



Nama: **Bilhaq Avi Dewantara (120140141)**  
Mata Kuliah: **Sistem Operasi (IF2223)**

Tugas Ke: **02**  
Tanggal: 09/04/2022

## 1 Tujuan Hands On 2

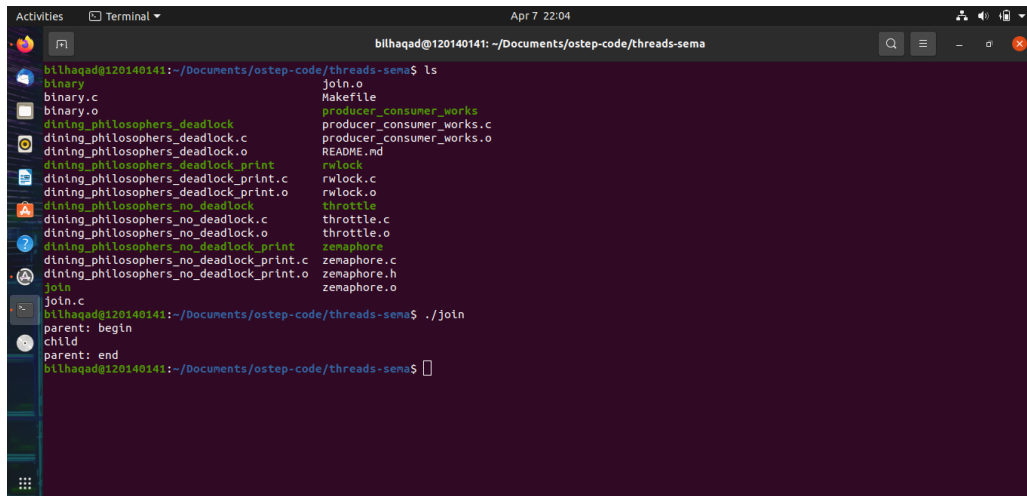
Tujuan adanya Hands On kedua adalah untuk memahami bagaimana sistem bersinkronisasi dan permasalahan yang ada, serta juga memahami solusinya saat menjalankan critical section. Adapun beberapa implementasi yang diharuskan untuk dipahami pada Hands On kedua ini antara lain : *join* menggunakan semaphores, *Binary Semaphores*, *Produces Consumer*, *Reader/Writer Locks*, dan *Dining Philosophers*.

## 2 Fork/Join

### 2.1 Source Code

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <pthread.h>
4  #include <unistd.h>
5
6  #include "common.h"
7  #include "common_threads.h"
8
9  #ifdef linux
10 #include <semaphore.h>
11 #elif __APPLE__
12 #include "zemaphore.h"
13 #endif
14
15 sem_t s;
16
17 void *child(void *arg) {
18     sleep(2);
19     printf("child\n");
20     Sem_post(&s); // signal here: child is done
21     return NULL;
22 }
23
24 int main(int argc, char *argv[]) {
25     Sem_init(&s, 0);
26     printf("parent: begin\n");
27     pthread_t c;
28     Pthread_create(&c, NULL, child, NULL);
29     Sem_wait(&s); // wait here for child
30     printf("parent: end\n");
31     return 0;
32 }
33
```

## 2.2 Output



Gambar 1: *Fork/Join*

## 2.3 Penjelasan Fork/Join

# 3 Binary Semaphores

## 3.1 Source Code

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <pthread.h>
4  #include <unistd.h>
5
6  #include "common.h"
7  #include "common_threads.h"
8
9  #ifdef linux
10 #include <semaphore.h>
11 #elif __APPLE__
12 #include "zemaaphore.h"
13 #endif
14
15 sem_t mutex;
16 volatile int counter = 0;
17
18 void *child(void *arg) {
19     int i;
20     for (i = 0; i < 100000000; i++) {
21         Sem_wait(&mutex);
22         counter++;
23         Sem_post(&mutex);
24     }
25     return NULL;
26 }
27
28 int main(int argc, char *argv[]) {
29     Sem_init(&mutex, 1);
30     pthread_t c1, c2;
31     Pthread_create(&c1, NULL, child, NULL);

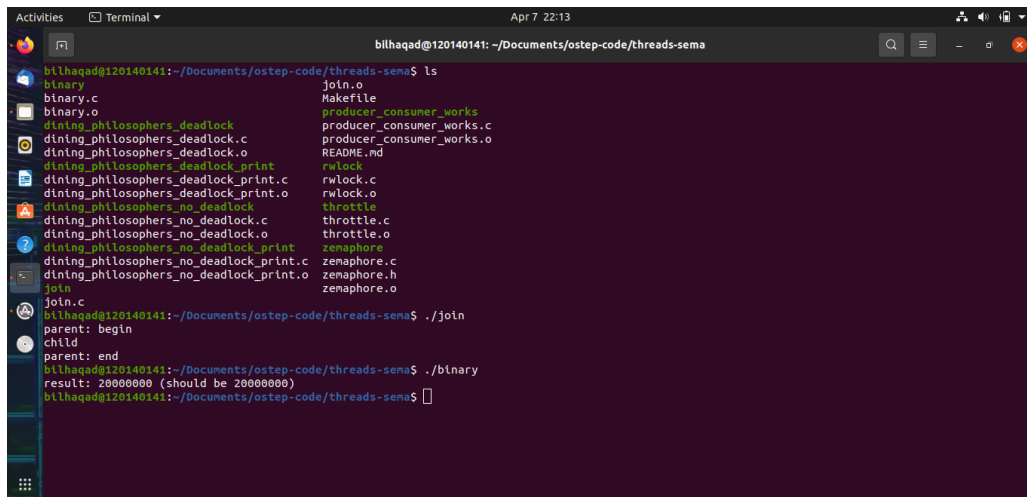
```

```

32  Pthread_create(&c2, NULL, child, NULL);
33  Pthread_join(c1, NULL);
34  Pthread_join(c2, NULL);
35  printf("result: %d (should be 20000000)\n", counter);
36  return 0;
37  }

```

### 3.2 Output



Gambar 2: Binary Semaphores

### 3.3 Penjelasan Binary Semaphores

## 4 Producer/Consumer

### 4.1 Source Code

```

1  #include <stdio.h>
2  #include <unistd.h>
3  #include <assert.h>
4  #include <pthread.h>
5  #include <stdlib.h>
6
7  #include "common.h"
8  #include "common-threads.h"
9
10 #ifdef linux
11 #include <semaphore.h>
12 #elif __APPLE__
13 #include "z semaphore.h"
14 #endif
15
16 int max;
17 int loops;
18 int *buffer;
19
20 int use = 0;
21 int fill = 0;
22
23 sem_t empty;

```

```
24 sem_t full;
25 sem_t mutex;
26
27 #define CMAX (10)
28 int consumers = 1;
29
30 void do_fill(int value) {
31     buffer[fill] = value;
32     fill++;
33     if (fill == max)
34         fill = 0;
35 }
36
37 int do_get() {
38     int tmp = buffer[use];
39     use++;
40     if (use == max)
41         use = 0;
42     return tmp;
43 }
44
45 void *producer(void *arg) {
46     int i;
47     for (i = 0; i < loops; i++) {
48         Sem_wait(&empty);
49         Sem_wait(&mutex);
50         do_fill(i);
51         Sem_post(&mutex);
52         Sem_post(&full);
53     }
54
55     // end case
56     for (i = 0; i < consumers; i++) {
57         Sem_wait(&empty);
58         Sem_wait(&mutex);
59         do_fill(-1);
60         Sem_post(&mutex);
61         Sem_post(&full);
62     }
63
64     return NULL;
65 }
66
67 void *consumer(void *arg) {
68     int tmp = 0;
69     while (tmp != -1) {
70         Sem_wait(&full);
71         Sem_wait(&mutex);
72         tmp = do_get();
73         Sem_post(&mutex);
74         Sem_post(&empty);
75         printf("%lld %d\n", (long long int) arg, tmp);
76     }
77     return NULL;
78 }
79
80 int main(int argc, char *argv[]) {
81     if (argc != 4) {
82         fprintf(stderr, "usage: %s <buffersize> <loops> <consumers>\n", argv[0]);
83         exit(1);
84     }
85     max = atoi(argv[1]);
```

```

86 loops = atoi(argv[2]);
87 consumers = atoi(argv[3]);
88 assert(consumers <= CMAX);
89
90 buffer = (int *) malloc(max * sizeof(int));
91 assert(buffer != NULL);
92 int i;
93 for (i = 0; i < max; i++) {
94     buffer[i] = 0;
95 }
96
97 Sem_init(&empty, max); // max are empty
98 Sem_init(&full, 0);    // 0 are full
99 Sem_init(&mutex, 1);   // mutex
100
101 pthread_t pid, cid[CMAX];
102 Pthread_create(&pid, NULL, producer, NULL);
103 for (i = 0; i < consumers; i++) {
104     Pthread_create(&cid[i], NULL, consumer, (void *) (long long int) i);
105 }
106 Pthread_join(pid, NULL);
107 for (i = 0; i < consumers; i++) {
108     Pthread_join(cid[i], NULL);
109 }
110 return 0;
111 }

```

## 4.2 Output

```

bilhaqad@120140141: ~/Documents/ostep-code/threads-sema
dining_philosophers_deadlock dining_philosophers_no_deadlock_print producer_consumer_works.o zenaphore.c
dining_philosophers_deadlock.c dining_philosophers_no_deadlock_print.c README.md zenaphore.h
dining_philosophers_deadlock.o dining_philosophers_no_deadlock_print.o rwlock.c zenaphore.o
dining_philosophers_deadlock_print join.c
dining_philosophers_deadlock_print.c join.o
dining_philosophers_deadlock_print.o join.o
bilhaqad@120140141:~/Documents/ostep-code/threads-sema$ ./producer_consumer_works 1 10 1
0 0
0 1
0 2
0 3
0 4
0 5
0 6
0 7
0 8
0 9
0 -1
bilhaqad@120140141:~/Documents/ostep-code/threads-sema$ ./producer_consumer_works 1 10 2
1 0
1 1
1 2
0 3
1 4
0 5
1 6
0 7
1 8
0 9
1 -1
0 -1
bilhaqad@120140141:~/Documents/ostep-code/threads-sema$

```

Gambar 3: *Producer/Consumer*

## 4.3 Penjelasan *Producer/Consumer*

# 5 Reader/Writer Locks

## 5.1 Source Code

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <pthread.h>

```

```
4  #include <unistd.h>
5
6  #include "common.h"
7  #include "common_threads.h"
8
9  #ifdef linux
10 #include <semaphore.h>
11 #elif __APPLE__
12 #include "zemaphore.h"
13 #endif
14
15 typedef struct _rwlock_t {
16     sem_t writelock;
17     sem_t lock;
18     int readers;
19 } rwlock_t;
20
21 void rwlock_init(rwlock_t *lock) {
22     lock->readers = 0;
23     Sem_init(&lock->lock, 1);
24     Sem_init(&lock->writelock, 1);
25 }
26
27 void rwlock_acquire_readlock(rwlock_t *lock) {
28     Sem_wait(&lock->lock);
29     lock->readers++;
30     if (lock->readers == 1)
31         Sem_wait(&lock->writelock);
32     Sem_post(&lock->lock);
33 }
34
35 void rwlock_release_readlock(rwlock_t *lock) {
36     Sem_wait(&lock->lock);
37     lock->readers--;
38     if (lock->readers == 0)
39         Sem_post(&lock->writelock);
40     Sem_post(&lock->lock);
41 }
42
43 void rwlock_acquire_writelock(rwlock_t *lock) {
44     Sem_wait(&lock->writelock);
45 }
46
47 void rwlock_release_writelock(rwlock_t *lock) {
48     Sem_post(&lock->writelock);
49 }
50
51 int read_loops;
52 int write_loops;
53 int counter = 0;
54
55 rwlock_t mutex;
56
57 void *reader(void *arg) {
58     int i;
59     int local = 0;
60     for (i = 0; i < read_loops; i++) {
61         rwlock_acquire_readlock(&mutex);
62         local = counter;
63         rwlock_release_readlock(&mutex);
64         printf("read %d\n", local);
65     }
```

```

66     printf("read done: %d\n", local);
67     return NULL;
68 }
69
70 void *writer(void *arg) {
71     int i;
72     for (i = 0; i < write_loops; i++) {
73         rwlock_acquire_writelock(&mutex);
74         counter++;
75         rwlock_release_writelock(&mutex);
76     }
77     printf("write done\n");
78     return NULL;
79 }
80
81 int main(int argc, char *argv[]) {
82     if (argc != 3) {
83         fprintf(stderr, "usage: rwlock readloops writeloops\n");
84         exit(1);
85     }
86     read_loops = atoi(argv[1]);
87     write_loops = atoi(argv[2]);
88
89     rwlock_init(&mutex);
90     pthread_t c1, c2;
91     Pthread_create(&c1, NULL, reader, NULL);
92     Pthread_create(&c2, NULL, writer, NULL);
93     Pthread_join(c1, NULL);
94     Pthread_join(c2, NULL);
95     printf("all done\n");
96     return 0;
97 }

```

## 5.2 Output

```

Activities Terminal Apr 7 22:50
bilhaqad@120140141: ~/Documents/ostep-code/threads-sema
dining_philosophers_deadlock.o dining_philosophers_no_deadlock_print.o zenaphore.o
dining_philosophers_deadlock_print.o join.c rwlock.c
dining_philosophers_deadlock_print.c join.c rwlock.o
dining_philosophers_deadlock_print.o join.o throttle
bilhaqad@120140141:~/Documents/ostep-code/threads-sema$ ./rwlock 10 10
write done
read 10
read 10
read 10
read 10
read 10
read 10
read 10
read 10
read 10
read 10
read done: 10
all done
bilhaqad@120140141:~/Documents/ostep-code/threads-sema$ ./rwlock 10 5
write done
read 5
read 5
read 5
read 5
read 5
read 5
read 5
read 5
read 5
read 5
read done: 5
all done

```

Gambar 4: Reader/Writer Locks

### 5.3 Penjelasan *Reader/Writer Locks*

## 6 Dining Philosophers

### 6.1 Deadlock

#### 6.1.1 Source Code

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <pthread.h>
4
5  #include "common.h"
6  #include "common_threads.h"
7
8  #ifdef linux
9  #include <semaphore.h>
10 #elif __APPLE__
11 #include "semaphore.h"
12 #endif
13
14 typedef struct {
15     int num_loops;
16     int thread_id;
17 } arg_t;
18
19 sem_t forks[5];
20 sem_t print_lock;
21
22 void space(int s) {
23     Sem_wait(&print_lock);
24     int i;
25     for (i = 0; i < s * 10; i++)
26         printf(" ");
27 }
28
29 void space_end() {
30     Sem_post(&print_lock);
31 }
32
33 int left(int p) {
34     return p;
35 }
36
37 int right(int p) {
38     return (p + 1) % 5;
39 }
40
41 void get_forks(int p) {
42     space(p); printf("%d: try %d\n", p, left(p)); space_end();
43     Sem_wait(&forks[left(p)]);
44     space(p); printf("%d: try %d\n", p, right(p)); space_end();
45     Sem_wait(&forks[right(p)]);
46 }
47
48 void put_forks(int p) {
49     Sem_post(&forks[left(p)]);
50     Sem_post(&forks[right(p)]);
51 }
52
53 void think() {
54     return;
```



```
55 }
56
57 void eat() {
58     return;
59 }
60
61 void *philosopher(void *arg) {
62     arg_t *args = (arg_t *) arg;
63
64     space(args->thread_id); printf("%d: start\n", args->thread_id); space_end();
65
66     int i;
67     for (i = 0; i < args->num_loops; i++) {
68         space(args->thread_id); printf("%d: think\n", args->thread_id); space_end();
69         think();
70         get_forks(args->thread_id);
71         space(args->thread_id); printf("%d: eat\n", args->thread_id); space_end();
72         eat();
73         put_forks(args->thread_id);
74         space(args->thread_id); printf("%d: done\n", args->thread_id); space_end();
75     }
76     return NULL;
77 }
78
79 int main(int argc, char *argv[]) {
80     if (argc != 2) {
81         fprintf(stderr, "usage: dining_philosophers <num_loops>\n");
82         exit(1);
83     }
84     printf("dining: started\n");
85
86     int i;
87     for (i = 0; i < 5; i++)
88         Sem_init(&forks[i], 1);
89     Sem_init(&print_lock, 1);
90
91     pthread_t p[5];
92     arg_t a[5];
93     for (i = 0; i < 5; i++) {
94         a[i].num_loops = atoi(argv[1]);
95         a[i].thread_id = i;
96         Pthread_create(&p[i], NULL, philosopher, &a[i]);
97     }
98
99     for (i = 0; i < 5; i++)
100         Pthread_join(p[i], NULL);
101
102     printf("dining: finished\n");
103     return 0;
104 }
```

### 6.1.2 Output

```

Activities Terminal Apr 7 22:33
bilhaqad@120140141: ~/Documents/ostep-code/threads-sema
bilhaqad@120140141:~/Documents/ostep-code/threads-sema$ ./dining_philosophers_deadlock 2
dining: started
dining: finished
bilhaqad@120140141:~/Documents/ostep-code/threads-sema$ ./dining_philosophers_deadlock_print 2
dining: started
                2: start
                2: think
                2: try 2
                2: try 3
                2: eat
                2: done
0: start
0: think
0: try 0
0: try 1
0: eat
0: done
0: think
0: try 0
0: try 1
0: eat
0: done
1: start
1: think
1: try 1
1: try 2
1: eat
1: done
1: think
1: try 1
                2: think
                2: try 2

Activities Terminal Apr 7 22:34
bilhaqad@120140141: ~/Documents/ostep-code/threads-sema
                2: think
                2: try 2
                2: try 3
                2: eat
                2: done
1: try 2
1: eat
1: done
                3: start
                3: think
                3: try 3
                3: try 4
                3: eat
                3: done
                3: think
                3: try 3
                3: try 4
                3: eat
                3: done
4: start
4: think
4: try 4
4: try 0
4: eat
4: done
4: think
4: try 4
4: try 0
4: eat
4: done
dining: finished
bilhaqad@120140141:~/Documents/ostep-code/threads-sema$

```

### 6.1.3 Penjelasan Dining Philosophers Deadlock

## 6.2 No Deadlock

### 6.2.1 Source Code

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <pthread.h>
4
5  #include "common.h"
6  #include "common_threads.h"
7
8  #ifdef linux
9  #include <semaphore.h>
10 #elif __APPLE__
11 #include "zemaphore.h"
12 #endif
13
14 typedef struct {
15     int num_loops;
16     int thread_id;
17 } arg_t;
18
19 sem_t forks[5];
20 sem_t print_lock;
21
22 void space(int s) {
23     Sem_wait(&print_lock);
24     int i;
25     for (i = 0; i < s * 10; i++)
26         printf(" ");
27 }
28
29 void space_end() {
30     Sem_post(&print_lock);
31 }
32
33 int left(int p) {
34     return p;
35 }
36
37 int right(int p) {
38     return (p + 1) % 5;
39 }
40
41 void get_forks(int p) {
42     if (p == 4) {
43         space(p); printf("4 try %d\n", right(p)); space_end();
44         Sem_wait(&forks[right(p)]);
45         space(p); printf("4 try %d\n", left(p)); space_end();
46         Sem_wait(&forks[left(p)]);
47     } else {
48         space(p); printf("try %d\n", left(p)); space_end();
49         Sem_wait(&forks[left(p)]);
50         space(p); printf("try %d\n", right(p)); space_end();
51         Sem_wait(&forks[right(p)]);
52     }
53 }
54
55 void put_forks(int p) {
56     Sem_post(&forks[left(p)]);
57     Sem_post(&forks[right(p)]);
58 }
```

```
59 void think() {
60     return;
61 }
62
63 void eat() {
64     return;
65 }
66
67 void *philosopher(void *arg) {
68     arg_t *args = (arg_t *) arg;
69
70     space(args->thread_id); printf("%d: start\n", args->thread_id); space_end();
71
72     int i;
73     for (i = 0; i < args->num_loops; i++) {
74         space(args->thread_id); printf("%d: think\n", args->thread_id); space_end();
75         think();
76         get_forks(args->thread_id);
77         space(args->thread_id); printf("%d: eat\n", args->thread_id); space_end();
78         eat();
79         put_forks(args->thread_id);
80         space(args->thread_id); printf("%d: done\n", args->thread_id); space_end();
81     }
82     return NULL;
83 }
84
85 int main(int argc, char *argv[]) {
86     if (argc != 2) {
87         fprintf(stderr, "usage: dining_philosophers <num_loops>\n");
88         exit(1);
89     }
90     printf("dining: started\n");
91
92     int i;
93     for (i = 0; i < 5; i++)
94         Sem_init(&forks[i], 1);
95     Sem_init(&print_lock, 1);
96
97     pthread_t p[5];
98     arg_t a[5];
99     for (i = 0; i < 5; i++) {
100         a[i].num_loops = atoi(argv[1]);
101         a[i].thread_id = i;
102         Pthread_create(&p[i], NULL, philosopher, &a[i]);
103     }
104
105     for (i = 0; i < 5; i++)
106         Pthread_join(p[i], NULL);
107
108     printf("dining: finished\n");
109     return 0;
110 }
111 }
```

## Hands On 2: Synchronisation and Deadlock

## 7 Kesimpulan

Pada Hands On 1 ini yang saya dapatkan setelah menjalankannya ialah saya dapat mengenal sistem operasi linux khususnya Ubuntu 20.04 LTS ini meskipun hanya menggunakan Oracle VirtualBox. Selain itu, saya dapat mengetahui banyak hal dari tugas ini yaitu itu menggunakan konsep-konsep baru dan mengimplementasikannya pada terminal di linux. Dengan begitu, saya dapat menyelesaikan tugas ini sesuai dengan kemampuan yang saya miliki terutama menggunakan latex ini yang baru bagi saya, sehingga banyak sekali yang saya dapatkan dari tugas ini.

## 8 Link GitHub

Link GitHub dari Hands On 2 ini : [Klik disini](#)