都大致如图 2所示。 可见，所有哲学家都能正常进餐，这些实现都是正确的。

6 实验总结

本次实现通过解决哲学家就餐问题，让我了解到了多进程同步的基本概念和方法。通过本次实验，我学会了使用文件和信号量来实现进程同步，并解决哲学家就餐问题。了解到了在并发的系统中保护Critical Section的基本方法。

图 2: 哲学家就餐问题

```
-> % ./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 1 is eating
philosopher 2 is eating
philosopher 3 is eating
philosopher 4 is eating
philosopher 0 is thinking
philosopher 1 is thinking
philosopher 2 is thinking
philosopher 3 is thinking
philosopher 4 is thinking
```

参考文献

References

[1] Wikipedia. Dining philosophers problem — Wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=Dining%20philosophers%20problem&oldid=1120886101, 2022. [Online; accessed 10-December-2022].

[2] William Stallings and Goutam Kumar Paul. *Operating systems: internals and design principles*, volume 9. Pearson New York, 2012.

附录：代码清单

代码 1: lock.c

```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);
}
```

代码 2: 使用文件实现

```c
//
// Created by bear on 12/3/2022.
//
#include "apue.h"
#include "lock.h"
#include "error.h"

#include "sys/wait.h"

#define N 5

static char *forks[5] = {"fork0", "fork1", "fork2", "fork3", "fork4"};

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

```
25          }
26  }
27
28  void takeFork(int i)
29  {
30          if (i == N - 1)
31          {
32                  lock(forks[0]);
33                  lock(forks[i]);
34          }
35          else
36          {
37                  lock(forks[i]);
38                  lock(forks[i + 1]);
39          }
40  }
41
42  void eating(int i, int nsecs)
43  {
44          printf("philosopher %d is eating\n", i);
45          sleep(nsecs);
46  }
47
48  void thinking(int i, int nsecs)
49  {
50          printf("philosopher %d is thinking\n", i);
51          sleep(nsecs);
52  }
53
54  void philosopher(int i, int nsecs)
55  {
56          for (;;)
57          {
58                  thinking(i, nsecs);
59                  takeFork(i);
60                  eating(i, nsecs);
61                  putFork(i);
62          }
63  }
64
65  int main(int argc, char **argv)
66  {
67          int nsecs = 0;
68          if (argc != 3 && argc != 1)
69          {
70                  printf("usage: %s [-t <nsecs>]", argv[0]);
71          }
72
73          if (strcmp(argv[2], "-t") == 0)
74          {
75                  nsecs = atoi(argv[3]);
76          }
77          else
78          {
79                  nsecs = 2;
```

```
80          }
81
82          for (int i = 0; i < N; i++)
83          {
84                  pid_t pid = fork();
85                  if (pid == 0)
86                  {
87                          philosopher(i, nsecs);
88                  }
89          }
90
91          wait(NULL);
92          return 0;
93  }
```

代码 3: 使用信号量实现

```
1   #define N 5
2
3   static sem_t forks_sems[5];
4
5   void init_sems()
6   {
7           for (int i = 0; i < N; i++)
8           {
9                   if (sem_init(&forks_sems[i], 1, 1) != 0)
10                  {
11                          err_sys("sem_open error");
12                  }
13          }
14  }
15
16  void destroy_sems()
17  {
18          for (int i = 0; i < N; i++)
19          {
20                  sem_destroy(&forks_sems[i]);
21          }
22  }
23
24  void putFork(int i)
25  {
26          if (i == N - 1)
27          {
28                  sem_post(&forks_sems[0]);
29                  sem_post(&forks_sems[i]);
30          }
31          else
32          {
33                  sem_post(&forks_sems[i]);
34                  sem_post(&forks_sems[i + 1]);
35          }
36  }
37
38  void takeFork(int i)
```

```
{
        if (i == N - 1)
        {
                sem_wait(&forks_sems[0]);
                sem_wait(&forks_sems[i]);
        }
        else
        {
                sem_wait(&forks_sems[i]);
                sem_wait(&forks_sems[i + 1]);
        }
}

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

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

void philosopher(int i, int nsecs)
{
        for (;;)
        {
                thinking(i, nsecs);
                takeFork(i);
                eating(i, nsecs);
                putFork(i);
        }
}
```

代码 4: 另一种实现

```
        #define N 5

static sem_t forks_sems[5];
static sem_t room_sem;

void init_sems()
{
        sem_init(&room_sem, 1, N - 1);
        for (int i = 0; i < N; i++)
        {
                if (sem_init(&forks_sems[i], 1, 1) != 0)
                {
                        err_sys("sem_open error");
                }
        }
}

```

```
18 void destroy_sems()
19 {
20         sem_destroy(&room_sem);
21         for (int i = 0; i < N; i++)
22         {
23                 sem_destroy(&forks_sems[i]);
24         }
25 }
26
27 void putFork(int i)
28 {
29         if (i == N - 1)
30         {
31                 sem_post(&forks_sems[0]);
32                 sem_post(&forks_sems[i]);
33         }
34         else
35         {
36                 sem_post(&forks_sems[i]);
37                 sem_post(&forks_sems[i + 1]);
38         }
39 }
40
41 void takeFork(int i)
42 {
43         if (i == N - 1)
44         {
45                 sem_wait(&forks_sems[0]);
46                 sem_wait(&forks_sems[i]);
47         }
48         else
49         {
50                 sem_wait(&forks_sems[i]);
51                 sem_wait(&forks_sems[i + 1]);
52         }
53 }
54
55 void eating(int i, int nsecs)
56 {
57         printf("philosopher %d is eating\n", i);
58         sleep(nsecs);
59 }
60
61 void thinking(int i, int nsecs)
62 {
63         printf("philosopher %d is thinking\n", i);
64         sleep(nsecs);
65 }
66
67 void philosopher(int i, int nsecs)
68 {
69         for (;;)
70         {
71                 thinking(i, nsecs);
72                 sem_wait(&room_sem);
```

```
73                  takeFork(i);
74                  eating(i, nsecs);
75                  putFork(i);
76                  sem_post(&room_sem);
77          }
78  }
```