**第10组**

**小组成员：**明文凱、张雷

**完成关卡：**2、3、4

**关卡任务：**2)信号量的应用；3)资源顺序分配法；4)生成者与消费者问题

**第2关 信号量的应用**

**关卡任务：**

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。

**源代码：**

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <unistd.h>

#include <pthread.h>

#include <semaphore.h>

#define CUSTOMER\_NUM  10

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);

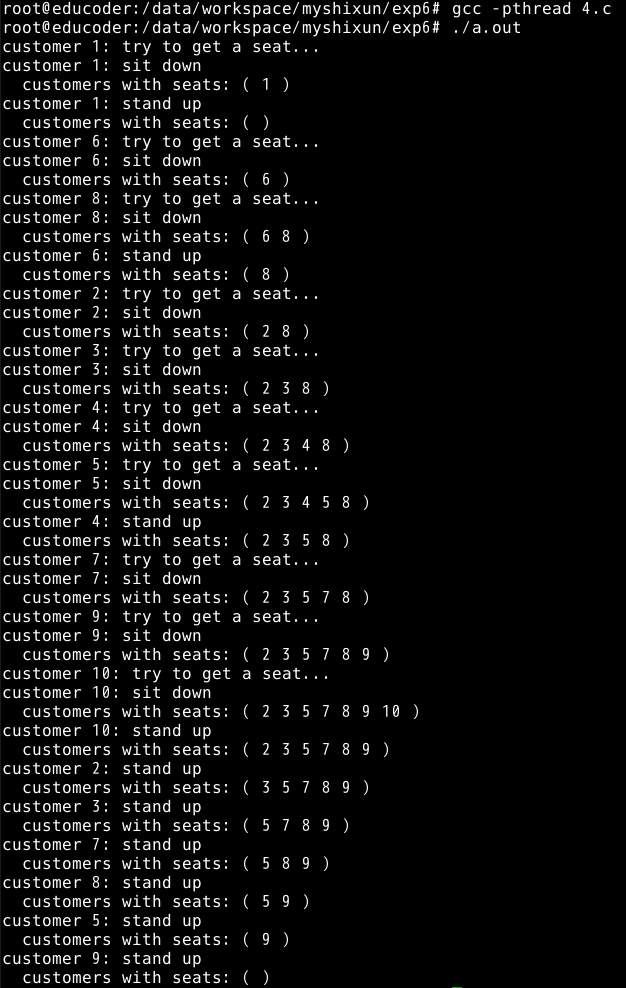
    }

  }

  return 0;

}

**源代码结果：**



**平台提供的参考代码主要完成的功能：**

创建顾客、空椅子，但由于一开始设定为10把椅子10位顾客，所以没有考虑椅子状态，所有的客人都有椅子可以做。修改后增加椅子状态，也就是信号量，当椅子减少到2把的时候，椅子被顾客占用时，其他顾客就需要等待椅子空闲，所以增加sem\_t s。以及sem\_init()、sem\_wait()、sem\_post()对椅子的状态进行设定与修改。

**修改部分的代码：**

int customer\_state[CUSTOMER\_NUM] = {0};

sem\_t s;//12行

  printf("customer %d: try to get a seat...\n", myid);

  sme\_wait(&s);//31行

  printf("customer %d: sit down\n", myid);

  sleep\_random(3);

  sem\_post(&s);//37行

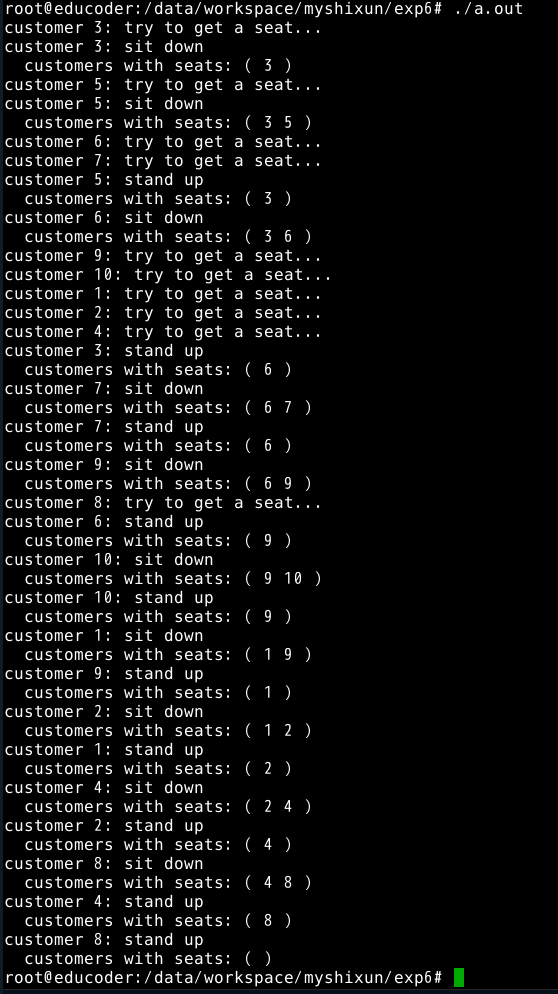
  printf("customer %d: stand up\n", myid);

  pthread\_t t[CUSTOMER\_NUM];

  sem\_init(&s,0,2);//46行

  srand((int)time(0));

**修改后代码的运行结果：**



**修改思路：**

增加信号量sem\_t s，对进程进行加锁，只能存在两个进程，也就是只有两把椅子可以使用，其他顾客需要等待椅子空了才能入座。通过sem\_init()、sem\_wait()、sem\_post()对椅子的状态进行设定与修改。

**第3关 资源顺序分配法**

**关卡任务：**

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。

**源代码：**

#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 \*\*\*\*\*\*\*\*\*\*\*\*\*/

      sem\_wait(&chopstick[left]);

      delay();

      sem\_wait(&chopstick[right]);

/\*\*\*\* end \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

    printf("phi #%d: start of eating\n", myid);

    sleep\_random(3);

    sem\_post(&chopstick[right]);

    sem\_post(&chopstick[left]);

    printf("phi #%d: end of eating\n", myid);

  }

}

int main(){

  int i, id[N];

  pthread\_t t[N];

  srand((int)time(0));

  for (i = 0; i < N; i++){

    id[i] = i + 1;

    sem\_init(&chopstick[i], 0, 1);

  }

  for (i = 0; i < N; i++)

    pthread\_create(&t[i], NULL, phi, &id[i]);

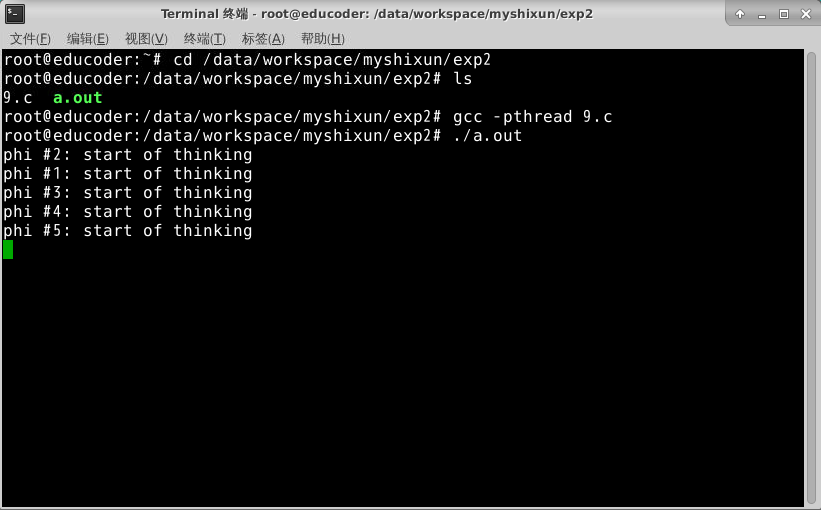
  for (i = 0; i < N; i++)

    pthread\_join(t[i], NULL);

  return 0;

}

**源代码结果：**



**平台提供的参考代码主要完成的功能：**

创建筷子与思考者，实现了思考者思考、进餐等流程，但是没有考虑所有思考者同时拿起左筷子导致死锁问题。

**修改后代码：**

/\*\*\*\* start \*\*\*\*\*\*\*\*\*\*\*\*\*/

if(myid%2==0)

      {

        sem\_wait(&chopstick[left]);

        delay();

        sem\_wait(&chopstick[right]);

      }else

      {

        sem\_wait(&chopstick[right]);

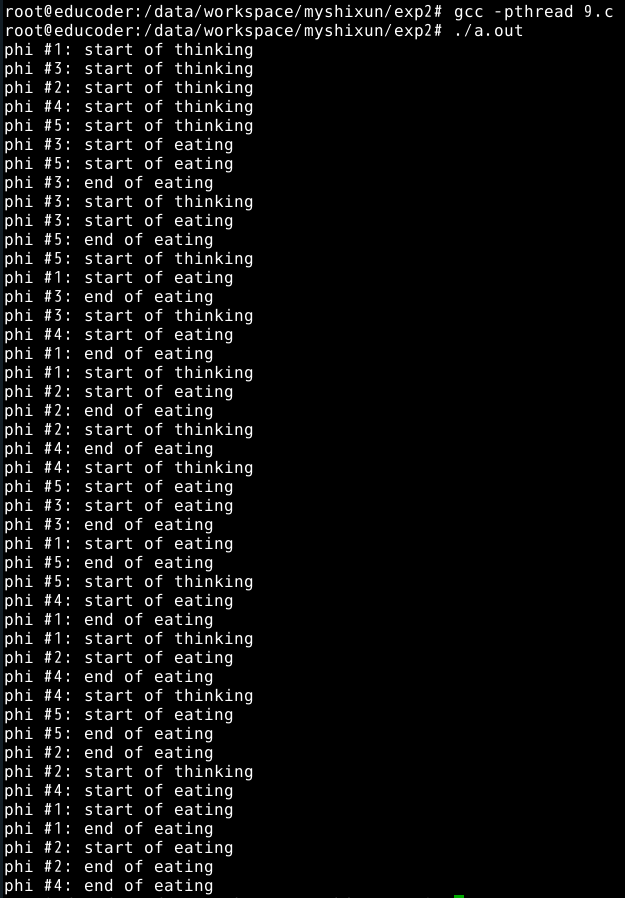
        delay();

        sem\_wait(&chopstick[left]);

      }

/\*\*\*\* end \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**修改后代码的运行结果：**



**修改思路：**

对思考者的编号进行分类，对于编号为偶数的思考者先拿左筷子，对于编号为奇数的思考者先拿右筷子。在原有代码的基础上增加奇偶数的判断if(myid%2==0)。以上是按照题目要求对代码进行修改，在不考虑资源顺序分配时可以考虑加入一个信号量，既房间只允许4人同时进餐，加入sem\_t room,限制同时只能出现4个进程，第五个人需要等待。这也是一种解法。

**第四关 生产者与消费者问题**

**关卡任务：**

程序 4-1.c 模拟了 1 个生产者和 1 个消费者，请改写该程序，模拟 5 个生产者和 5 个消费者，它们共享一个包含 8 个缓冲区的缓冲池。

**源代码：**

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <unistd.h>

#include <pthread.h>

#include <semaphore.h>

#define N  10

#define PRODUCT\_NUM 15

int buffer[N], readpos = 0, writepos = 0;

sem\_t full, empty;

void sleep\_random(int t) {

  sleep((int)(t \* (rand() / (RAND\_MAX \*1.0))));

}

void \*produce(){

  int i;

  for (i = 0; i < PRODUCT\_NUM; i++){

    sleep\_random(2);

  sem\_wait(&empty);

    buffer[writepos++] = i + 1;

    if (writepos >= N)

      writepos = 0;

    printf("produce:    %d\n", i+1);

    sem\_post(&full);

  }

}

void \*consume(){

  int i;

  for (i = 0; i < PRODUCT\_NUM; i++){

    sleep\_random(2);

    sem\_wait(&full);

    printf("consume: %d\n", buffer[readpos]);

    buffer[readpos++] =  - 1;

    if (readpos >= N)

      readpos = 0;

    sem\_post(&empty);

  }

}

int main(){

  int res, i;

  pthread\_t t1;

  for (i = 0; i < N; i++)

    buffer[i] =  - 1;

  srand((int)time(0));

  sem\_init(&full, 0, 0);

  sem\_init(&empty, 0, N);

  res = pthread\_create(&t1, NULL, produce, NULL);

  if (res != 0){

    perror("failed to create thread");

    exit(1);

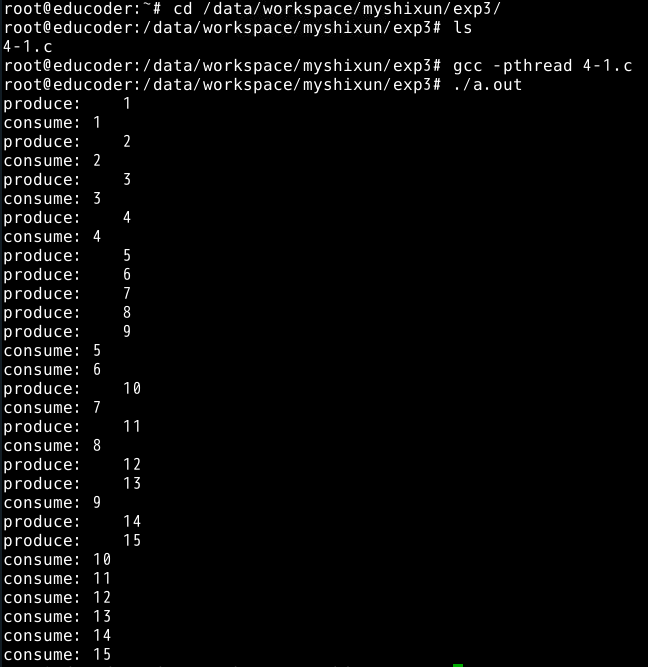
  }

  consume();

  return 0;

}

**源代码结果：**



**平台提供的参考代码主要完成的功能：**

源代码实现了单独一个生产者和一个消费者的操作流程，并给出了10个缓冲区，实现15次的生产和消费。同时设置了互斥锁，防止缓冲区数据被覆盖。

**修改后的源代码：**

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <unistd.h>

#include <pthread.h>

#include <semaphore.h>

#define N  8//修改缓冲区

#define PRODUCT\_NUM 15

int buffer[N], readpos = 0, writepos = 0;

sem\_t full, empty,pmutex,cmutex;//增加信号量

void sleep\_random(int t) {

  sleep((int)(t \* (rand() / (RAND\_MAX \*1.0))));

}

void \*produce(void \*id){

  int i;

  for (i = 0; i < PRODUCT\_NUM; i++){

    sleep\_random(2);

    sem\_wait(&empty);// 等待缓冲区有空闲位置

    sem\_wait(&pmutex);// 保证在product时不会有其他线程访问缓冲区

    int myid = \*(int \*)id \* 1000 + i + 1;

    buffer[writepos++] = myid;// 将新资源放到buffer[writepos]位置

    if (writepos >= N)

      writepos = 0;

    printf("produce:    %d\n", myid);// 唤醒的顺序可以不同

    sem\_post(&pmutex);// 通知consumer缓冲区有资源可以取走

    sem\_post(&full);

  }

}

void \*consume(){

  int i;

  for (i = 0; i < PRODUCT\_NUM; i++){

    sleep\_random(2);

    sem\_wait(&full);// 等待缓冲区有资源可以使用

    sem\_wait(&cmutex);// 保证在consume时不会有其他线程访问缓冲区

    printf("consume: %d\n", buffer[readpos]);// 将buffer[readpos]位置的资源取出

    buffer[readpos++] = - 1;// 重置buffer[out]位置的资源

    if (readpos >= N)

      readpos = 0;

    sem\_post(&cmutex);// 唤醒的顺序可以不同

    sem\_post(&empty);// 通知缓冲区有空闲位置

  }

}

int main(){

  int res, i,id[5];

  pthread\_t t1[5];

  pthread\_t t2[5];

  for (i = 0; i < N; i++)

    buffer[i] = i - 1;

  srand((int)time(0));

  sem\_init(&full, 0, 0);

  sem\_init(&empty, 0, N);

  sem\_init(&pmutex, 0, 1);

  sem\_init(&cmutex, 0, 1);

  //\*\*\*\*\*\*\*\*\*\*

  for(i = 0;i < 5;i++)

  {

    id[i]=i;

    pthread\_create(&t1[i], NULL, produce, &id[i]);

  }

  for(i = 0; i < 5; i++)

    pthread\_create(&t2[i],NULL,consume, &id[i]);

  for(i = 0;i < 5; i++)

  {

    pthread\_join(t1[i],NULL);

    pthread\_join(t2[i],NULL);

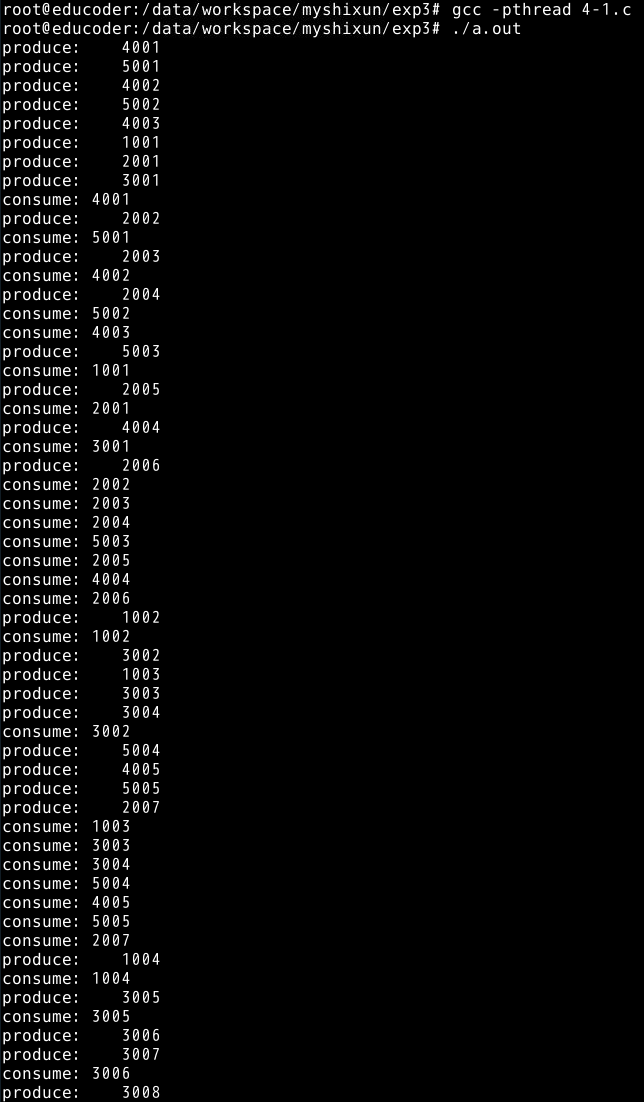
  }

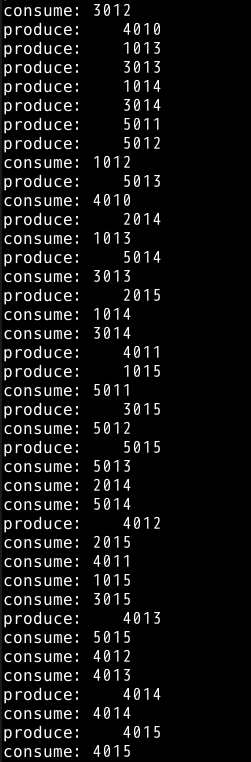
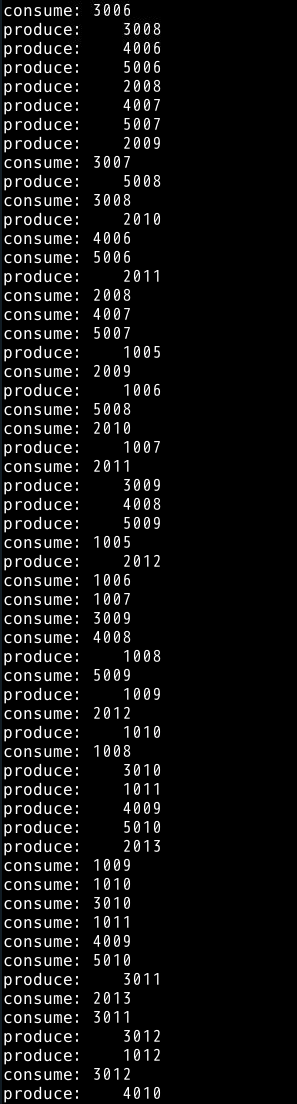
//\*\*\*\*\*\*\*\*\*\*

  return 0;

}

**修改后结果：**





**修改思路：**

首先将生产者和消费者增加至5个，给线程增加循环，把缓冲区修改为8个，然后添加临界区的锁，保证在product或者consume时不会有其他线程访问缓冲区。另外，在product中按照要求更改产品编号，使用void\*对id进行传参，在计算时需要使用int\*进行强制转换后才能计算。