Program Studi Teknik Informatika
Institut Teknologi Sumatera

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

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

1 Tujuan Hands On 2

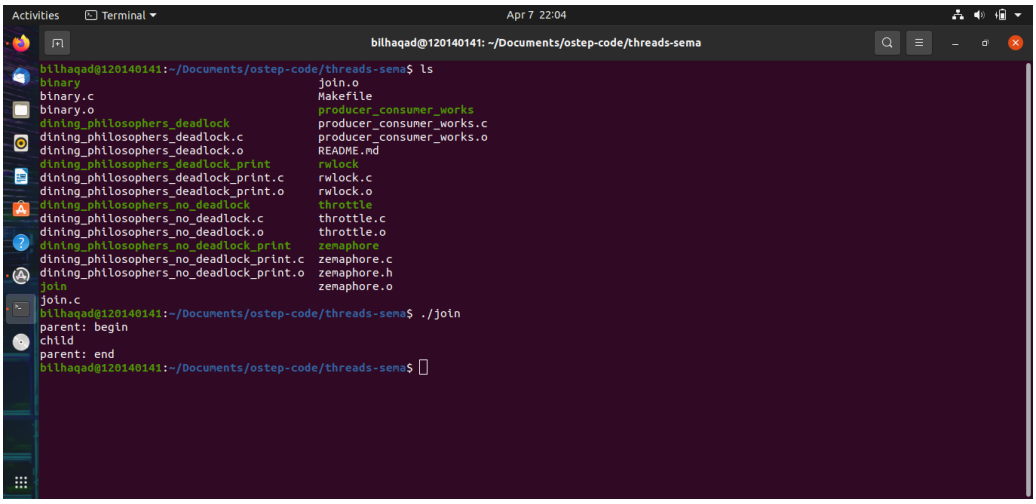
Tujuan adanya Hands On kedua adalah untuk memahami bagaimana sistem bersinkronisasi dan permasalahan yang ada, serta juga memahami solusinya saat menjalankan critical section. Adapun beberapa implementasi yang diharuskan untun dipahami pada Hands On kedua ini antara lain : *join* menggunakan semaphores,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

2 Fork/Join

2.1 Source Code

```c
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <unistd.h>

#include "common.h"
#include "common_threads.h"

#ifdef linux
#include <semaphore.h>
#elif __APPLE__
#include "zemaphore.h"
#endif

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;
}

```

2.2 Output



Gambar 1: *Fork/Join*

2.3 Penjelasan *Fork/Join*

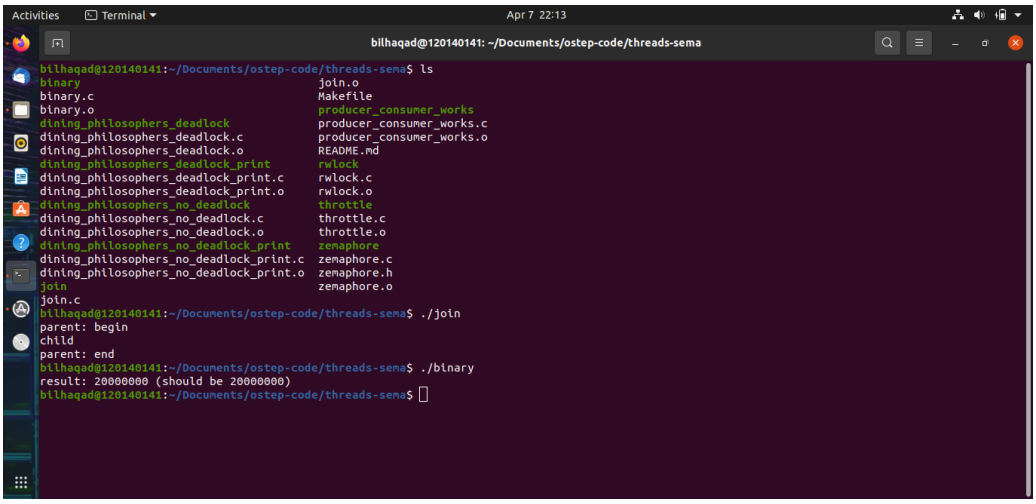
3 Binary Semaphores

3.1 Source Code

```c
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <unistd.h>

#include "common.h"
#include "common_threads.h"

#ifdef linux
#include <semaphore.h>
#elif __APPLE__
#include "zemaphore.h"
#endif

sem_t mutex;
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);
```

```
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

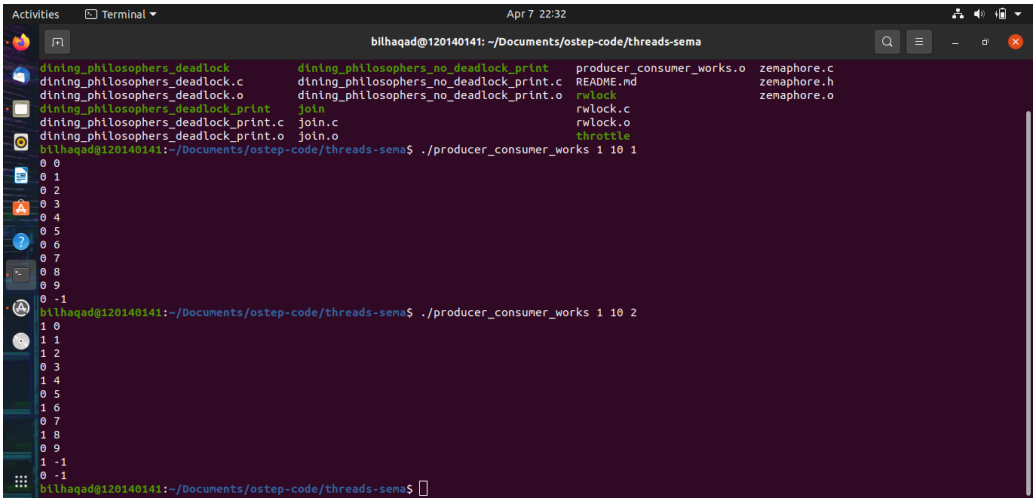
4.1 Source Code

```
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 "zemaphore.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



Gambar 3: *Producer/Consumer*

4.3 Penjelasan *Producer/Consumer*

5 Reader/Writer Locks

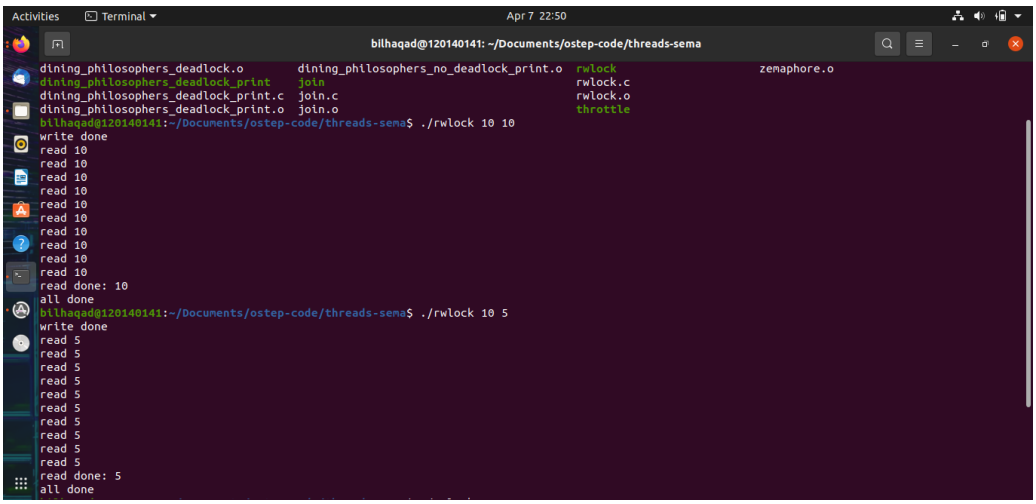
5.1 Source Code

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

```c
#include <unistd.h>

#include "common.h"
#include "common_threads.h"

#ifdef linux
#include <semaphore.h>
#elif __APPLE__
#include "zemaphore.h"
#endif

typedef struct _rwlock_t {
  sem_t writelock;
  sem_t lock;
  int readers;
} rwlock_t;

void rwlock_init(rwlock_t *lock) {
  lock->readers = 0;
  Sem_init(&lock->lock, 1);
  Sem_init(&lock->writelock, 1);
}

void rwlock_acquire_readlock(rwlock_t *lock) {
  Sem_wait(&lock->lock);
  lock->readers++;
  if (lock->readers == 1)
  Sem_wait(&lock->writelock);
  Sem_post(&lock->lock);
}

void rwlock_release_readlock(rwlock_t *lock) {
  Sem_wait(&lock->lock);
  lock->readers--;
  if (lock->readers == 0)
  Sem_post(&lock->writelock);
  Sem_post(&lock->lock);
}

void rwlock_acquire_writelock(rwlock_t *lock) {
  Sem_wait(&lock->writelock);
}

void rwlock_release_writelock(rwlock_t *lock) {
  Sem_post(&lock->writelock);
}

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);
  }
```

```
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



Gambar 4: *Reader/Writer Locks*

5.3 Penjelasan *Reader/Writer Locks*

6 Dining Philosophers

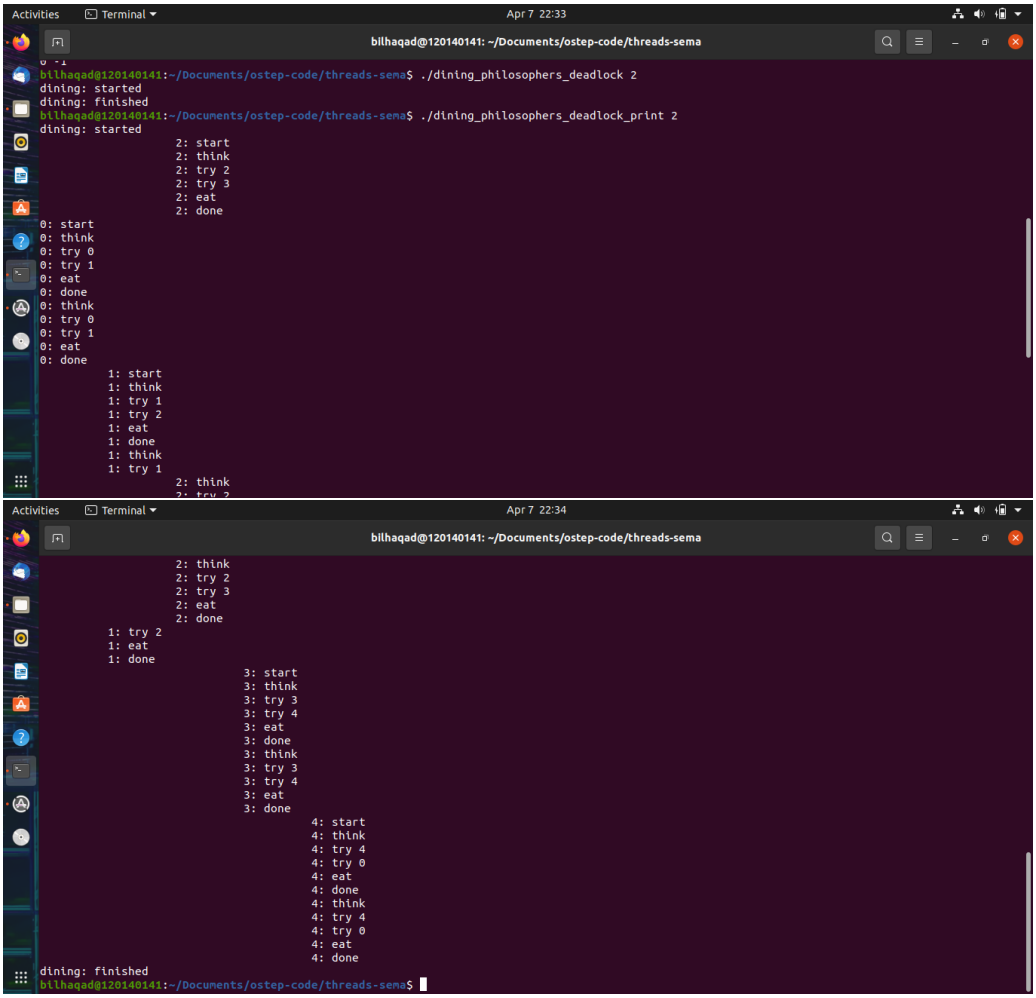
6.1 Deadlock

6.1.1 Source Code

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>

#include "common.h"
#include "common_threads.h"

#ifdef linux
#include <semaphore.h>
#elif __APPLE__
#include "zemaphore.h"
#endif

typedef struct {
  int num_loops;
  int thread_id;
} arg_t;

sem_t forks[5];
sem_t print_lock;

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;
}
```

6.1.2 Output



6.1.3 Penjelasan *Dining Philosophers Deadlock*

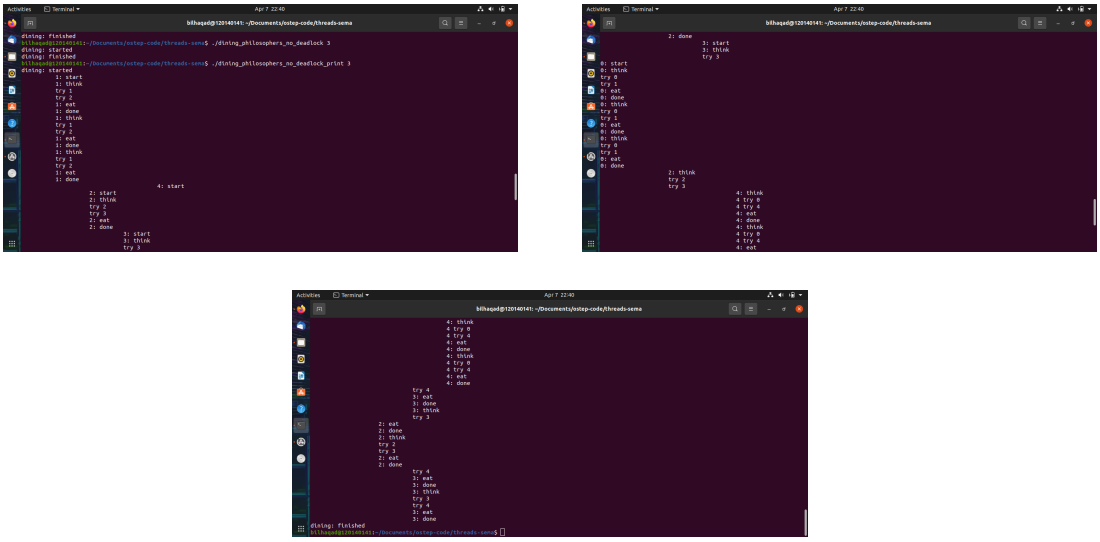
6.2 No Deadlock

6.2.1 Source Code

```c
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>

#include "common.h"
#include "common_threads.h"

#ifdef linux
#include <semaphore.h>
#elif __APPLE__
#include "zemaphore.h"
#endif

typedef struct {
  int num_loops;
  int thread_id;
} arg_t;

sem_t forks[5];
sem_t print_lock;

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)]);
}
```

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

6.2.2 Output



Gambar 5: Dining Philosophers No Deadlock

6.2.3 Penjelasan *Dining Philosophers No 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