CS305 作業系統概論

Prog. #3 Task Coordination 說明報告

汪文豪(學號：1071710)

- **如何編譯與測試操作程式：**

1. 編譯：g++ 檔名.cpp –o 檔名 –lrt -pthread
2. 執行：./檔名 m n, 欲讀入資料檔案之檔名 以及 m:演算法的模式 1-3

   n: 人數 3-11 個(threads)，倘若超出範圍，會報錯並且請您重新執行檔案
3. 輸出結果

範例(選擇演算法模式為 1、人數為 11 人)：

一. g++ RE_1071710_03.cpp –o trying –lrt -pthread

二. ./trying 1 11

三. 輸出結果

```
ryan@ubuntu:~/Desktop$ g++ RE_1071710_03.cpp -o trying -lrt -pthread
ryan@ubuntu:~/Desktop$ ./trying 1 11
*****This is algorithm-1 case*****
07:55:07-Phi 2-Im'in!
07:55:07-Phi 4-Im'in!
07:55:07-Phi 5-Im'in!
07:55:07-Phi 6-Im'in!
07:55:07-Phi 8-Im'in!
07:55:07-Phi 10-Im'in!
07:55:07-Phi 3-Im'in!
07:55:07-Phi 1-Im'in!
07:55:07-Phi 7-Im'in!
```

➢ 設計理念：

本次使用到的特殊函式庫

```
#include<stdlib.h>
#include<stdio.h>
#include<time.h> // about system wall-clock time
#include<iostream>
#include<pthread.h> // pthread API
#include<unistd.h> // get tid
```

本次程式說明：

✓ 完成部分：

1. [基礎] 正確產生模擬所有哲學家的 threads 者，最多可得 20 分。

```
23  struct parameter{
24      int number;
25      int algorithms;
26      int total_nums;
27  };
```

```
228     int m,n=0;
229     m = stoi(argv[1]);
230     n = stoi(argv[2]);
231     if(m<1 || m>3 || n<3 || n>11)
232     {
233         cout<<"outrange error... program exit.";
234         return 0;
235     }
236     cout<<"*****This is algorithm-"<<m<<" case*****"<<"\n";
237     pthread_t work_thread[n];
238     parameter args[n];
239     /*set work thread*/
240     for(int i=0;i<n;i++)
241     {
242         args[i].number = i+1;
243         args[i].algorithms=m;
244         args[i].total_nums=n;
245         if(pthread_create(&work_thread[i],NULL,algorithm,(void*)&args[i])){
246                 cerr<<"error occurred by work"<<i<<"\n";
247         };
248     }
249     for (int i = 0 ;i<n;i++)
250         pthread_join(work_thread[i], NULL);
251     return 0;
252 };
253
```

Parameter struct 儲存著

- number => 該哲學家的編號
- algorithms => 1 – 3 哪一種演算法
- total_nums => 3-11 人的總人數，用來計算筷子用

為求程式方便進行，因此多開一個空間，讓 0 的地方空著，以 1 開始起數。

可依照需求正確產生 threads 的數量

2. **[基礎] 亂數的產生必須在主程式一開始就用 srand(0) 的方式設定亂數種子，每次執行都會產生相同的亂數序列。(助教會修改 seed 值，進行測試)，最多可得 10 分。**

```
225  int main(int argc,char* argv[])
226  {
227      srand(0); //set random seeds
228      int m,n=0;
229      m = stoi(argv[1]);
230      n = stoi(argv[2]);
```

227 行設置 random seeds.

3. **正確使用 pthread API 中的 mutex 機制形成 critical section 來設計，最多可得 10 分。不使用此機制者， 此部分 0 分，以下各演算法的實作也不計分。**

```
using namespace std;
pthread_mutex_t chopstick[12]={
  PTHREAD_MUTEX_INITIALIZER,
  PTHREAD_MUTEX_INITIALIZER,
  PTHREAD_MUTEX_INITIALIZER,
  PTHREAD_MUTEX_INITIALIZER,
  PTHREAD_MUTEX_INITIALIZER,
  PTHREAD_MUTEX_INITIALIZER,
  PTHREAD_MUTEX_INITIALIZER,
  PTHREAD_MUTEX_INITIALIZER,
  PTHREAD_MUTEX_INITIALIZER,
  PTHREAD_MUTEX_INITIALIZER,
  PTHREAD_MUTEX_INITIALIZER,
  PTHREAD_MUTEX_INITIALIZER}; // mutex4thread
pthread_mutex_t enter = PTHREAD_MUTEX_INITIALIZER;
pthread_mutex_t output = PTHREAD_MUTEX_INITIALIZER;
```

使用 pthread 作為筷子的設計。

4. **各個演算法正確執行，可得以下分數：**

A. **基本演算法：完成本演算法者，最多可得 20 分。**

```
switch(mode)
{
    case 1: //algorithm 1
    for(int i=0;i<10;i++){
        think_time = rand()%(10-5+1)+5; eat_time = rand()%(5-1+1)+1;
        pthread_mutex_lock(&chopstick[left]);//take left chopstick.
        now = time(NULL);
        strftime(time_now,100,"%H:%M:%S",localtime(&now));
        pthread_mutex_lock(&output);
        cout<<time_now<<"-Phi "<<phi_num<<"-taking left chopstick"<<"\n";
        pthread_mutex_unlock(&output);
        pthread_mutex_lock(&chopstick[right]);//take right chopstick.
        now = time(NULL);
        strftime(time_now,100,"%H:%M:%S",localtime(&now));
        pthread_mutex_lock(&output);
        cout<<time_now<<"-Phi "<<phi_num<<"-taking right chopstick"<<"\n";
        pthread_mutex_unlock(&output);
        //get right & left chopstick
        now = time(NULL);
        strftime(time_now,100,"%H:%M:%S",localtime(&now));
        pthread_mutex_lock(&output);
        cout<<time_now<<"-Phi "<<phi_num<<"-eating"<<"\n";
        pthread_mutex_unlock(&output);
        sleep(eat_time);//eating.... then put down the chopstick.
        now = time(NULL);
        strftime(time_now,100,"%H:%M:%S",localtime(&now));
        pthread_mutex_lock(&output);
        cout<<time_now<<"-Phi "<<phi_num<<"-putting left chopstick"<<"\n";
        pthread_mutex_unlock(&output);
        pthread_mutex_unlock(&chopstick[left]);
        now = time(NULL);
        strftime(time_now,100,"%H:%M:%S",localtime(&now));
        pthread_mutex_lock(&output);
        cout<<time_now<<"-Phi "<<phi_num<<"-putting right chopstick"<<"\n";
        pthread_mutex_unlock(&output);
        pthread_mutex_unlock(&chopstick[right]);
        now = time(NULL);
        strftime(time_now,100,"%H:%M:%S",localtime(&now));
        cout<<time_now<<"-Phi "<<phi_num<<"-thinking"<<"\n";
        sleep(think_time);
    }
```

基本演算法，將哲學家的左右手邊的筷子的鎖拿到手後即可開始吃飯，吃 1-5 秒，之後放下筷子進行思考 5-10 秒，其中 print 出時用 output 上鎖，以免印出畫面時 thread 彼此衝突。 For 迴圈設定為 10 次，吃完 10 次並思考後結束並印出 thread leave 的時間

B. **Asymmetric 演算法：完成本演算法者，最多可得 20 分。**

大致上與上一個差不多，但多加一個限制，若是奇數哲學家，先拿左手邊的筷子，再拿右手邊的筷子

```
case 2:
for(int i=0;i<10;i++){
    think_time = rand()%(10-5+1)+5; eat_time = rand()%(5-1+1)+1;
    if(phi_num % 2 != 0){ //odd first pick left then pick right
        pthread_mutex_lock(&chopstick[left]);//take left chopstick.
        now = time(NULL);
        strftime(time_now,100,"%H:%M:%S",localtime(&now));
        pthread_mutex_lock(&output);
        cout<<time_now<<"-Phi "<<phi_num<<"-taking left chopstick"<<"\n";
        pthread_mutex_unlock(&output);
        pthread_mutex_lock(&chopstick[right]);//take right chopstick.
        now = time(NULL);
        strftime(time_now,100,"%H:%M:%S",localtime(&now));
        pthread_mutex_lock(&output);
        cout<<time_now<<"-Phi "<<phi_num<<"-taking right chopstick"<<"\n";
        pthread_mutex_unlock(&output);
        //get right & left chopstick
        now = time(NULL);
        strftime(time_now,100,"%H:%M:%S",localtime(&now));
        pthread_mutex_lock(&output);
        cout<<time_now<<"-Phi "<<phi_num<<"-eating"<<"\n";
        pthread_mutex_unlock(&output);
        sleep(eat_time);//eating.... then put down the chopstick.
        now = time(NULL);
        strftime(time_now,100,"%H:%M:%S",localtime(&now));
        pthread_mutex_lock(&output);
        cout<<time_now<<"-Phi "<<phi_num<<"-putting left chopstick"<<"\n";
        pthread_mutex_unlock(&output);
        pthread_mutex_unlock(&chopstick[left]);
        now = time(NULL);
        strftime(time_now,100,"%H:%M:%S",localtime(&now));
        pthread_mutex_lock(&output);
        cout<<time_now<<"-Phi "<<phi_num<<"-putting right chopstick"<<"\n";
        pthread_mutex_unlock(&output);
        pthread_mutex_unlock(&chopstick[right]);
        now = time(NULL);
        strftime(time_now,100,"%H:%M:%S",localtime(&now));
        cout<<time_now<<"-Phi "<<phi_num<<"-thinking"<<"\n";
        sleep(think_time);
    }
    else{  //even first pick right then pick left
```

而偶數則相反，先拿右手邊的筷子再拿左手

```
    }
    else{  //even first pick right then pick left
        pthread_mutex_lock(&chopstick[right]);//take right chopstick.
        now = time(NULL);
        strftime(time_now,100,"%H:%M:%S",localtime(&now));
        pthread_mutex_lock(&output);
        cout<<time_now<<"-Phi "<<phi_num<<"-taking right chopstick"<<"\n";
        pthread_mutex_unlock(&output);
        pthread_mutex_lock(&chopstick[left]);//take right chopstick.
        now = time(NULL);
        strftime(time_now,100,"%H:%M:%S",localtime(&now));
        pthread_mutex_lock(&output);
        cout<<time_now<<"-Phi "<<phi_num<<"-taking left chopstick"<<"\n";
        pthread_mutex_unlock(&output);
        //get right & left chopstick
        now = time(NULL);
        strftime(time_now,100,"%H:%M:%S",localtime(&now));
        pthread_mutex_lock(&output);
        cout<<time_now<<"-Phi "<<phi_num<<"-eating"<<"\n";
        pthread_mutex_unlock(&output);
        sleep(eat_time);//eating.... then put down the chopstick.
        now = time(NULL);
        strftime(time_now,100,"%H:%M:%S",localtime(&now));
        pthread_mutex_lock(&output);
        cout<<time_now<<"-Phi "<<phi_num<<"-putting right chopstick"<<"\n";
        pthread_mutex_unlock(&output);
        pthread_mutex_unlock(&chopstick[right]);
        now = time(NULL);
        strftime(time_now,100,"%H:%M:%S",localtime(&now));
        pthread_mutex_lock(&output);
        cout<<time_now<<"-Phi "<<phi_num<<"-putting left chopstick"<<"\n";
        pthread_mutex_unlock(&output);
        pthread_mutex_unlock(&chopstick[left]);
        now = time(NULL);
        strftime(time_now,100,"%H:%M:%S",localtime(&now));
        cout<<time_now<<"-Phi "<<phi_num<<"-thinking"<<"\n";
        sleep(think_time);
    }
}
break;
case 3:
```

C. **同時拿筷子的演算法：完成本演算法者，本演算法最多可得 20 分。**

第三個演算法則是先拿左手的筷子，若右手的筷子無法拿，則將左手的筷子放下，若可以拿則印出拿到兩支筷子的 message 並且進行吃飯

```
case 3:
for(int i=0;i<10;i++){
    think_time = rand()%(10-5+1)+5; eat_time = rand()%(5-1+1)+1;


    while(true){
        while(!pthread_mutex_trylock(&chopstick[left]));
        if(!pthread_mutex_trylock(&chopstick[right]))
        {
            break;
        }
        else
        {
            pthread_mutex_unlock(&chopstick[left]);
        }

    }
    now = time(NULL);
    strftime(time_now,100,"%H:%M:%S",localtime(&now));
    pthread_mutex_lock(&output);
    cout<<time_now<<"-Phi "<<phi_num<<"-taking two chopsticks"<<"\n";
    pthread_mutex_unlock(&output);

    //get right & left chopstick
    now = time(NULL);
    strftime(time_now,100,"%H:%M:%S",localtime(&now));
    pthread_mutex_lock(&output);
    cout<<time_now<<"-Phi "<<phi_num<<"-eating"<<"\n";
    pthread_mutex_unlock(&output);
    sleep(eat_time);//eating.... then put down the chopstick.
    now = time(NULL);
    strftime(time_now,100,"%H:%M:%S",localtime(&now));
    pthread_mutex_lock(&output);
    cout<<time_now<<"-Phi "<<phi_num<<"-putting left chopstick"<<"\n";
    pthread_mutex_unlock(&output);
    pthread_mutex_unlock(&chopstick[left]);
```