



西北工业大学
NORTHWESTERN POLYTECHNICAL UNIVERSITY

Lab **5** Report

Report Subject: OS Experiment - Lab **5**

Student ID: 2018380039

Student Name: Dikshya Kafle

Submission Date: **2021/11/24**

Computer Operating System Experiment

Laboratory 5 Synchronization

1) Objective:

- ❖ Learn to work with Linux and Pthread synchronization mechanisms.
- ❖ Practice the critical-section problem
- ❖ Examine several classical synchronization problems

2) Equipment:

- ❖ VirtualBox with Ubuntu Linux

3) Experiments:

Experiment 1: Thread Synchronization Problems

1. Do you think that the counter increase correctly? If not, what is wrong? And please increase the counter correctly using multithreads

The counter is a shared variable that will be edited by the threads. However it will not increase correctly cause it is not protected. This causes a race condition where the values of “counter” are unpredictable and vary depending on the timings of context switches of the threads in this case. To illustrate this, we run the program multiple time and we notice the different behaviours:

```
dikshya@dikshya-VirtualBox: ~/Desktop/Lab 5 OS
dikshya@dikshya-VirtualBox:~/Desktop/Lab 5 OS$ gcc ex1.c -o ex1 -lpthread -lrt
dikshya@dikshya-VirtualBox:~/Desktop/Lab 5 OS$ ./ex1

Job 1 started
Job 1 finished
Job 2 started
Job 2 finished
dikshya@dikshya-VirtualBox:~/Desktop/Lab 5 OS$
```

Notice how both threads will access the shared variable and result in race condition. To correct this, we need to protect the shared variable “counter” and make only one threads able to edit it at a time. For this we use the `pthread_mutex_t` data type (or a binary semaphore):

Open ▾



ex1.c

~/Desktop/Lab 5 OS

```
1 #include<stdio.h>
2
3 #include<string.h>
4
5 #include<pthread.h>
6
7 #include<stdlib.h>
8
9 #include<unistd.h>
10
11
12
13 pthread_t tid[2];
14
15 int counter;
16
17 pthread_mutex_t mutex;
18
19
20
21 void* doSomething(void *arg)
22
23 {
24     pthread_mutex_lock(&mutex);
25
26     long i = 0;
27
28     counter +=1;
29
30
31     printf("\n Job %d started\n", counter);
32
```

Open ▾



ex1.c

~/Desktop/Lab 5 OS

```
32
33     for(i=0; i<1000000;i++);
34
35         printf("\n Job %d finished\n", counter);
36
37         pthread_mutex_unlock(&mutex);
38
39     return NULL;
40
41 }
42
43
44
45 int main(void)
46
47 {
48
49     int i = 0;
50
51     int err;
52
53
54
55     if (pthread_mutex_init(&mutex, NULL) !=0)
56     {
57
58
59         printf("\n can't create thread :[%s]", strerror(err));
60
61         return 1;
62
```

```
Open ▾  ex1.c  
~/Desktop/Lab 5 OS  
60  
61         return 1;  
62  
63     }  
64  
65  
66  
67     while (i < 2)  
68     {  
69  
70         err = pthread_create(&(tid[1]), NULL, &doSomething, NULL);  
71  
72         if (err != 0)  
73             printf("\n can't create thread :[%s]", strerror(err));  
74  
75         i++;  
76     }  
77  
78  
79  
80  
81  
82     pthread_join(tid[0], NULL);  
83  
84     pthread_join(tid[1], NULL);  
85  
86     pthread_mutex_destroy(&mutex);  
87  
88     return 0;  
89  
90
```

Results:

```
dikshya@dikshya-VirtualBox: ~/Desktop/Lab 5 OS
dikshya@dikshya-VirtualBox:~/Desktop/Lab 5 OS$ gcc ex1.c -o ex1 -lpthread -lrt
dikshya@dikshya-VirtualBox:~/Desktop/Lab 5 OS$ ./ex1

Job 1 started
Job 1 finished
Job 2 started
Job 2 finished
dikshya@dikshya-VirtualBox:~/Desktop/Lab 5 OS$ ./ex1

Job 1 started
Job 1 finished
Job 2 started
Job 2 finished
dikshya@dikshya-VirtualBox:~/Desktop/Lab 5 OS$
```

2. if Thread 1 must run before Thread 2, how do you do?

The semaphore is initially to zero and we make the second thread wait first until the first thread signals the lock.

Open ▾



exp1.c

~/Desktop/Lab 5 OS

```
1 #include<stdio.h>
2 #include<string.h>
3 #include<pthread.h>
4 #include<stdlib.h>
5 #include<unistd.h>
6 #include<semaphore.h>
7 #include<sys/types.h>
8
9
10 pthread_t tid[2];
11 int counter;
12 sem_t mutex;
13 void* doSomething(void *arg)
14 {
15     if(pthread_self()== tid[1])
16         sem_wait(&mutex);
17
18     unsigned long i = 0;
19     counter += 1;
20     printf("\n Job %d started by %d\n", counter,gettid());
21     for(i=0; i<1000;i++);
22     printf("\n Job %d finished by %d\n", counter,gettid());
23
24     if(pthread_self()== tid[0])
25         sem_post(&mutex);
26     return NULL;
27 }
28
29 int main(void)
30 {
31     int i = 0;
```



```
exp1.c
~/Desktop/Lab 5 OS

18 unsigned long i = 0;
19 counter += 1;
20 printf("\n Job %d started by %d\n", counter, gettid());
21 for(i=0; i<1000;i++);
22 printf("\n Job %d finished by %d\n", counter, gettid());
23
24 if(pthread_self()== tid[0])
25     sem_post(&mutex);
26 return NULL;
27 }
28
29 int main(void)
30 {
31     int i = 0;
32     int err;
33
34     sem_init(&mutex,0,0);
35
36     while(i < 2)
37     {
38         err = pthread_create(&(tid[i]), NULL, &doSomething, NULL);
39         if (err != 0)
40             printf("\ncan't create thread :[%s]", strerror(err));
41         i++;
42     }
43
44     pthread_join(tid[0], NULL);
45     pthread_join(tid[1], NULL);
46     sem_destroy(&mutex);
47     return 0;
}
```

Result:

```
dikshya@dikshya-VirtualBox:~/Desktop/Lab 5 OS$ ./exp1

Job 1 started by 40029

Job 1 finished by 40029

Job 2 started by 40030

Job 2 finished by 40030
dikshya@dikshya-VirtualBox:~/Desktop/Lab 5 OS$
```

Experiment 2: The Bounded-Buffer Problem

As specified, The buffer is manipulated with two functions, insert item() and remove item() ,

which are called by the producer and consumer threads, respectively.

The main() function initializes the buffer and create the separate producer and consumer threads then sleeps for the determined period. Upon awakening, the program is terminated.

The producer thread alternates between sleeping for a random period of time and inserting a random integer into the buffer. Random numbers are produced using the rand() function, which produces random integers between 0 and RAND MAX . The consumer also sleeps for a random period of time and, upon awakening, it attempts to remove an item from the buffer.

❖ Command: Prod_com [num_prod] [num_cons] [sleep_time]

The needed application programming interface for the bounded buffer ADT (buffer.h & buffer.c):


Open ▾



buffer.c

~/Desktop/Lab 5 OS

```
1 #include "buffer.h"
2 Buffer_item buffer[BUFFER_SIZE];
3 int counter;
4 int insert_item(Buffer_item item) {
5     if(counter < BUFFER_SIZE) {
6         buffer[counter] = item;
7         counter++;
8         return 0;
9     }
10    else {
11        //buffer is full
12        return -1;
13    }
14 }
15 int remove_item(Buffer_item *item) {
16     if(counter > 0) {
17         *item = buffer[(counter-1)];
18         counter--;
19         return 0;
20     }
21     else {
22         //buffer empty
23         return -1;
24     }
25 }
```

Open ▾ 

buffer.h
~/Desktop/Lab 5 OS

```
1 #ifndef _BUFFER_H_
2 #define _BUFFER_H_
3
4 #define BUFFER_SIZE 5
5 typedef int Buffer_item;
6
7 /* the buffer */
8 extern Buffer_item buffer[BUFFER_SIZE];
9
10 /* buffer counter */
11 extern int counter;
12
13
14 int insert_item(Buffer_item item);
15 int remove_item(Buffer_item *item);
16
17 #endif
18
```

The program (prod_cons.c):

Open ▾ 

prod_cons.c
~/Desktop/Lab 5 OS

```
1 #include <stdlib.h>
2 #include <stdio.h>
3 #include <unistd.h>
4 #include <pthread.h>
5 #include <semaphore.h>
6 #include "buffer.h"
7
8 #define DIVISOR 100000000
9
10 pthread_mutex_t mutex;
11 sem_t full, empty;
12
13 pthread_t tid;
14 pthread_attr_t attr;
15
16 void *producer(void *param);
17 void *consumer(void *param);
18
19
20 int main(int argc, char *argv[]) {
21
22     if(argc != 4) {
23         fprintf(stderr, "USAGE: .Prod_com [num_prod] [num_cons] [sleep_time]\n");
24     }
25
```

Open ▾



prod_cons.c
~/Desktop/Lab 5 OS

```
25
26  int i;
27  int NProd = atoi(argv[1]);
28  int NCons = atoi(argv[2]);
29  int mainSleepTime = atoi(argv[3]);
30
31
32  pthread_mutex_init(&mutex, NULL);
33  sem_init(&full, 0, 0);
34  sem_init(&empty, 0, BUFFER_SIZE);
35  pthread_attr_init(&attr);
36  counter = 0;
37
38  //Create the producers
39  for(i = 0; i < NProd; i++) {
40      pthread_create(&tid, &attr, producer, NULL);
41  }
42
43
44  //Create the consumers
45  for(i = 0; i < NCons; i++) {
46      pthread_create(&tid, &attr, consumer, NULL);
47  }
48
49  //main function sleep
```

Open ▾

prod_cons.c
~/Desktop/Lab 5 OS

```
50     sleep(mainSleepTime);
51
52     return 0;
53 }
54
55
56
57 //Producer Thread
58 void *producer(void *param) {
59
60     Buffer_item item;
61
62     while(1) {
63         //sleep
64         int rNum = rand() / DIVISOR;
65         sleep(rNum);
66
67         item = rand();
68
69         sem_wait(&empty);
70         pthread_mutex_lock(&mutex);
71
72         if(insert_item(item)) {
73             fprintf(stderr, "Producer reported an error while inserting\n");
74         }
```

Firefox Web Browser

prod_cons.c
~/Desktop/Lab 5 OS

```
73         fprintf(stderr, "Producer reported an error while inserting\n");
74     }
75     else {
76         printf("producer produced %d\n", item);
77     }
78
79     pthread_mutex_unlock(&mutex);
80     sem_post(&full);
81 }
82 }
83
84 //Consumer Thread
85 void *consumer(void *param) {
86     Buffer_item item;
87
88     while(1) {
89         /* sleep for a random period of time */
90         int rNum = rand() / DIVISOR;
91         sleep(rNum);
92
93         sem_wait(&full);
94         pthread_mutex_lock(&mutex);
95
96         if(remove_item(&item)) {
97             fprintf(stderr, "Consumer reported an error while removing\n");
```

```
95
96     if(remove_item(&item)) {
97         fprintf(stderr, "Consumer reported an error while removing\n");
98     }
99     else {
100         printf("consumer consumed %d\n", item);
101     }
102     pthread_mutex_unlock(&mutex);
103     sem_post(&empty);
104 }
105 }
106
```

The Makefile:

```
philosophers.c
1 task1:
2     gcc -c buffer.c
3     gcc -c prod_cons.c
4     gcc prod_cons.o buffer.o -o prod_cons -lpthread -lrt
```

The result:

```
dikshya@dikshya-VirtualBox: ~/Desktop/Lab 5 OS
gcc prod_cons.o buffer.o -o prod_cons -lpthread -lrt
dikshya@dikshya-VirtualBox:~/Desktop/Lab 5 OS$ ./prod_cons 3 3 50
producer produced 719885386
consumer consumed 719885386
producer produced 1189641421
consumer consumed 1189641421
producer produced 783368690
consumer consumed 783368690
producer produced 1967513926
consumer consumed 1967513926
producer produced 304089172
producer produced 35005211
consumer consumed 35005211
consumer consumed 304089172
producer produced 336465782
producer produced 278722862
consumer consumed 278722862
consumer consumed 336465782
producer produced 468703135
producer produced 1315634022
producer produced 1369133069
producer produced 1059961393
consumer consumed 1059961393
dikshya@dikshya-VirtualBox:~/Desktop/Lab 5 OS$
```

3.3 Experiment 3: Dining Philosophers (optional if you are still available)

Questions:

1. Do you think that this solution is good? If not, what is wrong?


```
dikshya@dikshya-VirtualBox: ~/Desktop/Lab 5 OS
dikshya@dikshya-VirtualBox:~/Desktop/Lab 5 OS$ ./philosophers
Philosopher 2 is done thinking and now ready to eat.
Philosopher 1 is done thinking and now ready to eat.
Philosopher 2: got right chopstick 2
Philosopher 2: got left chopstick 3
Philosopher 2: eating.
Philosopher 4 is done thinking and now ready to eat.
Philosopher 4: got right chopstick 4
Philosopher 4: got left chopstick 0
Philosopher 4: eating.
Philosopher 1: got right chopstick 1
Philosopher 0 is done thinking and now ready to eat.
Philosopher 3 is done thinking and now ready to eat.
Philosopher 1: got left chopstick 2
Philosopher 1: eating.
Philosopher 3: got right chopstick 3
Philosopher 0: got right chopstick 0
Philosopher 4: got right chopstick 4
Philosopher 2: got right chopstick 2
Philosopher 0: got left chopstick 1
Philosopher 0: eating.
Philosopher 0: got right chopstick 0
Philosopher 0: got left chopstick 1
Philosopher 0: eating.
Philosopher 0: got right chopstick 0
Philosopher 0: got left chopstick 1
Philosopher 0: eating.
Philosopher 0: got right chopstick 0
Philosopher 0: got left chopstick 1
Philosopher 0: eating.
Philosopher 0: got right chopstick 0
Philosopher 0: got left chopstick 1
Philosopher 0: eating.
```

2. Can you modify this solution to meet the requirements? And use semaphores to implement it.

Open ▾

philosopher.c
~/Desktop/Lab 5 OS

```
1  #include <pthread.h>
2  #include <stdio.h>
3  #include <unistd.h>
4  #include <stdlib.h>
5  #include <errno.h>
6  #include <assert.h>
7
8  #define PHILOS 5
9  #define DELAY 5000
10 #define FOOD 50
11
12 void *philosopher (void *id);
13 void grab_chopstick (int,
14                     int,
15                     char *);
16 void down_chopsticks (int,
17                      int);
18 int food_on_table ();
19
20 pthread_mutex_t chopstick[PHILOS];
21 pthread_t philo[PHILOS];
22 pthread_mutex_t food_lock;
23 int sleep_seconds = 0;
24
25 int
26 main (int argn,
27       char **argv)
28 {
29     int i;
30
31     if (argn == 2)
```

```
31         if (argn == 2)
32             sleep_seconds = atoi (argv[1]);
33
34         pthread_mutex_init (&food_lock, NULL);
35         for (i = 0; i < PHILOS; i++)
36             pthread_mutex_init (&chopstick[i], NULL);
37         for (i = 0; i < PHILOS; i++)
38             pthread_create (&philo[i], NULL, philosopher, (void *)i);
39         for (i = 0; i < PHILOS; i++)
40             pthread_join (philo[i], NULL);
41         return 0;
42     }
43
44     void *
45     philosopher (void *num)
46     {
47         int id;
48         int i, left_chopstick, right_chopstick, f;
49
50         id = (int)num;
51         printf ("Philosopher %d is done thinking and now ready to eat.\n", id);
52         right_chopstick = id;
53         left_chopstick = id + 1;
54
55         /* Wrap around the chopsticks. */
56         if (left_chopstick == PHILOS)
57             left_chopstick = 0;
58
59         while (f = food_on_table ()) {
60
61             /* Thanks to philosophers #1 who would like to take a nap
```

```
61      /* Thanks to philosophers #1 who would like to take a nap
62      * before picking up the chopsticks, the other philosophers
63      * may be able to eat their dishes and not deadlock.
64      */
65      if (id == 1)
66          sleep (sleep_seconds);
67
68      grab_chopstick (id, right_chopstick, "right ");
69      grab_chopstick (id, left_chopstick, "left");
70
71      printf ("Philosopher %d: eating.\n", id);
72      usleep (DELAY * (FOOD - f + 1));
73      down_chopsticks (left_chopstick, right_chopstick);
74  }
75
76      printf ("Philosopher %d is done eating.\n", id);
77      return (NULL);
78  }
79
80  int
81  food_on_table ()
82  {
83      static int food = FOOD;
84      int myfood;
85
86      pthread_mutex_lock (&food_lock);
87      if (food > 0) {
88          food--;
89      }
90      myfood = food;
91      pthread_mutex_unlock (&food_lock);
92
93      myfood = food;
94      pthread_mutex_unlock (&food_lock);
95      return myfood;
96  }
97
98  void
99  grab_chopstick (int phil,
100                 int c,
101                 char *hand)
102  {
103      pthread_mutex_lock (&chopstick[c]);
104      printf ("Philosopher %d: got %s chopstick %d\n", phil, hand, c);
105  }
106
107  void
108  down_chopsticks (int c1,
109                  int c2)
110  {
111      pthread_mutex_unlock (&chopstick[c1]);
112      pthread_mutex_unlock (&chopstick[c2]);
113  }
```

Results:

```
dikshya@dikshya-VirtualBox:~/Desktop/Lab 5 OS$ ./philosopher
Philosopher 2 is done thinking and now ready to eat.
Philosopher 2: got right chopstick 2
Philosopher 2: got left chopstick 3
Philosopher 2: eating.
Philosopher 1 is done thinking and now ready to eat.
Philosopher 0 is done thinking and now ready to eat.
Philosopher 0: got right chopstick 0
Philosopher 0: got left chopstick 1
Philosopher 0: eating.
Philosopher 3 is done thinking and now ready to eat.
Philosopher 4 is done thinking and now ready to eat.
Philosopher 4: got right chopstick 4
Philosopher 2: got right chopstick 2
Philosopher 2: got left chopstick 3
Philosopher 2: eating.
Philosopher 1: got right chopstick 1
Philosopher 0: got right chopstick 0
Philosopher 2: got right chopstick 2
Philosopher 2: got left chopstick 3
Philosopher 2: eating.
Philosopher 2: got right chopstick 2
Philosopher 2: got left chopstick 3
Philosopher 2: eating.
Philosopher 2: got right chopstick 2
Philosopher 2: got left chopstick 3
Philosopher 2: eating.
Philosopher 2: got right chopstick 2
Philosopher 3: got right chopstick 3
```