华北水利水电大学

North China University of Water Resources and Electric Power

《操作系统》实验报告

实验三

Linux 进程的同步与互斥

院　　系　信息工程学院

专　　业　人工智能

学　　号　202018526

姓　　名　高树林

指 导 教 师　杨学颖

完成时间：2022-12-23

实验目的

1. 了解和掌握多线程编程
2. 了解和掌握信号量的应用
3. 了解和掌握资源顺序分配法

实验要求

1. 程序 3.c 使用 2 个线程计算从 1 到 200 的累加和，请将其改为用 3 个线程实现， 3 个线程分别完成函数 p1、p2 和 p3 的计算任务，总体功能不变。
2. 程序 4.c 模拟了如下场景：某休息厅里有足够多（10 把以上）的椅子，10 位顾客先后进入休息厅寻找空椅子，找到后开始在椅子上休息，休息完后让出空椅子、退出休息厅。请只在该程序中插入一些代码（不删除现有代码），来将上述场景调整为休息厅里只有 2 把椅子。
3. 程序 9.c 实现了哲学家就餐问题，里面存在死锁问题，请用资源顺序分配法改写该程序，避免死锁。

实验步骤

步骤 1：创建 p3 进程实现 p1 进程和 p2 进程的结果相加运算，实现的核心代码如下所示。

```
void p3(){
 printf("sum: %d\n",sum1 + sum2);
}
```

步骤 2：根据题目要求可知，椅子是临界资源，由题可知，本体的临界资源数量是两个，因此只需要把源代码中的临界资源数量做修改为 2 即可。

步骤 3：避免死锁的方法就是破坏死锁的四个条件之一就能达到避免死锁的效果。在哲学家就餐问题中，只需让最后一后一个哲学家不就餐就能解决死锁问题。代码如下

```
if(left<right){
    sem_wait(&chopstick[left]);
    delay();
    sem_wait(&chopstick[right]);
   }
else
   {
```

```
8.     sem_wait(&chostick[right]);
9.     delay();
10.    sem_wait(&chostick[left])
11.  }
```

实验结果

结果 1： 步骤 1 多线程编程完成加法计算得到的结果如下图 1 所示。

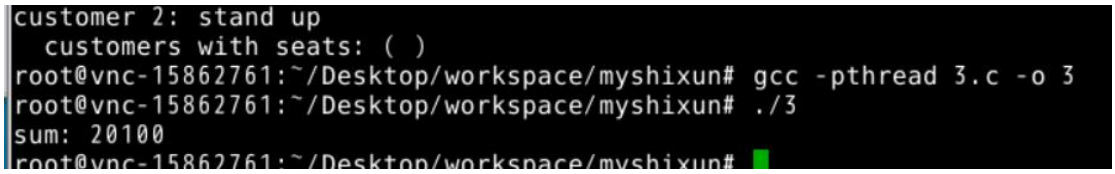

图 1 步骤 1 计算得到的结果

结果 2：步骤 2 信号量的应用调整椅子数目的结果如下图 2 所示。

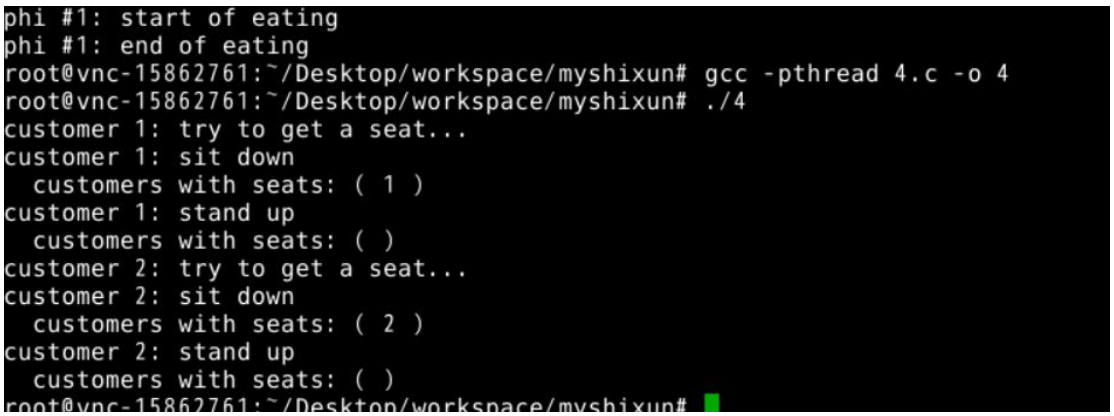

图 2 信号量的应用减少临界资源

结果 3：步骤 3 资源顺序分配法解决哲学家死锁的结果如下图 3 所示。

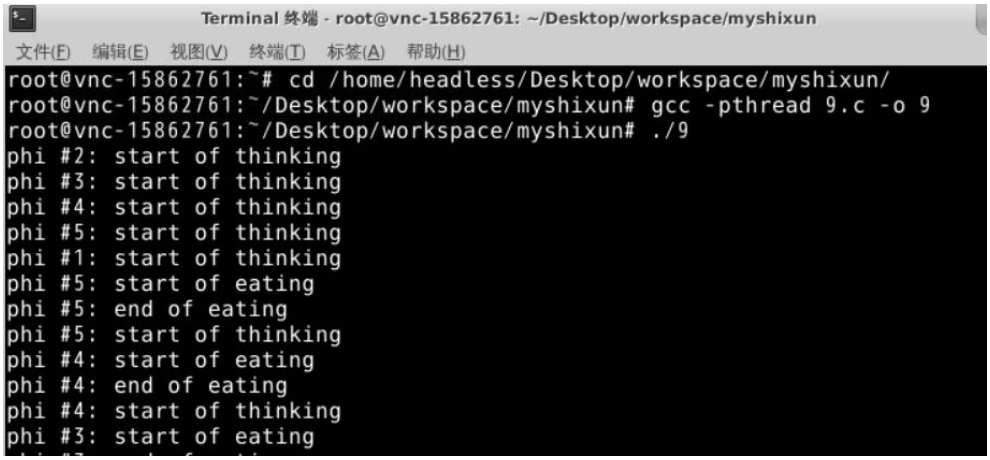

图 3 资源顺序分配法解决哲学家就餐死锁问题

实验总结

通过本次实验，我学到了通过编写程序实现进程同步和互斥，掌握了有关进程(线程)同步与互斥的原理以及解决进程(线程)同步和互斥的算法，从而进一步巩固进程(线程)同步和互斥等有关的内容。

同时我对造成死锁的四个必要条件有了更为深刻的理解，对于如何避免死锁的方法也有了极为深刻的了解。只有当资源互斥访问、线程又处于不可剥夺的状态、并且维持着请求和保持条件，又在循环等待，只有上述四点全部满足，这个时候才有可能达到死锁状态，而避免死锁，也就是破坏上述四个必要条件中的任何一个即可。

代码：

多线程编程：

```
1.  #include <stdio.h>
2.  #include <stdlib.h>
3.  #include <pthread.h>
4.
5.  int sum1 = 0, sum2 = 0;
6.
7.  void *p1(){
8.   int i, tmp = 0;
9.   for (i = 1; i <= 100; i++)
10.   tmp += i;
11.  sum1 += tmp;
12. }
13.
14. void *p2(){
15.  int i, tmp = 0;
16.  for (i = 101; i <= 200; i++)
17.   tmp += i;
18.  sum2 += tmp;
19. }
20.
21. void p3(){
22.  printf("sum: %d\n", sum1 + sum2);
23. }
24.
25. int main(){
26.  int res;
27.  pthread_t t1;
28.  pthread_t t2;
29.  void *thread_result;
30.  res = pthread_create(&t1, NULL, p1, NULL);
31.  if (res != 0){
32.   perror("failed to create thread");
33.   exit(1);
34.  }
35.  res = pthread_create(&t2, NULL, p2, NULL);
36.  if (res != 0){
37.   perror("failed to join thread");
38.   exit(2);
39.  }
40.  res = pthread_join(t1, &thread_result);
41.  res = pthread_join(t2, &thread_result);
```

```
42.
43.  p3();
44.  return 0;
45. }
```

信号量的应用：

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <unistd.h>
#include <pthread.h>
#include <semaphore.h>

#define CUSTOMER_NUM  2

int customer_state[CUSTOMER_NUM] = {0};

void sleep_random(int t) {
 sleep((int)(t * (rand() / (RAND_MAX *1.0))));
}

void print_cur_state() {
 int i;
 printf("  customers with seats: (");
 for (i = 0; i < CUSTOMER_NUM; i++)
 {
  if (0 != customer_state[i])
   printf(" %d", i+1);
 }
 printf(" )\n");
}

void *customer(void *id)
{
 const int myid = *(int*)id;
 sleep_random(2);
 printf("customer %d: try to get a seat...\n", myid);

 printf("customer %d: sit down\n", myid);
 customer_state[myid-1] = 1;
 print_cur_state();

 sleep_random(3);

 printf("customer %d: stand up\n", myid);
 customer_state[myid-1] = 0;
 print_cur_state();
}

```

```
int main()
{
 int i, id[CUSTOMER_NUM], res;
 pthread_t t[CUSTOMER_NUM];

 srand((int)time(0));

 for (i = 0; i < CUSTOMER_NUM; i++)
 {
  id[i] = i + 1;
  pthread_create(&t[i], NULL, customer, &id[i]);
 }
 for (i = 0; i < CUSTOMER_NUM; i++)
 {
  res = pthread_join(t[i], NULL);
  if (res != 0){
   perror("failed to join thread");
   exit(2);
  }
 }
 return 0;
}
```

资源顺序分配法：

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <unistd.h>
#include <pthread.h>
#include <semaphore.h>

void sleep_random(int t) {
 sleep((int)(t * (rand() / (RAND_MAX *1.0))));
}

void delay(){
 int i = 10000000;
 while (i--)
  ;
}

#define N  5
sem_t chopstick[N];

void *phi(void *id){  /* 'id' starts from 1 */
 int i, left, right, myid = *(int*)id;
 left = myid - 1;
 right = (myid < N) ? myid : 0;
 for (i = 0; i < 3; i++){
  printf("phi #%d: start of thinking\n", myid);
/* start */
  if(left<right){
   sem_wait(&chopstick[left]);
   delay();
   sem_wait(&chopstick[right]);
  }
  else
  {
   sem_wait(&chopstick[right]);
   delay();
   sem_wait(&chopstick[left]);
  }
/* end */
  printf("phi #%d: start of eating\n", myid);
  sleep_random(3);
  sem_post(&chopstick[left]);
  sem_post(&chopstick[right]);
```

```
44.   printf("phi #%d: end of eating\n", myid);
45.  }
46. }
47.
48. int main(){
49.  int i, id[N];
50.  pthread_t t[N];
51.  srand((int)time(0));
52.  for (i = 0; i < N; i++){
53.   id[i] = i + 1;
54.   sem_init(&chopstick[i], 0, 1);
55.  }
56.  for (i = 0; i < N; i++)
57.   pthread_create(&t[i], NULL, phi, &id[i]);
58.  for (i = 0; i < N; i++)
59.   pthread_join(t[i], NULL);
60.  return 0;
61. }
```