Program Studi Teknik Informatika
Institut Teknologi Sumatera

Nama: **Muhammad Qomarudin (120140116)** Tugas Ke: **02**
Mata Kuliah: **Sistem Operasi (IF2223)** Tanggal: 09/04/2022

1 Tujuan HandsOn

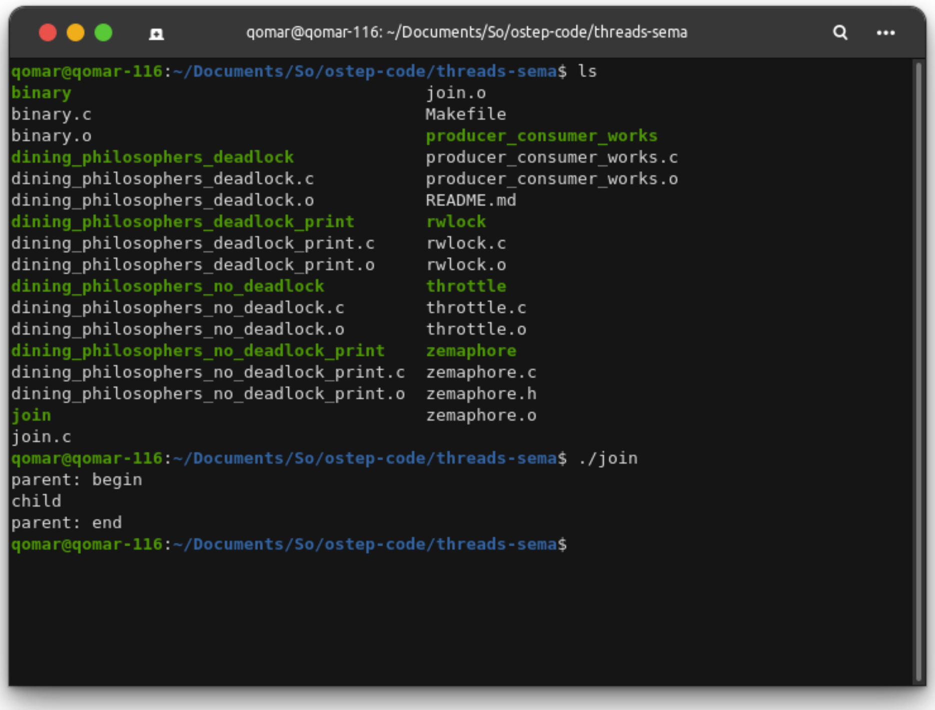
pengerjaan HandsOn ini memeiliki beberapa jutauan yang di antaranya yaitu :

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

2 *Fork/Join*

2.1 souce code

```
sem_t s;

void *child(void *arg) {
  sleep(2);
  printf("child\n");
  Sem_post(&s); // signal here: child is done
  return NULL;
}

int main(int argc, char *argv[]) {
  Sem_init(&s, 0);
  printf("parent: begin\n");
  pthread_t c;
  Pthread_create(&c, NULL, child, NULL);
  Sem_wait(&s); // wait here for child
  printf("parent: end\n");
  return 0;
}
```



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

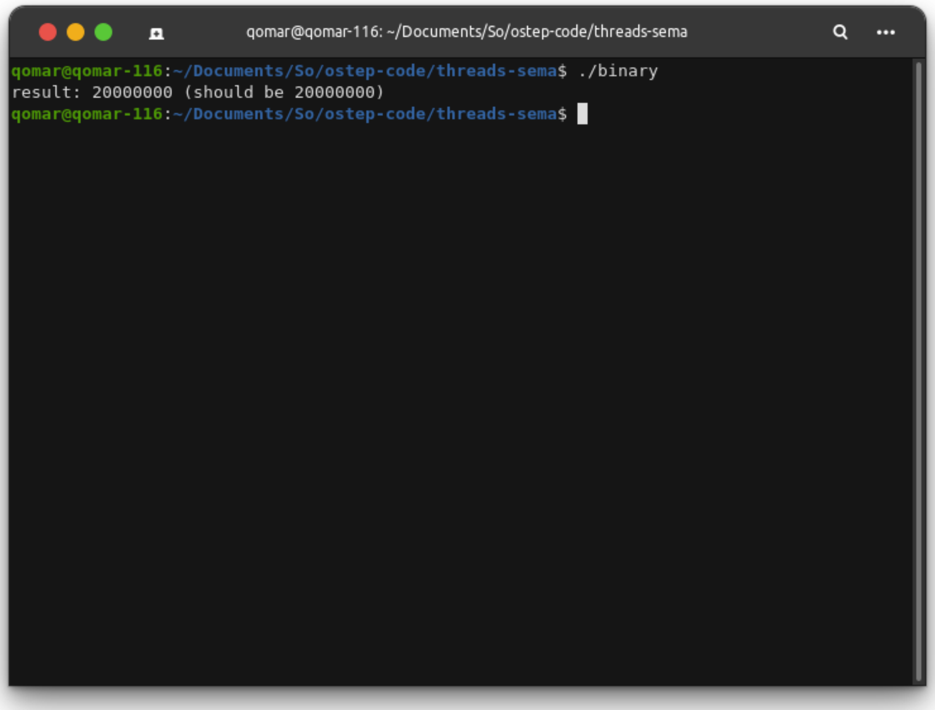
2.2 penjelasan

3 *Binary Semaphores*

souce code

```c

volatile int counter = 0;

void *child(void *arg) {
  int i;
  for (i = 0; i < 10000000; i++) {
  Sem_wait(&mutex);
  counter++;
  Sem_post(&mutex);
  }
  return NULL;
}

int main(int argc, char *argv[]) {
  Sem_init(&mutex, 1);
  pthread_t c1, c2;
  Pthread_create(&c1, NULL, child, NULL);
  Pthread_create(&c2, NULL, child, NULL);
  Pthread_join(c1, NULL);
  Pthread_join(c2, NULL);
  printf("result: %d (should be 20000000)\n", counter);
  return 0;
}
```

output



```
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

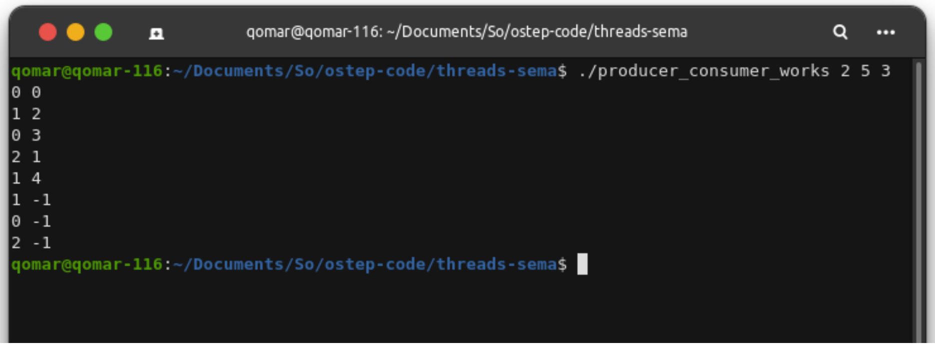
3.1 penjelasan

4 *Producer Consumer*

4.1 souce code

```c
int max;
int loops;
int *buffer;

int use  = 0;
int fill = 0;

sem_t empty;
sem_t full;
sem_t mutex;

#define CMAX (10)
int consumers = 1;

void do_fill(int value) {
  buffer[fill] = value;
  fill++;
  if (fill == max)
  fill = 0;
}

int do_get() {
  int tmp = buffer[use];
  use++;
  if (use == max)
  use = 0;
  return tmp;
}

void *producer(void *arg) {
  int i;
  for (i = 0; i < loops; i++) {
  Sem_wait(&empty);
  Sem_wait(&mutex);
  do_fill(i);
  Sem_post(&mutex);
  Sem_post(&full);
  }

  // end case
  for (i = 0; i < consumers; i++) {
  Sem_wait(&empty);
  Sem_wait(&mutex);
  do_fill(-1);
  Sem_post(&mutex);
  Sem_post(&full);
  }

  return NULL;
}

void *consumer(void *arg) {
  int tmp = 0;
  while (tmp != -1) {
  Sem_wait(&full);
  Sem_wait(&mutex);
  tmp = do_get();
  Sem_post(&mutex);
  Sem_post(&empty);
  printf("%lld %d\n", (long long int) arg, tmp);
  }
  return NULL;
}

int main(int argc, char *argv[]) {
  if (argc != 4) {
  fprintf(stderr, "usage: %s <buffersize> <loops> <consumers>\n", argv[0]);
  exit(1);
  }
  max   = atoi(argv[1]);
  loops = atoi(argv[2]);
  consumers = atoi(argv[3]);
  assert(consumers <= CMAX);

  buffer = (int *) malloc(max * sizeof(int));
  assert(buffer != NULL);
  int i;
  for (i = 0; i < max; i++) {
  buffer[i] = 0;
  }

  Sem_init(&empty, max); // max are empty
  Sem_init(&full, 0);    // 0 are full
  Sem_init(&mutex, 1);   // mutex

  pthread_t pid, cid[CMAX];
  Pthread_create(&pid, NULL, producer, NULL);
  for (i = 0; i < consumers; i++) {
  Pthread_create(&cid[i], NULL, consumer, (void *) (long long int) i);
  }
  Pthread_join(pid, NULL);
  for (i = 0; i < consumers; i++) {
  Pthread_join(cid[i], NULL);
  }
  return 0;
}
```

output



```
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

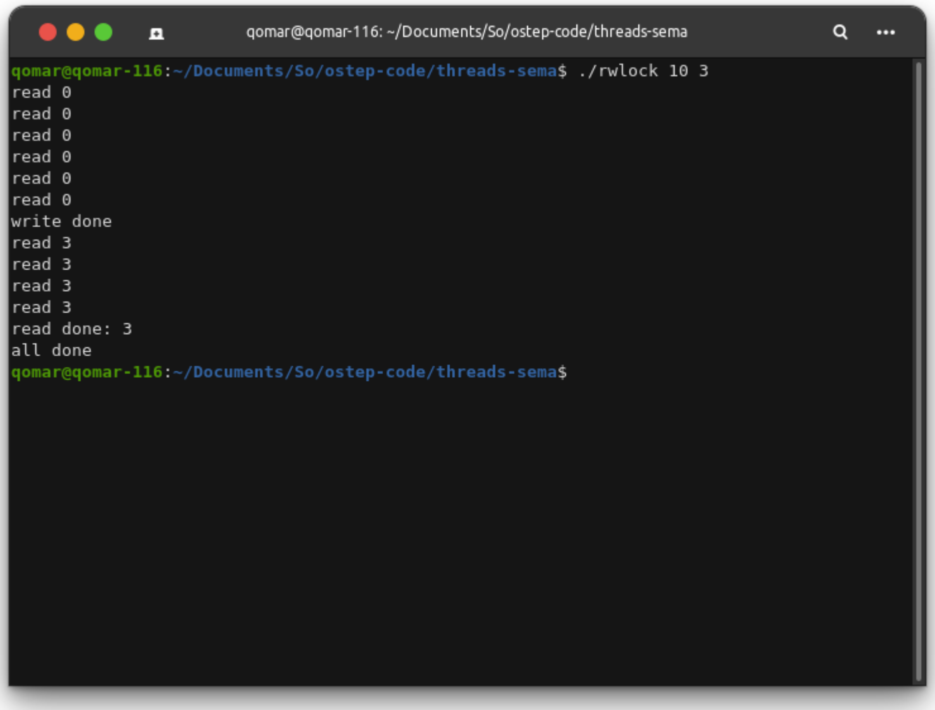
4.2 penjelasan

5 *Reader / Writer*

5.1 souce code

```c
int read_loops;
int write_loops;
int counter = 0;

rwlock_t mutex;

void *reader(void *arg) {
  int i;
  int local = 0;
  for (i = 0; i < read_loops; i++) {
  rwlock_acquire_readlock(&mutex);
  local = counter;
  rwlock_release_readlock(&mutex);
  printf("read %d\n", local);
  }
  printf("read done: %d\n", local);
  return NULL;
}

void *writer(void *arg) {
  int i;
  for (i = 0; i < write_loops; i++) {
  rwlock_acquire_writelock(&mutex);
  counter++;
  rwlock_release_writelock(&mutex);
  }
  printf("write done\n");
  return NULL;
}

int main(int argc, char *argv[]) {
  if (argc != 3) {
  fprintf(stderr, "usage: rwlock readloops writeloops\n");
  exit(1);
  }
  read_loops = atoi(argv[1]);
  write_loops = atoi(argv[2]);

  rwlock_init(&mutex);
  pthread_t c1, c2;
  Pthread_create(&c1, NULL, reader, NULL);
  Pthread_create(&c2, NULL, writer, NULL);
  Pthread_join(c1, NULL);
  Pthread_join(c2, NULL);
  printf("all done\n");
  return 0;
}
```

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
write done
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

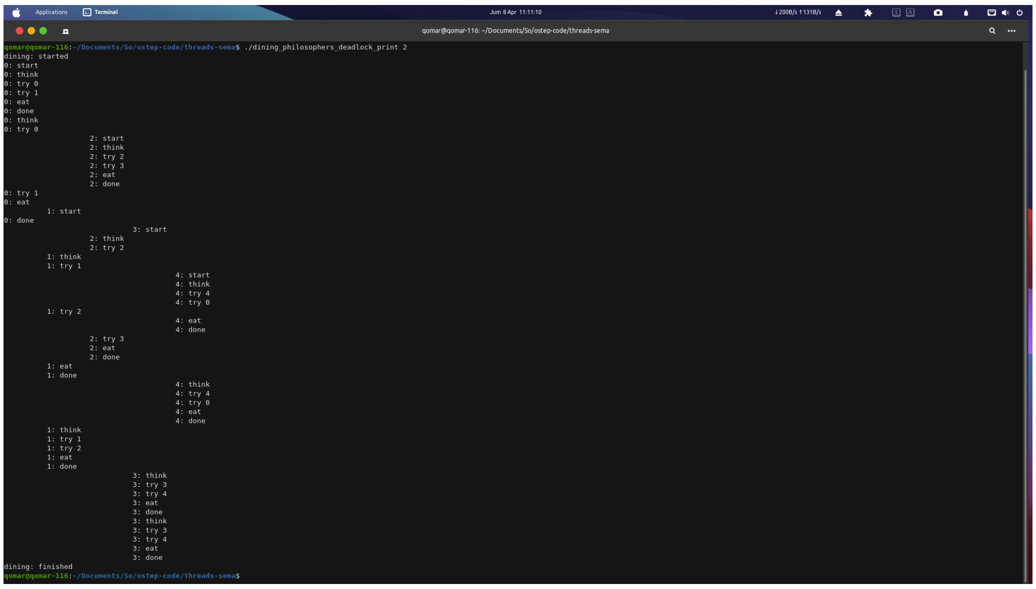
6 *Dining Philosophers*

6.1 *Dining Philosophers Deadlock*

6.1.1 source code

```c

void space(int s) {
  Sem_wait(&print_lock);
  int i;
  for (i = 0; i < s * 10; i++)
  printf(" ");
}

void space_end() {
  Sem_post(&print_lock);
}

int left(int p)  {
  return p;
}

int right(int p) {
  return (p + 1) % 5;
}

void get_forks(int p) {
  space(p); printf("%d: try %d\n", p, left(p)); space_end();
  Sem_wait(&forks[left(p)]);
  space(p); printf("%d: try %d\n", p, right(p)); space_end();
  Sem_wait(&forks[right(p)]);
}

void put_forks(int p) {
  Sem_post(&forks[left(p)]);
  Sem_post(&forks[right(p)]);
}

void think() {
  return;
}

void eat() {
  return;
}

void *philosopher(void *arg) {
  arg_t *args = (arg_t *) arg;

  space(args->thread_id); printf("%d: start\n", args->thread_id); space_end();

  int i;
  for (i = 0; i < args->num_loops; i++) {
  space(args->thread_id); printf("%d: think\n", args->thread_id); space_end();
  think();
  get_forks(args->thread_id);
  space(args->thread_id); printf("%d: eat\n", args->thread_id); space_end();
  eat();
  put_forks(args->thread_id);
  space(args->thread_id); printf("%d: done\n", args->thread_id); space_end();
  }
  return NULL;
}

int main(int argc, char *argv[]) {
  if (argc != 2) {
  fprintf(stderr, "usage: dining_philosophers <num_loops>\n");
  exit(1);
  }
  printf("dining: started\n");

  int i;
  for (i = 0; i < 5; i++)
  Sem_init(&forks[i], 1);
  Sem_init(&print_lock, 1);

  pthread_t p[5];
  arg_t a[5];
  for (i = 0; i < 5; i++) {
  a[i].num_loops = atoi(argv[1]);
  a[i].thread_id = i;
  Pthread_create(&p[i], NULL, philosopher, &a[i]);
  }

  for (i = 0; i < 5; i++)
  Pthread_join(p[i], NULL);

  printf("dining: finished\n");
  return 0;
}
```

output



```
qomar@qomar-116:~/Documents/So/ostep-code/threads-sema$ ./dining_philosophers_deadlock_print 2
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
0: done
                              3: start
                    2: think
                    2: try 2
          1: think
          1: try 1
                                        4: start
                                        4: think
                                        4: try 4
                                        4: try 0
          1: try 2
                                        4: eat
                                        4: done
                    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
qomar@qomar-116:~/Documents/So/ostep-code/threads-sema$
```

Gambar 5: Dining Philosophers Deadlock

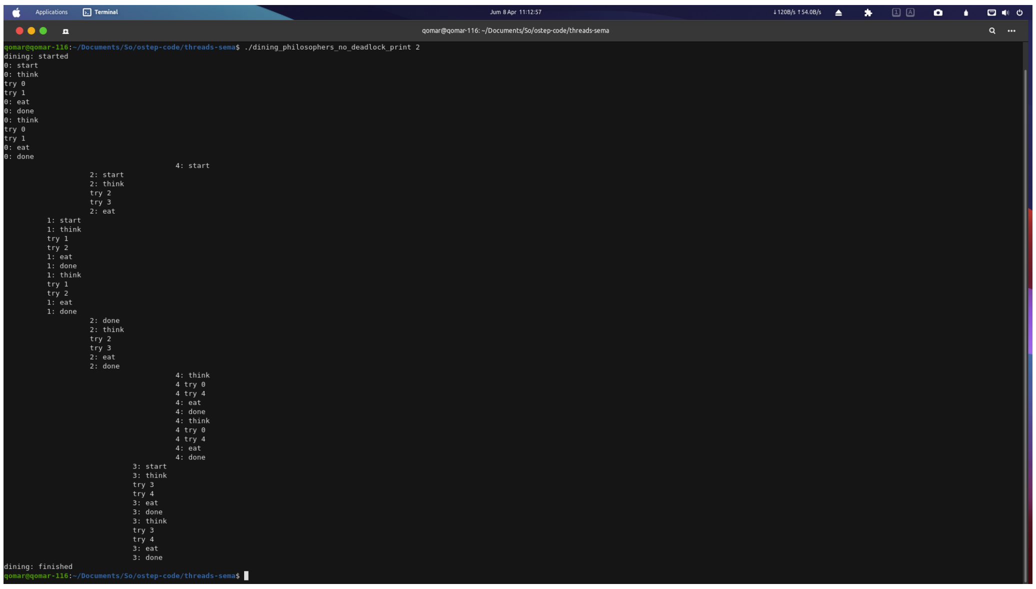
6.1.2 penjelasan

6.2 *Dining Philosophers no Deadlock*

6.2.1 source code

```c
void space(int s) {
  Sem_wait(&print_lock);
  int i;
  for (i = 0; i < s * 10; i++)
  printf(" ");
}

void space_end() {
  Sem_post(&print_lock);
}

int left(int p)  {
  return p;
}

int right(int p) {
  return (p + 1) % 5;
}

void get_forks(int p) {
  if (p == 4) {
  space(p); printf("4 try %d\n", right(p)); space_end();
  Sem_wait(&forks[right(p)]);
  space(p); printf("4 try %d\n", left(p)); space_end();
  Sem_wait(&forks[left(p)]);
  } else {
  space(p); printf("try %d\n", left(p)); space_end();
  Sem_wait(&forks[left(p)]);
  space(p); printf("try %d\n", right(p)); space_end();
  Sem_wait(&forks[right(p)]);
  }
}

void put_forks(int p) {
  Sem_post(&forks[left(p)]);
  Sem_post(&forks[right(p)]);
}

void think() {
  return;
}

void eat() {
  return;
}

void *philosopher(void *arg) {
  arg_t *args = (arg_t *) arg;

  space(args->thread_id); printf("%d: start\n", args->thread_id); space_end();

  int i;
  for (i = 0; i < args->num_loops; i++) {
  space(args->thread_id); printf("%d: think\n", args->thread_id); space_end();
  think();
  get_forks(args->thread_id);
  space(args->thread_id); printf("%d: eat\n", args->thread_id); space_end();
  eat();
  put_forks(args->thread_id);
  space(args->thread_id); printf("%d: done\n", args->thread_id); space_end();
  }
  return NULL;
}

int main(int argc, char *argv[]) {
  if (argc != 2) {
  fprintf(stderr, "usage: dining_philosophers <num_loops>\n");
  exit(1);
  }
  printf("dining: started\n");

  int i;
  for (i = 0; i < 5; i++)
  Sem_init(&forks[i], 1);
  Sem_init(&print_lock, 1);

  pthread_t p[5];
  arg_t a[5];
  for (i = 0; i < 5; i++) {
  a[i].num_loops = atoi(argv[1]);
  a[i].thread_id = i;
  Pthread_create(&p[i], NULL, philosopher, &a[i]);
  }

  for (i = 0; i < 5; i++)
  Pthread_join(p[i], NULL);

  printf("dining: finished\n");
  return 0;
}
```

output



```
qomar@qomar-116:~/Documents/So/ostep-code/threads-sema$ ./dining_philosophers_no_deadlock_print 2
dining: started
0: start
0: think
try 0
try 1
0: eat
0: done
0: think
try 0
try 1
0: eat
0: done
                                        4: start
                    2: start
                    2: think
                    try 2
                    try 3
                    2: eat
          1: start
          1: think
          try 1
          try 2
          1: eat
          1: done
          1: think
          try 1
          try 2
          1: eat
          1: done
                    2: done
                    2: think
                    try 2
                    try 3
                    2: eat
                    2: done
                                        4: think
                                        4 try 0
                                        4 try 4
                                        4: eat
                                        4: done
                                        4: think
                                        4 try 0
                                        4 try 4
                                        4: eat
                                        4: done
                              3: start
                              3: think
                              try 3
                              try 4
                              3: eat
                              3: done
                              3: think
                              try 3
                              try 4
                              3: eat
                              3: done
dining: finished
qomar@qomar-116:~/Documents/So/ostep-code/threads-sema$
```

Gambar 6: Dining Philosophers no Deadlock

6.2.2 penjelasan

7 kesimpulan

8 link laporan dan berkas

- Link github : HandsOn_2_120140116_SO