# Southern Methodist University
# Bobby B. Lyle School of Engineering Department of Computer Science
# Homework 5

Operating System and Software System

Name: Bingying Liang
ID: 48999397
Email: bingyingl@smu.edu

CS7343 Distance

Feb 28 2023

- CS 5343 students must answer exactly 2 projects
- CS 7343 students must answer all projects

1. **Project 2—The Sleeping Teaching Assistant (Chapter 7, P-38)**
2. **Project 3—The Dining-Philosophers Problem (Chapter 7, P-39)**
3. **Project 4—The Producer – Consumer Problem (Chapter 7, P-40)**

For each project, please submit

1. The source code of the project
2. Brief description of how to test your program
3. A sample Output

All of the above should be place on folder (called Project1 & Project2). These two folders should be placed in a folder bearing your name (e.g., Mike Smith). The folder "Mike Smith" should be zipped and uploaded to Canvas.

# Environment

Laptop: MacBook Air M2 2022, macOS 13.3

Programming APP: CLion 2022.2.3

Runtime version: 17.0.4+7-b469.53 aarch64

VM: OpenJDK 64-Bit Server VM by JetBrains s.r.o.

GC: G1 Young Generation, G1 Old Generation

Memory: 2000M

Cores: 8

Running: Ubuntu 22.04 ARM64 Parallels Desktop as virtual environment.

# Project 2–The sleeping Teaching Assistant

A university computer science department has a teaching assistant (TA) who helps undergraduate students with their programming assignments during regular office hours. The TA's office is rather small and has room for only one desk with a chair and computer. There are three chairs in the hallway outside the office where students can sit and wait if the TA is currently helping another student. When there are no students who need help during office hours, the TA sits at the desk and takes a nap. If a student arrives during office hours and finds the TA sleeping, the student must awaken the TA to ask for help. If a student arrives and finds the TA currently helping another student, the student sits on one of the chairs in the hallway and waits. If no chairs are available, the student will come back at a later time.
Using POSIX threads, mutex locks, and semaphores, implement a solution that cooridnates the activities of the TA and the students. Details for this assignment are provided below.

## The Students and the TA

Using Pthreads(Section 4.4.1), begin by creating $n$ students where each student will run as a separate thread. The TA will run as a separate thread as well. Students thread will alternate between programming for a period of time and seeking help from the TA. If the TA is available, they will obtain help. Otherwise, they will either sit in a chair in the hallway or, if no chairs are available, will resume programming and will seek help at a later time. If a student arrives and notices that the TA is sleeping, the student must notify the TA using a semaphore. When the TA finishes helping a student, the TA must check to see if there are students waiting for help in the hallway. If so, the TA must help each of these students in turn. If no students are present, the TA may return to napping. Perhaps the best option for simulating students programming – as well as the TA providing help to a student – is to have the appropriate threads sleep for a random period of time. Coverage of POSIX mutex locks and semaphores is provided Section 7.3. Consult that section for details.

**Solution:**

```c
// Created by Eve Liang on 3/26/23.
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <pthread.h>
#include <semaphore.h>
#include <sys/time.h>

int ta_teaching_time;
int s_visit_time;
int s_leave_time;

int chair_num;
int ta_num;
```

```
16  int student_num;
17
18  sem_t student_sem;
19  sem_t ta_sem;
20
21  sem_t mutex;
22  sem_t s_mutex;
23  sem_t t_mutex;
24
25  int i, j, k;
26
27  int working_ta = 1;
28  int waiting_students = 0;
29
30  int leave_cnt = 0;
31  int served_cnt = 0;
32  int ta_serve_cnt[50] = {0};
33  void msleep(int tms);
34
35  void set_useed(){
36      struct timeval tv;
37      gettimeofday(&tv, NULL);
38      srand(tv.tv_sec + tv.tv_usec);
39  }
40  void *TA(void *tid_){
41      while (1)
42      {
43          set_useed();
44          long tid = (long)tid_;
45
46          sem_wait(&student_sem);
47          sem_wait(&mutex);
48          waiting_students--;
49          printf("TA start teaching.\n");
50          printf("The number of waiting students: %d\n", waiting_students);
51          sem_post(&mutex);
52          set_useed();
53          ta_teaching_time = rand() % 1001;
54          msleep(ta_teaching_time);
55
56          int current_served_cnt;
57          sem_wait(&t_mutex);
58          served_cnt++;
59          current_served_cnt = served_cnt;
60          ta_serve_cnt[tid]++;
```

```
61          sem_post(&t_mutex);
62
63          printf("%dth arrived student finishes.", current_served_cnt);
64          printf("This time TA service time is: %d\n", ta_teaching_time);
65          sem_post(&ta_sem);
66          printf("TA if no students will sleep\n");
67
68      }
69  }
70
71  void *student(void *sid_){
72      s_leave_time = rand() % 21;
73      long sid = (long)sid_;
74      printf("%ldth students come here\n", sid);
75
76      sem_wait(&mutex);     // for waiting_students
77      sem_wait(&s_mutex);
78
79      if (waiting_students == chair_num){
80          leave_cnt++;
81
82          printf("There is no seats, %ldth student leave:%d\n", sid, leave_cnt);
83          /* free mutexes */
84          sem_post(&s_mutex);
85          sem_post(&mutex);
86      }
87      else{
88          waiting_students++;
89          printf("%ldth student has chair, sit down. The waiting student number is:%d\n", sid
90
91          int student_value;
92
93          /* free mutexes */
94          sem_post(&s_mutex);
95          sem_post(&mutex);
96
97          sem_post(&student_sem);
98
99          sem_wait(&ta_sem);
100     }
101
102     msleep(s_leave_time);
103 }
104
105 void msleep(int tms){
```

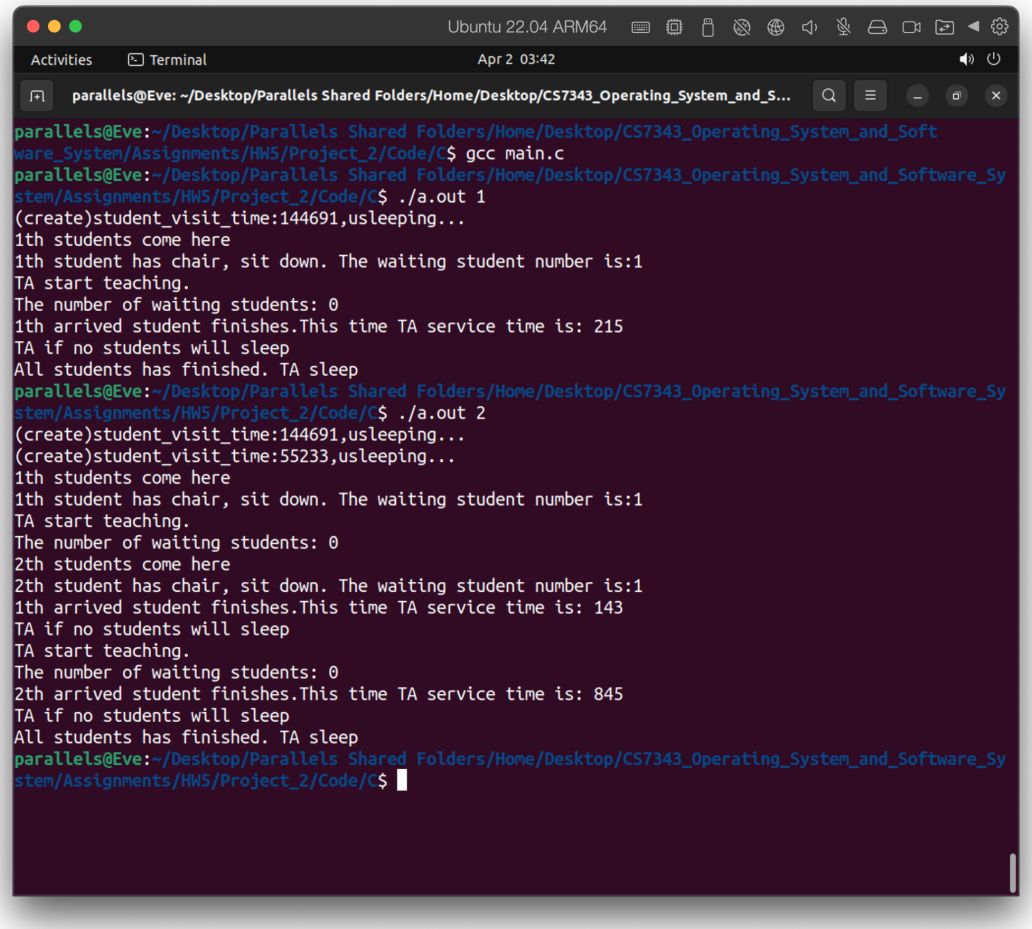
```
106     struct timeval tv;
107
108     tv.tv_sec = tms / 1000;
109     tv.tv_usec = (tms % 1000) * 1000;
110     select(0, NULL, NULL, NULL, &tv);
111 }
112
113 int main(int argc, char *argv[]){
114
115     chair_num = 4; // one in office, the other three hallway
116     ta_num = 1;
117
118     student_num = atoi(argv[1]);
119
120     if (argc != 2){
121         fprintf(stderr, "Please use the command format: <file> <number of students>");
122     }
123
124     pthread_t ta_thread[ta_num], student_thread[student_num];
125
126     sem_init(&ta_sem, 0, 0);
127     sem_init(&student_sem, 0, 0);
128
129     sem_init(&mutex, 0, 1);
130     sem_init(&s_mutex, 0, 1);
131     sem_init(&t_mutex, 0, 1);
132
133
134     for (i = 1; i <= ta_num; i++)
135     {
136         pthread_create(&ta_thread[i], NULL, TA, (void *) (long) (i));
137     }
138
139     for (i = 1; i <= student_num; i++)
140     {
141         s_visit_time = rand() % 500000 / 2;
142         printf("(create)student_visit_time:%d,usleeping...\n", s_visit_time);
143         usleep(s_visit_time);
144         pthread_create(&student_thread[i], NULL, student, (void *) (long) (i));
145
146     }
147
148     for (i = 1; i <= student_num; i++)
149     {
150         pthread_join(student_thread[i], NULL);
```

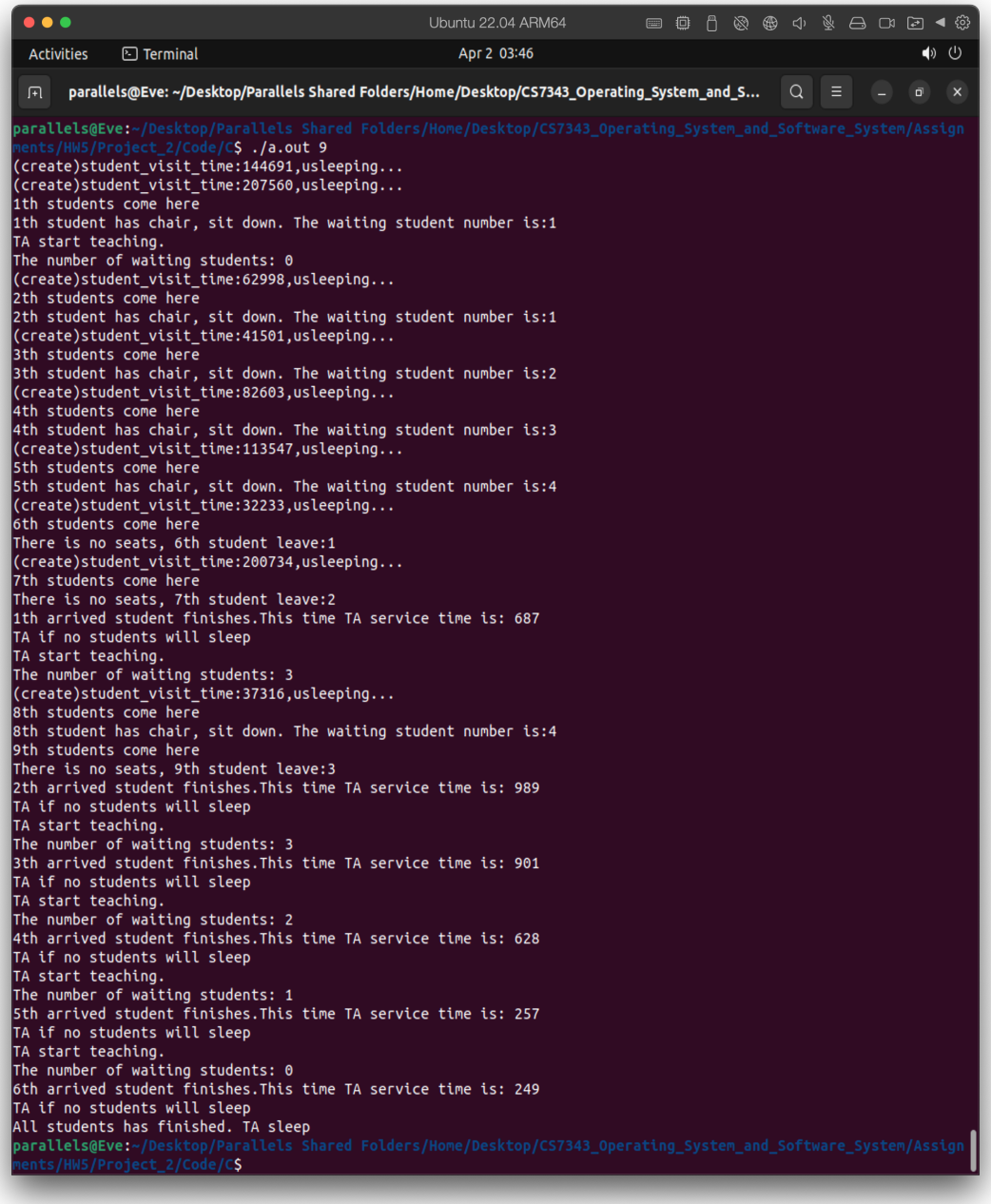
```
151         }
152
153         printf("All students has finished. TA sleep\n");
154
155         return 0;
156
157 }
158
```

### The result

```
./<filename> <The numbers of students>
```

The result for running of each philosopher only eating one times and two times.




Ubuntu 22.04 ARM64
Activities
Terminal
Apr 2 03:46
parallels@Eve: ~/Desktop/Parallels Shared Folders/Home/Desktop/CS7343_Operating_System_and_S...
parallels@Eve:~/Desktop/Parallels Shared Folders/Home/Desktop/CS7343_Operating_System_and_Software_System/Assignments/HW5/Project_2/Code/C$ ./a.out 9
(create)student_visit_time:144691,usleeping...
(create)student_visit_time:207560,usleeping...
1th students come here
1th student has chair, sit down. The waiting student number is:1
TA start teaching.
The number of waiting students: 0
(create)student_visit_time:62998,usleeping...
2th students come here
2th student has chair, sit down. The waiting student number is:1
(create)student_visit_time:41501,usleeping...
3th students come here
3th student has chair, sit down. The waiting student number is:2
(create)student_visit_time:82603,usleeping...
4th students come here
4th student has chair, sit down. The waiting student number is:3
(create)student_visit_time:113547,usleeping...
5th students come here
5th student has chair, sit down. The waiting student number is:4
(create)student_visit_time:32233,usleeping...
6th students come here
There is no seats, 6th student leave:1
(create)student_visit_time:200734,usleeping...
7th students come here
There is no seats, 7th student leave:2
1th arrived student finishes.This time TA service time is: 687
TA if no students will sleep
TA start teaching.
The number of waiting students: 3
(create)student_visit_time:37316,usleeping...
8th students come here
8th student has chair, sit down. The waiting student number is:4
9th students come here
There is no seats, 9th student leave:3
2th arrived student finishes.This time TA service time is: 989
TA if no students will sleep
TA start teaching.
The number of waiting students: 3
3th arrived student finishes.This time TA service time is: 901
TA if no students will sleep
TA start teaching.
The number of waiting students: 2
4th arrived student finishes.This time TA service time is: 628
TA if no students will sleep
TA start teaching.
The number of waiting students: 1
5th arrived student finishes.This time TA service time is: 257
TA if no students will sleep
TA start teaching.
The number of waiting students: 0
6th arrived student finishes.This time TA service time is: 249
TA if no students will sleep
All students has finished. TA sleep
parallels@Eve:~/Desktop/Parallels Shared Folders/Home/Desktop/CS7343_Operating_System_and_Software_System/Assignments/HW5/Project_2/Code/C$

# Project 3–The Dining-Philosophers Problem

In Section 7.1.3, we provide an outline of a solution to the dining-philosophers problem using monitors. This project involves implementing a solution to this problem using either POSIX mutex locks and condition variables of Java condition variables. Solutions will be based on the algorithm illustrated in Figure 7.7.
Both implementations will require creating five philosophers, each identified by a number $0 \ldots 4$. Each philosopher will run as a separate thread. Philosophers alternate between thinking and eating. To simulate both activities, have each thread sleep for a random period between one and three seconds.

**Solution:**
**main.c**:

```c
//
// Created by Eve Liang on 3/26/23.
//
#include <stdio.h>
#include <pthread.h>
#include <unistd.h>
#include <stdlib.h>

/*    p0   p1   p2
        p3   p4
 */


void pickup(int i);
void putdown(int i);
void test(int i);
void *runner(void *param); /* threads call this function */

pthread_mutex_t mutex;
pthread_cond_t cond_var[5];

int eating_number;

// monitor DiningPhilosophers
enum {THINKING, HUNGRY, EATING} state[5];

// pi hungry, pi want to pickup if yes state --> EAT
void pickup(int i){
    pthread_mutex_lock(&mutex);
    state[i] = HUNGRY;
    test(i);
```

```
32      while (state[i] != EATING) {
33          //self[i].wait();
34          //printf("Philosopher%d is HUNGRY and waiting to pick up.\n", i);
35          pthread_cond_wait(&cond_var[i], &mutex);
36      }
37      pthread_mutex_unlock(&mutex);
38  }
39
40  // check pi whether can eat
41  void test(int i){
42      if ((state[(i + 4) % 5] != EATING) &&
43      (state[i] == HUNGRY) && (state[(i + 1) % 5] != EATING)){
44          state[i] = EATING;
45          pthread_cond_signal(&cond_var[i]);                    // self[i].signal();
46      }
47  }
48
49  // putdown, pi change to think, and check neighbour pi+1, p-1 whether want to pickup
50  void putdown(int i){
51      pthread_mutex_lock(&mutex);
52      state[i] = THINKING;
53      test((i + 4) % 5);
54      test((i + 1) % 5);
55      pthread_mutex_unlock(&mutex);
56  }
57
58  int main(int argc, char *argv[]) {
59      if (argc != 2){
60          fprintf(stderr,"Please use the command format:
61          <the number of philosophers eating times>");
62          return -1;
63      }
64      eating_number = atoi(argv[1]);
65      int index[5];
66
67      pthread_t tid[5]; /* the thread identifier */
68      pthread_attr_t attr; /* set of the thread attributes */
69
70      /* set the default attributes of the thread */
71      pthread_attr_init(&attr);
72      // create and initialize a thread's attribute structure
73
74      pthread_mutex_init(&mutex, NULL);
75
76      for (int i = 0; i < 5; i++) {
```

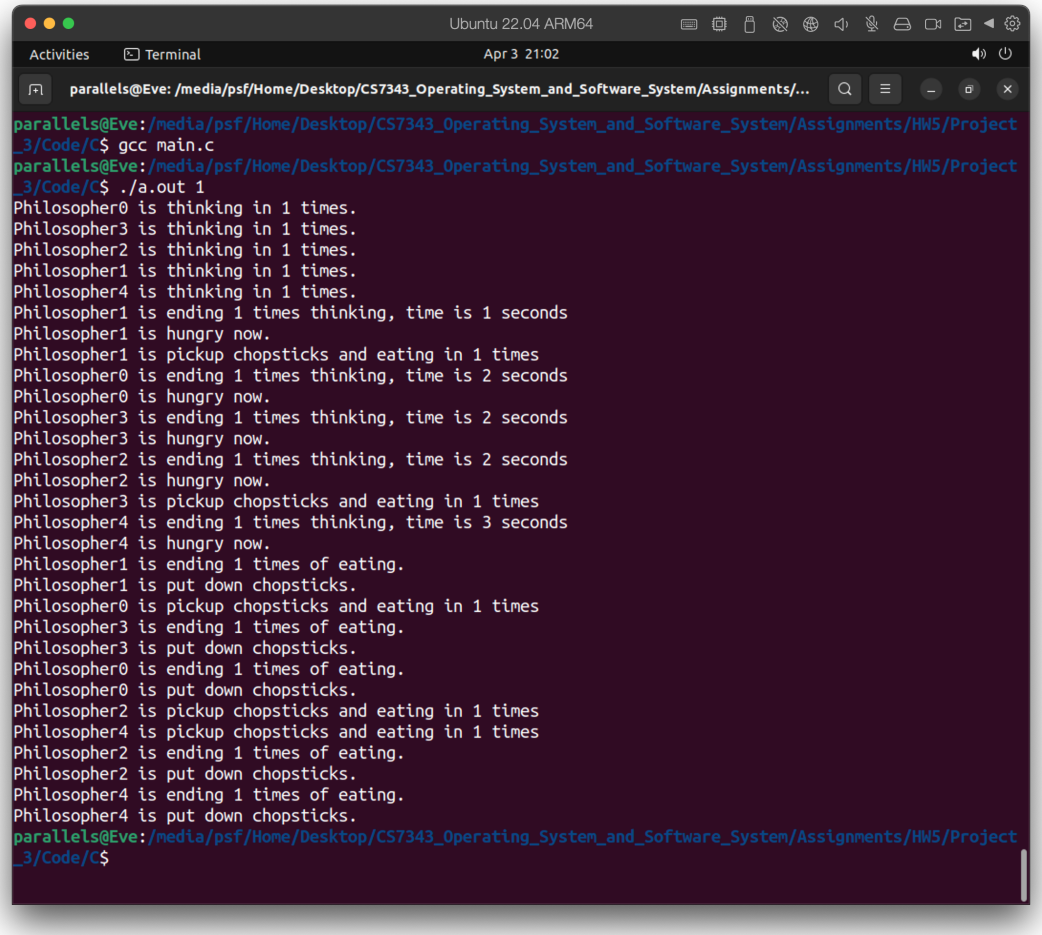
```
77          state[i] = THINKING;
78          pthread_cond_init(&cond_var[i], NULL);
79      }
80
81      //  create the thread
82      for (int i = 0; i < 5; i++) {
83          index[i] = i;
84          pthread_create(&tid[i], &attr, runner, &index[i]); // create a new thread
85      }
86
87      /* wait for the thread to exit */
88      for (int i = 0; i < 5; i++) {
89          pthread_join(tid[i], NULL); // Wait for a specific thread to exit4
90      }
91
92      return 0;
93  }
94
95  void *runner(void *param){
96      int i = *((int *) param);
97      int count = 0;
98      int time; // sleep for a random period between one and three seconds.
99      while (count < eating_number){
100         count++;
101         printf("Philosopher%d is thinking in %d times.\n", i, count);
102         time = rand() % 3 + 1;
103         sleep(time);
104         printf("Philosopher%d is ending %d times thinking, time is %d seconds\n",
105                i, count, time);
106         printf("Philosopher%d is hungry now.\n", i);
107         pickup(i);
108         printf("Philosopher%d is pickup chopsticks and eating in %d times\n",
109                i, count);
110         time = rand() % 3 + 1;
111         sleep(time);
112         printf("Philosopher%d is ending %d times of eating.\n",
113                 i, count);
114         putdown(i);
115         printf("Philosopher%d is put down chopsticks.\n", i);
116
117     }
118
119     pthread_exit(0);
120 }
```

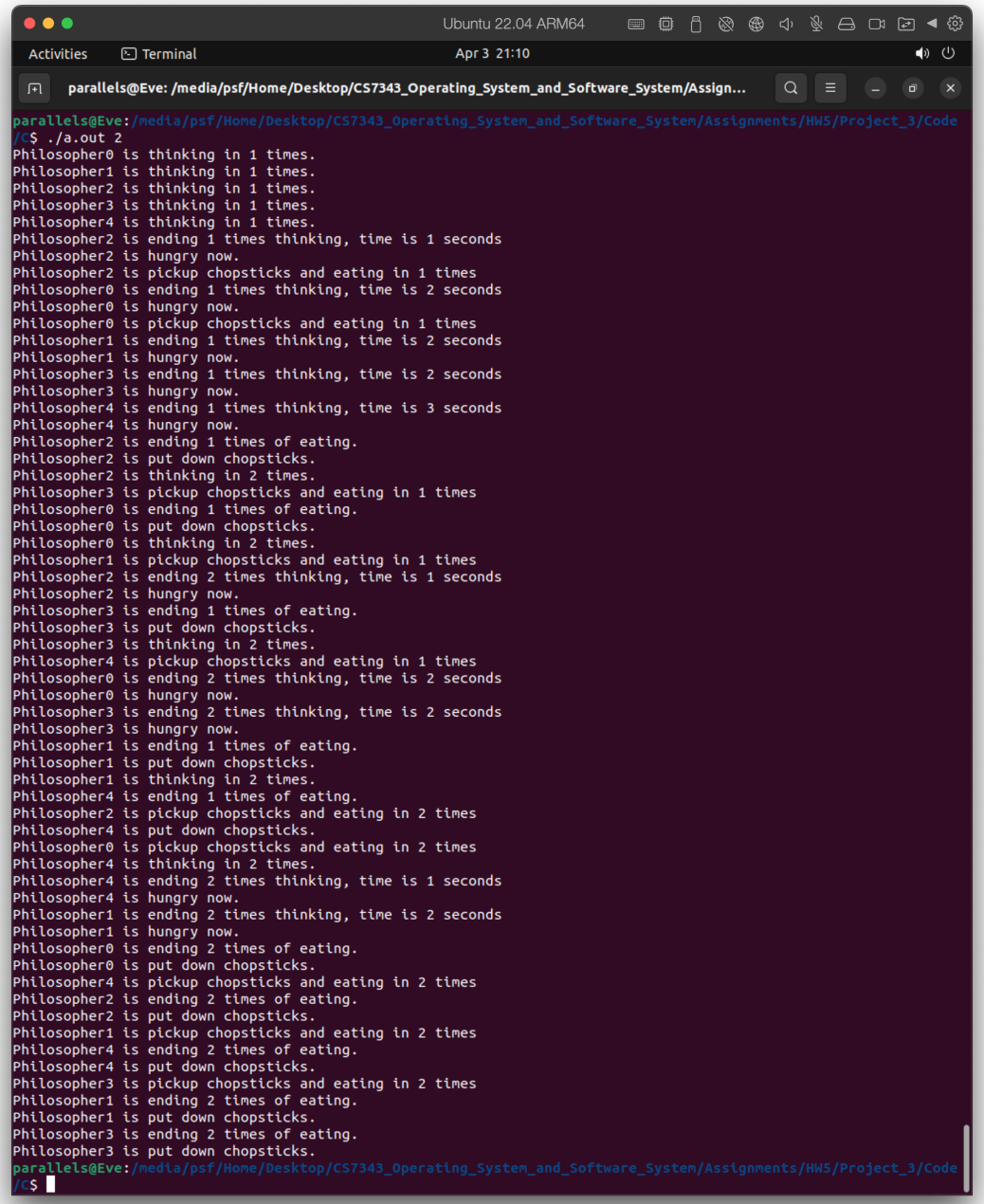
121

## The result

```
./<filename> <The times of philosopher eating>
```

The result for running of each philosopher only eating one times and two times.




Ubuntu 22.04 ARM64
Activities
Terminal
Apr 3 21:10
parallels@Eve: /media/psf/Home/Desktop/CS7343_Operating_System_and_Software_System/Assign...
parallels@Eve:/media/psf/Home/Desktop/CS7343_Operating_System_and_Software_System/Assignments/HW5/Project_3/Code/c$ ./a.out 2
Philosopher0 is thinking in 1 times.
Philosopher1 is thinking in 1 times.
Philosopher2 is thinking in 1 times.
Philosopher3 is thinking in 1 times.
Philosopher4 is thinking in 1 times.
Philosopher2 is ending 1 times thinking, time is 1 seconds
Philosopher2 is hungry now.
Philosopher2 is pickup chopsticks and eating in 1 times
Philosopher0 is ending 1 times thinking, time is 2 seconds
Philosopher0 is hungry now.
Philosopher0 is pickup chopsticks and eating in 1 times
Philosopher1 is ending 1 times thinking, time is 2 seconds
Philosopher1 is hungry now.
Philosopher3 is ending 1 times thinking, time is 2 seconds
Philosopher3 is hungry now.
Philosopher4 is ending 1 times thinking, time is 3 seconds
Philosopher4 is hungry now.
Philosopher2 is ending 1 times of eating.
Philosopher2 is put down chopsticks.
Philosopher2 is thinking in 2 times.
Philosopher3 is pickup chopsticks and eating in 1 times
Philosopher0 is ending 1 times of eating.
Philosopher0 is put down chopsticks.
Philosopher0 is thinking in 2 times.
Philosopher1 is pickup chopsticks and eating in 1 times
Philosopher2 is ending 2 times thinking, time is 1 seconds
Philosopher2 is hungry now.
Philosopher3 is ending 1 times of eating.
Philosopher3 is put down chopsticks.
Philosopher3 is thinking in 2 times.
Philosopher4 is pickup chopsticks and eating in 1 times
Philosopher0 is ending 2 times thinking, time is 2 seconds
Philosopher0 is hungry now.
Philosopher3 is ending 2 times thinking, time is 2 seconds
Philosopher3 is hungry now.
Philosopher1 is ending 1 times of eating.
Philosopher1 is put down chopsticks.
Philosopher1 is thinking in 2 times.
Philosopher4 is ending 1 times of eating.
Philosopher2 is pickup chopsticks and eating in 2 times
Philosopher4 is put down chopsticks.
Philosopher0 is pickup chopsticks and eating in 2 times
Philosopher4 is thinking in 2 times.
Philosopher4 is ending 2 times thinking, time is 1 seconds
Philosopher4 is hungry now.
Philosopher1 is ending 2 times thinking, time is 2 seconds
Philosopher1 is hungry now.
Philosopher0 is ending 2 times of eating.
Philosopher0 is put down chopsticks.
Philosopher4 is pickup chopsticks and eating in 2 times
Philosopher2 is ending 2 times of eating.
Philosopher2 is put down chopsticks.
Philosopher1 is pickup chopsticks and eating in 2 times
Philosopher4 is ending 2 times of eating.
Philosopher4 is put down chopsticks.
Philosopher3 is pickup chopsticks and eating in 2 times
Philosopher1 is ending 2 times of eating.
Philosopher1 is put down chopsticks.
Philosopher3 is ending 2 times of eating.
Philosopher3 is put down chopsticks.
parallels@Eve:/media/psf/Home/Desktop/CS7343_Operating_System_and_Software_System/Assignments/HW5/Project_3/Code/c$

# Project 4–The Producer-Consumer Problem

In section 7.1.1, we presented a semaphore-based solution to the producer-consumer problem using a bounded buffer. In this project, you will design a programming solution to the bounded-buffer problem using the producer and consumer processes shown in Figure 5.9 and 5.10. The solution presented in Section 7.1.1 uses three semaphores: empty and full, which count the number of empty and full slots in the buffer, and mutex, which is a binary (or mutual-exclusion) semaphore that protects the actual insertion or removal of items in the buffer. For this project, you will use standard counting semaphores for empty and full and a mutex lock rather than a binary semaphore, to represent mutex. The producer and consumer–running as separate threads–will move items to and from a buffer that is synchronized with the empty, full, and mutex structures. You can solve this problem using either Pthreads or the Windows API.

**Solution:**

(a) **The buffer:** buffer.h

```c
// Created by Eve Liang on 3/26/23.
//
/* buffer.h */
/* buffer fix array, statement of item; */
typedef int buffer_item;

/* The definition of the array size */
#define BUFFER_SIZE 5

/* The operation in the buffer, separately used for
producer and consumer thread */
int insert_item(buffer_item item);
int remove_item(buffer_item *item);
```

(b) **The buffer operations:** buffer.c

```c
// Created by Eve Liang on 3/26/23.
//
#include "buffer.h"

#include <pthread.h>
#include <semaphore.h>

/* the buffer */
buffer_item buffer[BUFFER_SIZE];

pthread_mutex_t mutex;   /*pthread mutex lock*/
```

```
12
13  /* semaphore data type */
14  sem_t empty; // buffer empty
15  sem_t full;  // buffer full
16
17  int insertIndex = 0;
18  int removeIndex = 0;
19
20  /* threads function */
21  void *producer(void *param);
22  void *consumer(void *param);
23
24
25  int insert_item(buffer_item item){
26      /* insert item into buffer
27      return 0 if successful, otherwise
28      return -1 indicating an error condition */
29
30      /* produce an item in next_produced */
31      sem_wait(&empty); // wait(empty)
32      pthread_mutex_lock(&mutex); // wait(mutex)
33
34      /* add next_produced to the buffer */
35      buffer[insertIndex] = item;
36      insertIndex++;
37      insertIndex = insertIndex % BUFFER_SIZE;
38
39      pthread_mutex_unlock(&mutex); // signal(mutex)
40      sem_post(&full); // signal(full)
41
42      return 0;
43  }
44
45  int remove_item(buffer_item *item){
46      /* remove an object from buffer
47      placing it in item
48      return 0 if successful, otherwise
49      return -1 if indicating an error condition */
50      sem_wait(&full);   // wait(full)
51      pthread_mutex_lock(&mutex);  // wait(mutex)
52
53      /* remove an item from buffer to next_consumed */
54      *item = buffer[removeIndex];
55      buffer[removeIndex] = -1;
56      removeIndex++;
```

```
    removeIndex = removeIndex % BUFFER_SIZE;

    pthread_mutex_unlock(&mutex);  // signal(mutex)
    sem_post(&empty);        // signal(empty)

    /* consume the item in next_consumed */
    return 0;
}
```

(c) **Main:** main.c

```
// Created by Eve Liang on 3/26/23.
//
#include "buffer.h"
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <semaphore.h>
#include <unistd.h>

#include "pcthreads.c"
#include "buffer.c"

#define TRUE 1

int main(int argc, char*argv[]) {
    /*  The `main()` function will be passed three parameters on the command line:
        1. How long to sleep before terminatin
        2. The number of producer threads
        3. The number of consumer threads
     */

    /* 1. Get command line arguments argv[1], argv[2], argv[3] */
    int sleepTime;
    int n_producer;  // n_producer thread
    int n_consumer;  // n_consumer thread
    int i, j;
    if (argc != 4) {
        fprintf(stderr,
        "Please use the command format: <How long to sleep before terminatin>
        <producer number> <consumer number>");
        return -1;
    }
```

```
33
34      sleepTime = atoi(argv[1]);
35      n_producer = atoi(argv[2]);
36      n_consumer = atoi(argv[3]);
37
38      /* 2. Initialize buffer */
39      pthread_mutex_init(&mutex, NULL);
40      sem_init(&empty, 0, BUFFER_SIZE);
41      sem_init(&empty, 0, BUFFER_SIZE);
42      srand(time(0));
43
44      /* 3. Create producer thread(s) */
45      for (i = 0; i < n_producer; i++) {
46          pthread_t tid;
47          pthread_attr_t attr;
48          pthread_attr_init(&attr);
49          pthread_create(&tid, &attr, producer, NULL);
50      }
51
52      /* 4. Create n_consumer thread(s) */
53      for (j = 0; j < n_consumer; j++) {
54          pthread_t tid;
55          pthread_attr_t attr;
56          pthread_attr_init(&attr);
57          pthread_create(&tid, &attr, consumer, NULL);
58      }
59
60      /* 5. Sleep */
61      sleep(sleepTime);
62
63      /* 6 Exit */
64      return 0;
65  }
66
```

(d) **The Producer and Consumer Threads:** pcthreads.c

```
1   // Created by Eve Liang on 3/26/23.
2   //
3   #include "buffer.h"
4   #include <stdio.h>
5   #include <stdlib.h>
6   #include <pthread.h>
7   #include <semaphore.h>
```

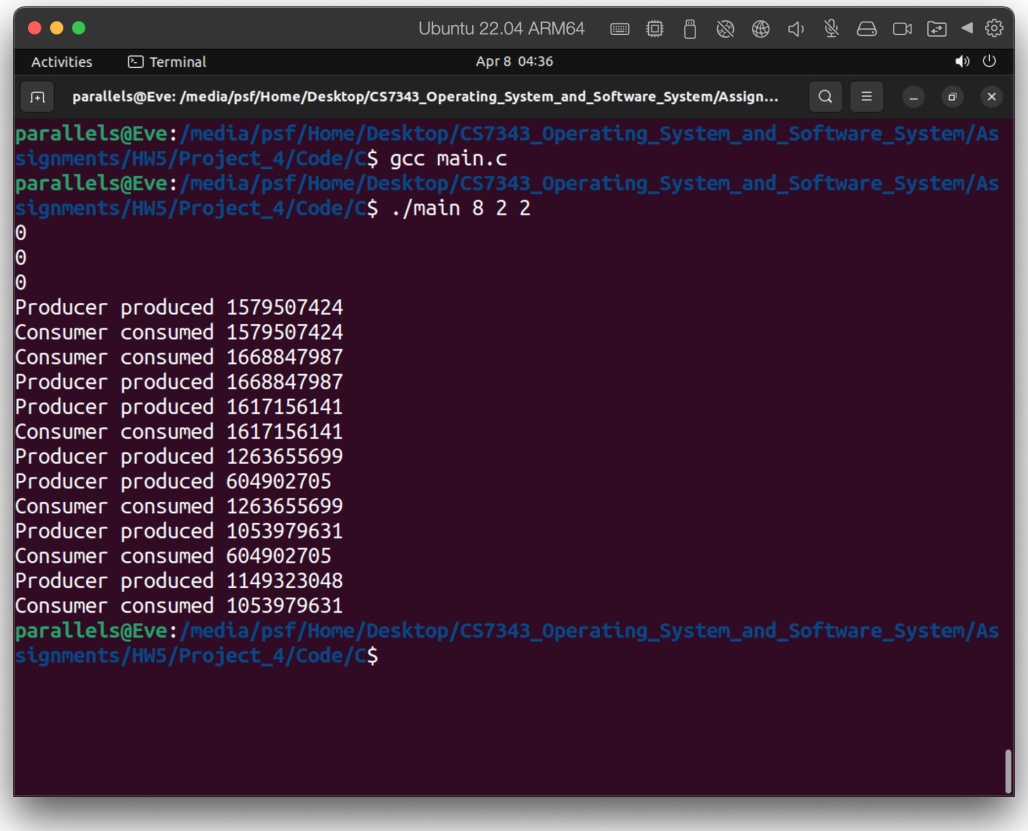
```c
#include <unistd.h>

#include "pcthreads.c"
#include "buffer.c"

#define TRUE 1

int main(int argc, char*argv[]) {
    /*  The `main()` function will be passed three parameters on the command line:
        1. How long to sleep before terminatin
        2. The number of producer threads
        3. The number of consumer threads
     */

    /* 1. Get command line arguments argv[1], argv[2], argv[3] */
    int sleepTime;
    int n_producer;  // n_producer thread
    int n_consumer;  // n_consumer thread
    int i, j;
    if (argc != 4) {
        fprintf(stderr,
                "Please use the command format: <How long to sleep before terminatin>
        return -1;
    }

    sleepTime = atoi(argv[1]);
    n_producer = atoi(argv[2]);
    n_consumer = atoi(argv[3]);

    /* 2. Initialize buffer */
    pthread_mutex_init(&mutex, NULL);
    sem_init(&empty, 0, BUFFER_SIZE);
    sem_init(&empty, 0, BUFFER_SIZE);
    srand(time(0));

    /* 3. Create producer thread(s) */
    for (i = 0; i < n_producer; i++) {
        pthread_t tid;
        pthread_attr_t attr;
        pthread_attr_init(&attr);
        pthread_create(&tid, &attr, producer, NULL);
    }

    /* 4. Create n_consumer thread(s) */
    for (j = 0; j < n_consumer; j++) {
```

```
53          pthread_t tid;
54          pthread_attr_t attr;
55          pthread_attr_init(&attr);
56          pthread_create(&tid, &attr, consumer, NULL);
57      }
58
59      /* 5. Sleep */
60      sleep(sleepTime);
61
62      /* 6 Exit */
63      return 0;
64  }
```

### The result

```
./<filename> <execute time> <producers number> <consumer number>
```

The result for running the for 8 seconds with 2 producer and 2 consumer.




More code detail can see in the source code of the projects.