



Nama: **Muhammad Qomarudin (120140116)**  
Mata Kuliah: **Sistem Operasi (IF2223)**

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

## 1 Tujuan HandsOn

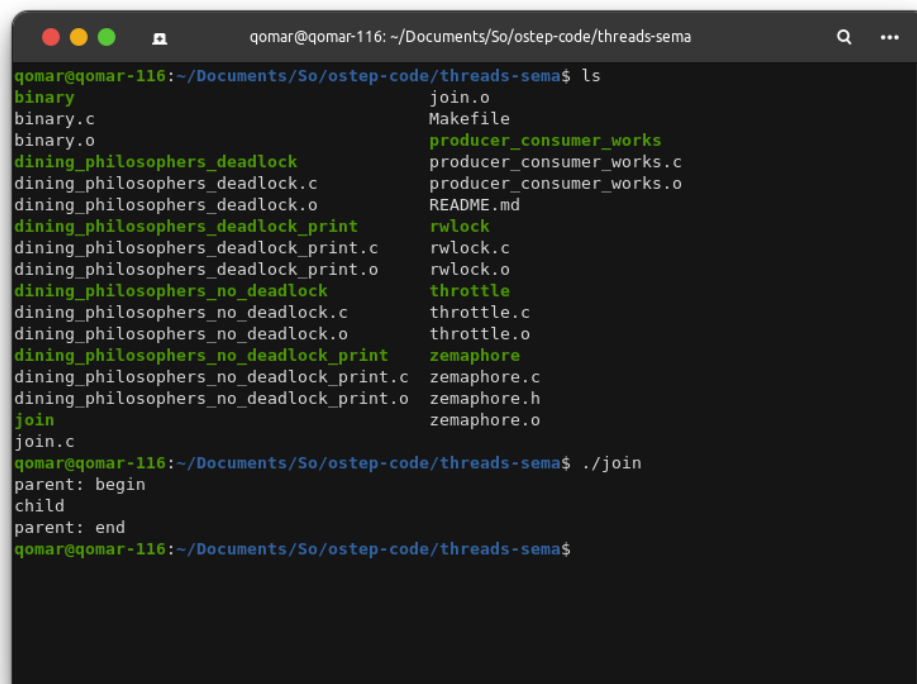
pengerjaan HandsOn ini memiliki beberapa tujuan yang di antaranya yaitu :

- memahami sistem sinkronisasi dan permasalahan yang terjadi dalam sistem Operasi
- Memahami solusi dalam menangani *critical section*
- memahami implementasi dari :
  - *join* menggunakan *semaphores*
  - *Binary semaphores*
  - *Producer Consumer*
  - *Reader / Writer*
  - *Dining Philosophers*

## 2 Fork/Join

### 2.1 source code

```
1  sem_t s;  
2  
3  void *child(void *arg) {  
4      sleep(2);  
5      printf("child\n");  
6      Sem_post(&s); // signal here: child is done  
7      return NULL;  
8  }  
9  
10 int main(int argc, char *argv[]) {  
11     Sem_init(&s, 0);  
12     printf("parent: begin\n");  
13     pthread_t c;  
14     Pthread_create(&c, NULL, child, NULL);  
15     Sem_wait(&s); // wait here for child  
16     printf("parent: end\n");  
17     return 0;  
18 }
```

A terminal window titled 'qomar@qomar-116: ~/Documents/So/ostep-code/threads-sema'. The user runs 'ls' and lists files: binary, binary.c, binary.o, dining\_philosophers\_deadlock, dining\_philosophers\_deadlock.c, dining\_philosophers\_deadlock.o, dining\_philosophers\_deadlock\_print, dining\_philosophers\_deadlock\_print.c, dining\_philosophers\_deadlock\_print.o, dining\_philosophers\_no\_deadlock, dining\_philosophers\_no\_deadlock.c, dining\_philosophers\_no\_deadlock.o, dining\_philosophers\_no\_deadlock\_print, dining\_philosophers\_no\_deadlock\_print.c, dining\_philosophers\_no\_deadlock\_print.o, join, join.c, join.o, Makefile, producer\_consumer\_works, producer\_consumer\_works.c, producer\_consumer\_works.o, README.md, rwlock, rwlock.c, rwlock.o, throttle, throttle.c, throttle.o, zemaphore, zemaphore.c, zemaphore.h, zemaphore.o. Then the user runs './join' and the output is: 'parent: begin', 'child', 'parent: end'.

```
qomar@qomar-116: ~/Documents/So/ostep-code/threads-sema$ ls
binary
binary.c
binary.o
dining_philosophers_deadlock
dining_philosophers_deadlock.c
dining_philosophers_deadlock.o
dining_philosophers_deadlock_print
dining_philosophers_deadlock_print.c
dining_philosophers_deadlock_print.o
dining_philosophers_no_deadlock
dining_philosophers_no_deadlock.c
dining_philosophers_no_deadlock.o
dining_philosophers_no_deadlock_print
dining_philosophers_no_deadlock_print.c
dining_philosophers_no_deadlock_print.o
join
join.c
join.o
Makefile
producer_consumer_works
producer_consumer_works.c
producer_consumer_works.o
README.md
rwlock
rwlock.c
rwlock.o
throttle
throttle.c
throttle.o
zemaphore
zemaphore.c
zemaphore.h
zemaphore.o
qomar@qomar-116: ~/Documents/So/ostep-code/threads-sema$ ./join
parent: begin
child
parent: end
qomar@qomar-116: ~/Documents/So/ostep-code/threads-sema$
```

Gambar 1: Fork/Join

output

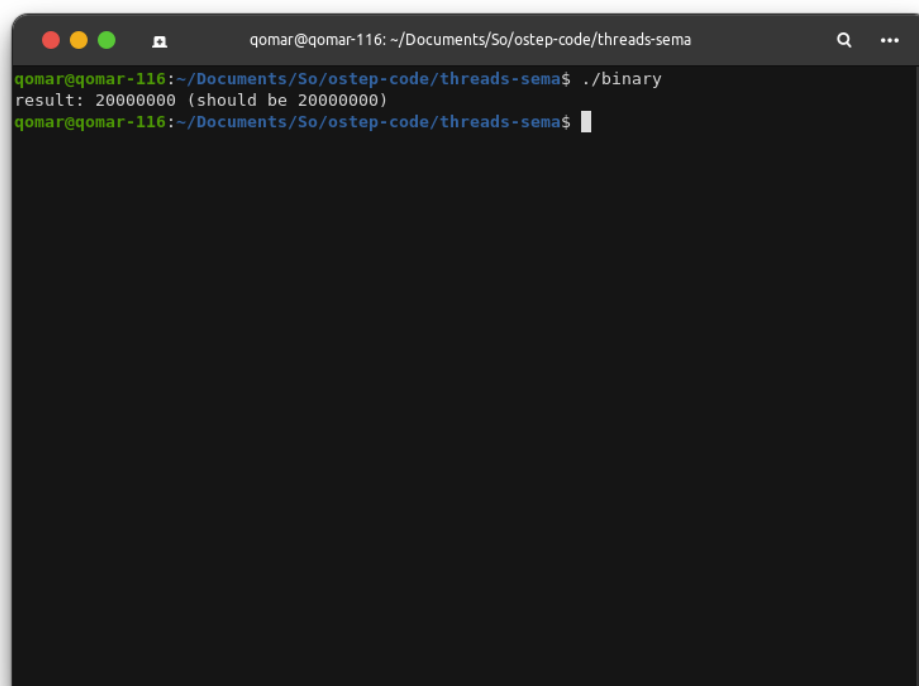
## 2.2 penjelasan

### 3 Binary Semaphores

#### source code

```
1  volatile int counter = 0;
2
3
4  void *child(void *arg) {
5      int i;
6      for (i = 0; i < 10000000; i++) {
7          Sem_wait(&mutex);
8          counter++;
9          Sem_post(&mutex);
10     }
11     return NULL;
12 }
13
14 int main(int argc, char *argv[]) {
15     Sem_init(&mutex, 1);
16     pthread_t c1, c2;
17     Pthread_create(&c1, NULL, child, NULL);
18     Pthread_create(&c2, NULL, child, NULL);
19     Pthread_join(c1, NULL);
20     Pthread_join(c2, NULL);
21     printf("result: %d (should be 20000000)\n", counter);
22     return 0;
23 }
```

#### output



```
qomar@qomar-116: ~/Documents/So/ostep-code/threads-sema
qomar@qomar-116:~/Documents/So/ostep-code/threads-sema$ ./binary
result: 20000000 (should be 20000000)
qomar@qomar-116:~/Documents/So/ostep-code/threads-sema$
```

Gambar 2: Binary Semaphores

### 3.1 penjelasan

## 4 *Producer Consumer*

### 4.1 source code

```
1  int max;
2  int loops;
3  int *buffer;
4
5  int use = 0;
6  int fill = 0;
7
8  sem_t empty;
9  sem_t full;
10 sem_t mutex;
11
12 #define CMAX (10)
13 int consumers = 1;
14
15 void do_fill(int value) {
16     buffer[fill] = value;
17     fill++;
18     if (fill == max)
19         fill = 0;
20 }
21
22 int do_get() {
23     int tmp = buffer[use];
24     use++;
25     if (use == max)
26         use = 0;
27     return tmp;
28 }
29
30 void *producer(void *arg) {
31     int i;
32     for (i = 0; i < loops; i++) {
33         Sem_wait(&empty);
34         Sem_wait(&mutex);
35         do_fill(i);
36         Sem_post(&mutex);
37         Sem_post(&full);
38     }
39
40     // end case
41     for (i = 0; i < consumers; i++) {
42         Sem_wait(&empty);
43         Sem_wait(&mutex);
44         do_fill(-1);
45         Sem_post(&mutex);
46         Sem_post(&full);
47     }
48
49     return NULL;
50 }
51
52 void *consumer(void *arg) {
53     int tmp = 0;
54     while (tmp != -1) {
55         Sem_wait(&full);
56         Sem_wait(&mutex);
57         tmp = do_get();
58         Sem_post(&mutex);
```

```

59 Sem_post(&empty);
60 printf("%lld %d\n", (long long int) arg, tmp);
61 }
62 return NULL;
63 }
64
65 int main(int argc, char *argv[]) {
66     if (argc != 4) {
67         fprintf(stderr, "usage: %s <buffersize> <loops> <consumers>\n", argv[0]);
68         exit(1);
69     }
70     max = atoi(argv[1]);
71     loops = atoi(argv[2]);
72     consumers = atoi(argv[3]);
73     assert(consumers <= CMAX);
74
75     buffer = (int *) malloc(max * sizeof(int));
76     assert(buffer != NULL);
77     int i;
78     for (i = 0; i < max; i++) {
79         buffer[i] = 0;
80     }
81
82     Sem_init(&empty, max); // max are empty
83     Sem_init(&full, 0);    // 0 are full
84     Sem_init(&mutex, 1);   // mutex
85
86     pthread_t pid, cid[CMAX];
87     Pthread_create(&pid, NULL, producer, NULL);
88     for (i = 0; i < consumers; i++) {
89         Pthread_create(&cid[i], NULL, consumer, (void *) (long long int) i);
90     }
91     Pthread_join(pid, NULL);
92     for (i = 0; i < consumers; i++) {
93         Pthread_join(cid[i], NULL);
94     }
95     return 0;
96 }

```

## output

```

qomar@qomar-116: ~/Documents/So/ostep-code/threads-sema
qomar@qomar-116:~/Documents/So/ostep-code/threads-sema$ ./producer_consumer_works 2 5 3
0 0
1 2
0 3
2 1
1 4
1 -1
0 -1
2 -1
qomar@qomar-116:~/Documents/So/ostep-code/threads-sema$

```

Gambar 3: Producer Consumer

## 4.2 penjelasan

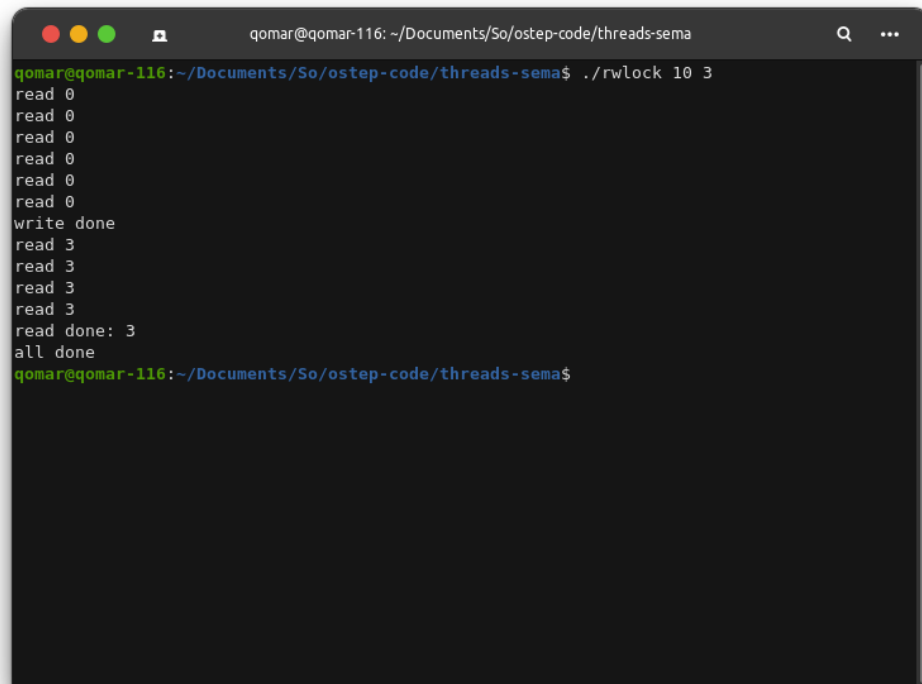
## 5 Reader / Writer

### 5.1 source code

```
1  int read_loops;
2  int write_loops;
3  int counter = 0;
4
5  rwlock_t mutex;
6
7  void *reader(void *arg) {
8      int i;
9      int local = 0;
10     for (i = 0; i < read_loops; i++) {
11         rwlock_acquire_readlock(&mutex);
12         local = counter;
13         rwlock_release_readlock(&mutex);
14         printf("read %d\n", local);
15     }
16     printf("read done: %d\n", local);
17     return NULL;
18 }
19
20 void *writer(void *arg) {
21     int i;
22     for (i = 0; i < write_loops; i++) {
23         rwlock_acquire_writelock(&mutex);
24         counter++;
25         rwlock_release_writelock(&mutex);
26     }
27     printf("write done\n");
28     return NULL;
29 }
30
31 int main(int argc, char *argv[]) {
32     if (argc != 3) {
33         fprintf(stderr, "usage: rwlock readloops writeloops\n");
34         exit(1);
35     }
36     read_loops = atoi(argv[1]);
37     write_loops = atoi(argv[2]);
38
39     rwlock_init(&mutex);
40     pthread_t c1, c2;
41     Pthread_create(&c1, NULL, reader, NULL);
42     Pthread_create(&c2, NULL, writer, NULL);
43     Pthread_join(c1, NULL);
44     Pthread_join(c2, NULL);
45     printf("all done\n");
46     return 0;
47 }
```



## output



```
qomar@qomar-116: ~/Documents/So/ostep-code/threads-sema$ ./rwlock 10 3
read 0
read 0
read 0
read 0
read 0
read 0
read 0
read 0
write done
read 3
read 3
read 3
read 3
read 3
read done: 3
all done
qomar@qomar-116: ~/Documents/So/ostep-code/threads-sema$
```

Gambar 4: Reader / Writer

## 5.2 penjelasan

## 6 Dining Philosophers

### 6.1 Dining Philosophers Deadlock

#### 6.1.1 source code

```
1
2 void space(int s) {
3     Sem_wait(&print_lock);
4     int i;
5     for (i = 0; i < s * 10; i++)
6         printf(" ");
7 }
8
9 void space_end() {
10    Sem_post(&print_lock);
11 }
12
13 int left(int p) {
14     return p;
15 }
16
17 int right(int p) {
18     return (p + 1) % 5;
19 }
20
21 void get_forks(int p) {
22     space(p); printf("%d: try %d\n", p, left(p)); space_end();
23     Sem_wait(&forks[left(p)]);
24     space(p); printf("%d: try %d\n", p, right(p)); space_end();
25     Sem_wait(&forks[right(p)]);
26 }
27
28 void put_forks(int p) {
29     Sem_post(&forks[left(p)]);
30     Sem_post(&forks[right(p)]);
31 }
32
33 void think() {
34     return;
35 }
36
37 void eat() {
38     return;
39 }
40
41 void *philosopher(void *arg) {
42     arg_t *args = (arg_t *) arg;
43
44     space(args->thread_id); printf("%d: start\n", args->thread_id); space_end();
45
46     int i;
47     for (i = 0; i < args->num_loops; i++) {
48         space(args->thread_id); printf("%d: think\n", args->thread_id); space_end();
49         think();
50         get_forks(args->thread_id);
51         space(args->thread_id); printf("%d: eat\n", args->thread_id); space_end();
52         eat();
53         put_forks(args->thread_id);
54         space(args->thread_id); printf("%d: done\n", args->thread_id); space_end();
55     }
56     return NULL;
```

```

57 }
58
59 int main(int argc, char *argv[]) {
60     if (argc != 2) {
61         fprintf(stderr, "usage: dining_philosophers <num_loops>\n");
62         exit(1);
63     }
64     printf("dining: started\n");
65
66     int i;
67     for (i = 0; i < 5; i++)
68         Sem_init(&forks[i], 1);
69     Sem_init(&print_lock, 1);
70
71     pthread_t p[5];
72     arg_t a[5];
73     for (i = 0; i < 5; i++) {
74         a[i].num_loops = atoi(argv[1]);
75         a[i].thread_id = i;
76         Pthread_create(&p[i], NULL, philosopher, &a[i]);
77     }
78
79     for (i = 0; i < 5; i++)
80         Pthread_join(p[i], NULL);
81
82     printf("dining: finished\n");
83     return 0;
84 }

```

output

```

dining: started
0: start
0: think
0: try 0
0: try 1
0: eat
0: done
0: think
0: try 0
2: start
2: think
2: try 2
2: try 3
2: eat
2: done
0: try 1
0: eat
1: start
1: think
1: try 1
2: think
2: try 2
3: start
4: start
4: think
4: try 4
4: try 0
4: eat
4: done
1: try 2
2: try 3
2: eat
2: done
1: eat
1: done
4: think
4: try 4
4: try 0
4: eat
4: done
1: think
1: try 1
1: try 2
1: eat
1: done
3: think
3: try 3
3: try 4
3: eat
3: done
3: think
3: try 3
3: try 4
3: eat
3: done
dining: finished

```

Gambar 5: Dining Philosophers Deadlock

### 6.1.2 penjelasan

## 6.2 Dining Philosophers no Deadlock

### 6.2.1 source code

```

1  void space(int s) {
2      Sem_wait(&print_lock);
3      int i;
4      for (i = 0; i < s * 10; i++)
5          printf(" ");
6  }
7
8  void space_end() {
9      Sem_post(&print_lock);
10 }
11
12 int left(int p) {
13     return p;
14 }
15
16 int right(int p) {
17     return (p + 1) % 5;
18 }
19
20 void get_forks(int p) {
21     if (p == 4) {
22         space(p); printf("4 try %d\n", right(p)); space_end();
23         Sem_wait(&forks[right(p)]);
24         space(p); printf("4 try %d\n", left(p)); space_end();
25         Sem_wait(&forks[left(p)]);
26     } else {
27         space(p); printf("try %d\n", left(p)); space_end();
28         Sem_wait(&forks[left(p)]);
29         space(p); printf("try %d\n", right(p)); space_end();
30         Sem_wait(&forks[right(p)]);
31     }
32 }
33
34 void put_forks(int p) {
35     Sem_post(&forks[left(p)]);
36     Sem_post(&forks[right(p)]);
37 }
38
39 void think() {
40     return;
41 }
42
43 void eat() {
44     return;
45 }
46
47 void *philosopher(void *arg) {
48     arg_t *args = (arg_t *) arg;
49
50     space(args->thread_id); printf("%d: start\n", args->thread_id); space_end();
51
52     int i;
53     for (i = 0; i < args->num_loops; i++) {
54         space(args->thread_id); printf("%d: think\n", args->thread_id); space_end();
55         think();
56         get_forks(args->thread_id);
57         space(args->thread_id); printf("%d: eat\n", args->thread_id); space_end();
58         eat();

```

```

59     put_forks(args->thread_id);
60     space(args->thread_id); printf("%d: done\n", args->thread_id); space_end();
61     }
62     return NULL;
63 }
64
65 int main(int argc, char *argv[]) {
66     if (argc != 2) {
67         fprintf(stderr, "usage: dining_philosophers <num_loops>\n");
68         exit(1);
69     }
70     printf("dining: started\n");
71
72     int i;
73     for (i = 0; i < 5; i++)
74         Sem_init(&forks[i], 1);
75     Sem_init(&print_lock, 1);
76
77     pthread_t p[5];
78     arg_t a[5];
79     for (i = 0; i < 5; i++) {
80         a[i].num_loops = atoi(argv[1]);
81         a[i].thread_id = i;
82         Pthread_create(&p[i], NULL, philosopher, &a[i]);
83     }
84
85     for (i = 0; i < 5; i++)
86         Pthread_join(p[i], NULL);
87
88     printf("dining: finished\n");
89     return 0;
90 }

```

### output

```

dining: started
0: start
0: think
0: try 0
0: eat
0: done
0: think
0: try 0
0: eat
0: done
1: start
1: think
1: try 1
1: eat
1: done
1: think
1: try 1
1: eat
1: done
2: start
2: think
2: try 2
2: eat
2: done
3: start
3: think
3: try 3
3: eat
3: done
4: start
4: think
4: try 4
4: eat
4: done
dining: finished

```

Gambar 6: Dining Philosophers no Deadlock

### 6.2.2 penjelasan

## **7 kesimpulan**

## **8 link laporan dan berkas**

- Link github : [HandsOn\\_2\\_120140116\\_SO](#)