LABORATORIUM JARINGAN KOMPUTER
FAKULTAS ILMU KOMPUTER
UNIVERSITAS BRAWIJAYA

PRAKTIKUM SISTEM OPERASI

SEMESTER : GENAP **TAHUN : 2015/2016**

BAB V

JUDUL BAB : DEADLOCK

DISUSUN OLEH : MOH ARIF ANDRIAN

NIM : 156150600111002

ASISTEN : SISKA PERMATASARI
ZAENAL KURNIAWAN

KOORDINATOR ASISTEN : DANY RAHMANA

LAPORAN PRAKTIKUM SISTEM OPERASI
FAKULTAS ILMU KOMPUTER
UNIVERSITAS BRAWIJAYA

Nama : Moh. Arif Andrian
NIM : 156150600111002
Laporan : BAB V
Asisten : Siska Permatasari
Zaenal Kurniawan

BAB V
DEADLOCK

5.1 DASAR TEORI

5.1.1 Konsep Dasar

Permasalahan deadlock terjadi karena sekumpulan proses-proses yang di-blok dimana setiap proses membawa sebuah sumber daya dan menunggu mendapatkan sumber daya yang dibawa oleh proses lain. Misalnya sistem mempunyai 2 tape drive dan terdapat dua proses P1 dan P2 yang masing masing membawa satu tape drive dan masing-masing memerlukan tape drive yang dibawa proses lain sehingga terjadi keadaan saling menunggu resource dan sistem di-blok. Contoh lain, misalnya terdapat semaphore A dan B yang diinisialisasi 1 dan terdapat dua proses P0 dan P1 masing-masing membawa semaphore A dan B. Kemudian P0 dan P1 meminta semaphore B dan A dengan menjalankan operasi wait. Hal ini mengakibatkan proses di-blok dan terjadi deadlock.

5.1.2 Dining Philosophers Problem

Dining Philosophers Problem merupakan salah satu masalah klasik dalam sinkronisasi. Dining Philosohers Problem dapat diilustrasikan sebagai berikut, terdapat lima orang filsuf yang sedang duduk mengelilingi sebuah meja. Terdapat lima mangkuk mie di depan masing-masing filsuf dan satu sumpit di antara masing-masing filsuf. Para filsuf menghabiskan waktu dengan berpikir (ketika kenyang) dan makan (ketika lapar). Ketika lapar, filsuf akan mengambil dua buah sumpit (di tangan kiri dan tangan kanan) dan makan. Namun adakalanya, hanya diambil satu sumpit saja. Jika ada filsuf yang mengambil dua buah sumpit, maka dua filsuf di samping filsuf yang sedang makan harus menunggu sampai sumpit ditaruh kembali. Hal ini dapat diimplementasikan dengan wait dan signal.

Meskipun solusi ini menjamin bahwa tidak ada 2 tetangga yang makan bersama-sama, namun masih mungkin terjadi deadlock, yaitu jika tiap-tiap filsuf lapar dan mengambil sumpit kiri, maka semua nilai sumpit=0, dan kemudian tiap-tiap filsuf akan mengambil sumpit kanan, maka akan terjadi deadlock.

5.1.3 Cara Menghindari Deadlock

Ada beberapa cara untuk menghindari deadlock, antara lain:

a. Mengijinkan paling banyak 4 orang filsuf yang duduk bersama-sama pada satu meja.
b. Mengijinkan seorang filsuf mengambil sumpit hanya jika kedua sumpit itu ada.
c. Menggunakan suatu solusi asimetrik, yaitu filsuf pada nomor ganjil mengambil sumpit kiri dulu baru sumpit kanan. Sedangkan filsuf yang duduk di kursi genap mengambil sumpit kanan dulu baru sumpit kiri

5.2 MATERI PRAKTIKUM

Percobaan 5.2

Lakukan percobaan 5.2 berikut, kemudian amati apa yang akan terjadi!

```
andrian@1561506001110002:~$ sudo su
[sudo] password for andrian:
root@1561506001110002:/home/andrian# cd Deadlock/
```

1. Buatlah sebuah file bernama din_philo.c

```
root@1561506001110002:/home/andrian/Deadlock# nano di_philo.c
```

2. Tuliskan kode seperti pada program 5.2 kemudian compile dengan syntax gcc –pthread –o din_philo din_philo.c

```
root@1561506001110002:/home/andrian/Deadlock# gcc -pthread -o din_philo din_philo.c
din_philo.c: In function ‘main’:
din_philo.c:40:57: warning: cast to pointer from integer of different size [-Wint-to-
pointer-cast]
          pthread_create (&philo[i], NULL, philosopher, (void *)i);
                                                        ^
din_philo.c: In function ‘philosopher’:
din_philo.c:52:12: warning: cast from pointer to integer of different size [-Wpointer
-to-int-cast]
       id = (int)num;
            ^
```

3. Jalankan dengan perintah ./din_philo

```
root@1561506001110002:/home/andrian/Deadlock# ./din_philo
```

4. Sehingga nanti muncul seperti gambar dibawah ini

```
yudha@google:~/Documents/deadlock/3.1$ ./din_philo
Philosopher 1 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 4: got left chopstick 0
Philosopher 4: eating.
Philosopher 0 is done thinking and now ready to eat.
Philosopher 3 is done thinking and now ready to eat.
Philosopher 3: got right  chopstick 3
Philosopher 1: got right  chopstick 1
Philosopher 1: got left chopstick 2
Philosopher 1: eating.
Philosopher 2 is done thinking and now ready to eat.
Philosopher 1: got right  chopstick 1
Philosopher 2: got right  chopstick 2
Philosopher 4: got right  chopstick 4
Philosopher 0: got right  chopstick 0
```

```
root@156150600111002:/home/andrian/Deadlock# ./din_philo
Philosopher 1 is done thinking and now ready to eat.
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 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 4: got left chopstick 0
Philosopher 4: eating.
Philosopher 0 is done thinking and now ready to eat.
Philosopher 1: got right  chopstick 1
Philosopher 2: got right  chopstick 2
Philosopher 2: got left chopstick 3
Philosopher 2: eating.
Philosopher 4: got right  chopstick 4
Philosopher 0: got right  chopstick 0
Philosopher 2: got right  chopstick 2
Philosopher 3: got right  chopstick 3
```

Program 5.2

```
/* din_philo.c */
  #include <pthread.h>
  #include <stdio.h>
  #include <unistd.h>
  #include <stdlib.h>
  #include <errno.h>
  #include <assert.h>

  #define PHILOS 5
  #define DELAY 5000
  #define FOOD 50
```

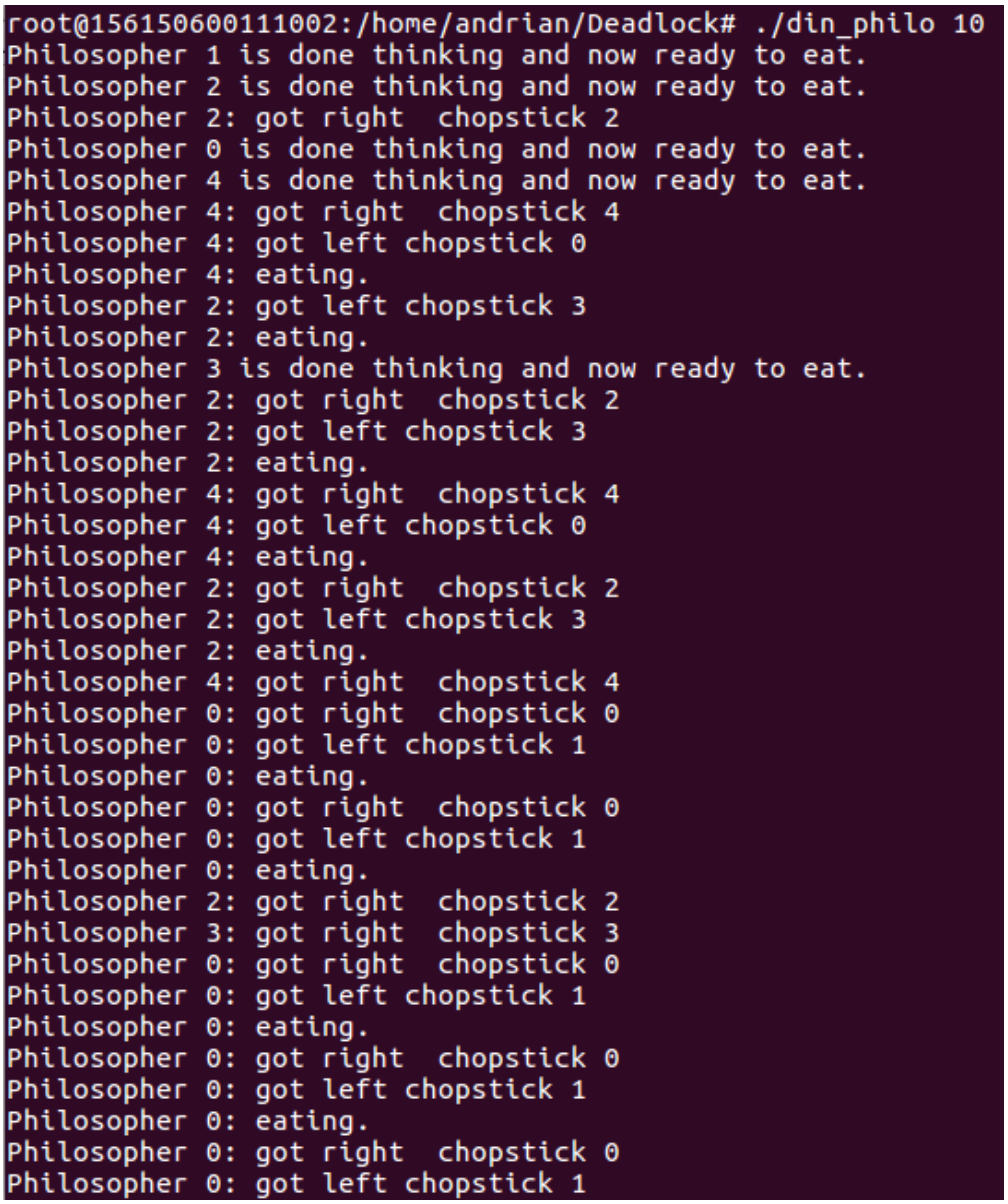
```
  void *philosopher (void *id);
  void grab_chopstick (int,
                       int,
                       char *);
  void down_chopsticks (int,
                        int);
  int food_on_table ();

  pthread_mutex_t chopstick[PHILOS];
  pthread_t philo[PHILOS];
  pthread_mutex_t food_lock;
  int sleep_seconds = 0;


  int
  main (int argn,
        char **argv)
  {
      int i;

      if (argn == 2)
          sleep_seconds = atoi (argv[1]);

      pthread_mutex_init (&food_lock, NULL);
      for (i = 0; i < PHILOS; i++)
          pthread_mutex_init (&chopstick[i], NULL);
      for (i = 0; i < PHILOS; i++)
          pthread_create (&philo[i], NULL, philosopher,
(void *)i);
      for (i = 0; i < PHILOS; i++)
          pthread_join (philo[i], NULL);
      return 0;
  }

  void *
  philosopher (void *num)
  {
      int id;
      int i, left_chopstick, right_chopstick, f;

      id = (int)num;
      printf ("Philosopher %d is done thinking and now
ready to eat.\n", id);
      right_chopstick = id;
      left_chopstick = id + 1;
```

```
      /* Wrap around the chopsticks. */
      if (left_chopstick == PHILOS)
          left_chopstick = 0;

      while (f = food_on_table ()) {

          /* Thanks to philosophers #1 who would like to
take a nap
           * before picking up the chopsticks, the other
philosophers
           * may be able to eat their dishes and not
deadlock.
           */
          if (id == 1)
              sleep (sleep_seconds);

          grab_chopstick (id, right_chopstick, "right ");
          grab_chopstick (id, left_chopstick, "left");

          printf ("Philosopher %d: eating.\n", id);
          usleep (DELAY * (FOOD - f + 1));
          down_chopsticks               (left_chopstick,
right_chopstick);
      }

      printf ("Philosopher %d is done eating.\n", id);
      return (NULL);
  }

  int
  food_on_table ()
  {
      static int food = FOOD;
      int myfood;

      pthread_mutex_lock (&food_lock);
      if (food > 0) {
          food--;
      }
      myfood = food;
      pthread_mutex_unlock (&food_lock);
      return myfood;
  }

  void
```

```
  grab_chopstick (int phil,
                  int c,
                  char *hand)
  {
      pthread_mutex_lock (&chopstick[c]);
      printf ("Philosopher %d: got %s chopstick %d\n",
phil, hand, c);
  }

  void
  down_chopsticks (int c1,
                   int c2)
  {
      pthread_mutex_unlock (&chopstick[c1]);
      pthread_mutex_unlock (&chopstick[c2]);
  }
```

5. Lakukan penambahan parameter pada kode diatas sehingga menampilkan output seperti berikut!

```
yudha@google:~/Documents/deadlock/3.1$ ./din_philo 30
Philosopher 1 is done thinking and now ready to eat.
Philosopher 3 is done thinking and now ready to eat.
Philosopher 3: got right  chopstick 3
Philosopher 3: got left chopstick 4
Philosopher 3: eating.
Philosopher 2 is done thinking and now ready to eat.
Philosopher 2: got right  chopstick 2
Philosopher 0 is done thinking and now ready to eat.
Philosopher 0: got right  chopstick 0
Philosopher 0: got left chopstick 1
Philosopher 4: got left chopstick 0
Philosopher 4: eating.
Philosopher 4 is done eating.
Philosopher 3: got left chopstick 4
Philosopher 3: eating.
Philosopher 3 is done eating.
Philosopher 2: got left chopstick 3
Philosopher 2: eating.
Philosopher 2 is done eating.
Philosopher 1: got right  chopstick 1
Philosopher 1: got left chopstick 2
Philosopher 1: eating.
Philosopher 1 is done eating.
yudha@google:~/Documents/deadlock/3.1$
```


root@156150600111002:/home/andrian/Deadlock# ./din_philo 10
Philosopher 1 is done thinking and now ready to eat.
Philosopher 2 is done thinking and now ready to eat.
Philosopher 2: got right chopstick 2
Philosopher 0 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 4: got left chopstick 0
Philosopher 4: eating.
Philosopher 2: got left chopstick 3
Philosopher 2: eating.
Philosopher 3 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: got right chopstick 4
Philosopher 4: got left chopstick 0
Philosopher 4: eating.
Philosopher 2: got right chopstick 2
Philosopher 2: got left chopstick 3
Philosopher 2: eating.
Philosopher 4: got right chopstick 4
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 2: got right chopstick 2
Philosopher 3: got right chopstick 3
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: 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.
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 is done eating.
Philosopher 4: got left chopstick 0
Philosopher 4: eating.
Philosopher 4 is done eating.
Philosopher 3: got left chopstick 4
Philosopher 3: eating.
Philosopher 3 is done eating.
Philosopher 2: got left chopstick 3
Philosopher 2: eating.
Philosopher 2 is done eating.
Philosopher 1: got right  chopstick 1
Philosopher 1: got left chopstick 2
Philosopher 1: eating.
Philosopher 1 is done eating.
```

LAPORAN PRAKTIKUM SISTEM OPERASI
FAKULTAS ILMU KOMPUTER
UNIVERSITAS BRAWIJAYA

Nama : Moh. Arif Andrian
NIM : 156150600111002
Tugas : BAB V
Asisten : Siska Permatasari
Zaenal Kurniawan

TUGAS PRAKTIKUM

Lakukan sama seperti percobaan 5.1, kemudian amati perbedaan proses output dari kedua percobaan(antara Percobaan 5.1 dan Percobaan 5.2), lakukan analisis dan jelaskan letak perbedaannya.

```
#include <pthread.h>
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <errno.h>
#include <assert.h>

#define PHILOS 5
#define DELAY 5000
#define FOOD 50

void *philosopher (void *id);
void grab_chopstick (int,
                     int,
                     char *);
void down_chopsticks (int,
                      int);
int food_on_table ();
int get_token ();
void return_token ();

pthread_mutex_t chopstick[PHILOS];
pthread_t philo[PHILOS];
pthread_mutex_t food_lock;
pthread_mutex_t num_can_eat_lock;
int sleep_seconds = 0;
uint32_t num_can_eat = PHILOS - 1;


int
main (int argn,
      char **argv)
{
    int i;
```

```
    pthread_mutex_init (&food_lock, NULL);
    pthread_mutex_init (&num_can_eat_lock, NULL);
    for (i = 0; i < PHILOS; i++)
        pthread_mutex_init (&chopstick[i], NULL);
    for (i = 0; i < PHILOS; i++)
        pthread_create (&philo[i], NULL, philosopher, (void
*)i);
    for (i = 0; i < PHILOS; i++)
        pthread_join (philo[i], NULL);
    return 0;
}

void *
philosopher (void *num)
{
    int id;
    int i, left_chopstick, right_chopstick, f;

    id = (int)num;
    printf ("Philosopher %d is done thinking and now ready
to eat.\n", id);
    right_chopstick = id;
    left_chopstick = id + 1;

    /* Wrap around the chopsticks. */
    if (left_chopstick == PHILOS)
        left_chopstick = 0;

    while (f = food_on_table ()) {
        get_token ();

        grab_chopstick (id, right_chopstick, "right ");
        grab_chopstick (id, left_chopstick, "left");

        printf ("Philosopher %d: eating.\n", id);
        usleep (DELAY * (FOOD - f + 1));
        down_chopsticks (left_chopstick, right_chopstick);

        return_token ();
    }

    printf ("Philosopher %d is done eating.\n", id);
    return (NULL);
}
```

```
int
food_on_table ()
{
    static int food = FOOD;
    int myfood;

    pthread_mutex_lock (&food_lock);
    if (food > 0) {
        food--;
    }
    myfood = food;
    pthread_mutex_unlock (&food_lock);
    return myfood;
}

void
grab_chopstick (int phil,
                int c,
                char *hand)
{
    pthread_mutex_lock (&chopstick[c]);
    printf ("Philosopher %d: got %s chopstick %d\n", phil,
hand, c);
}

void
down_chopsticks (int c1,
                 int c2)
{
    pthread_mutex_unlock (&chopstick[c1]);
    pthread_mutex_unlock (&chopstick[c2]);
}


int
get_token ()
{
    int successful = 0;

    while (!successful) {
        pthread_mutex_lock (&num_can_eat_lock);
        if (num_can_eat > 0) {
            num_can_eat--;
            successful = 1;
        }
        else {
```

```
                successful = 0;
            }
            pthread_mutex_unlock (&num_can_eat_lock);
        }
    }

    void
    return_token ()
    {
        pthread_mutex_lock (&num_can_eat_lock);
        num_can_eat++;
        pthread_mutex_unlock (&num_can_eat_lock);
    }
```

1. Membuat file bernama din_philo1.c

```
root@156150600111002:/home/andrian/Deadlock# nano din_philo1.c
```

2. Melakukan compiling terhadap file din_philo1.c

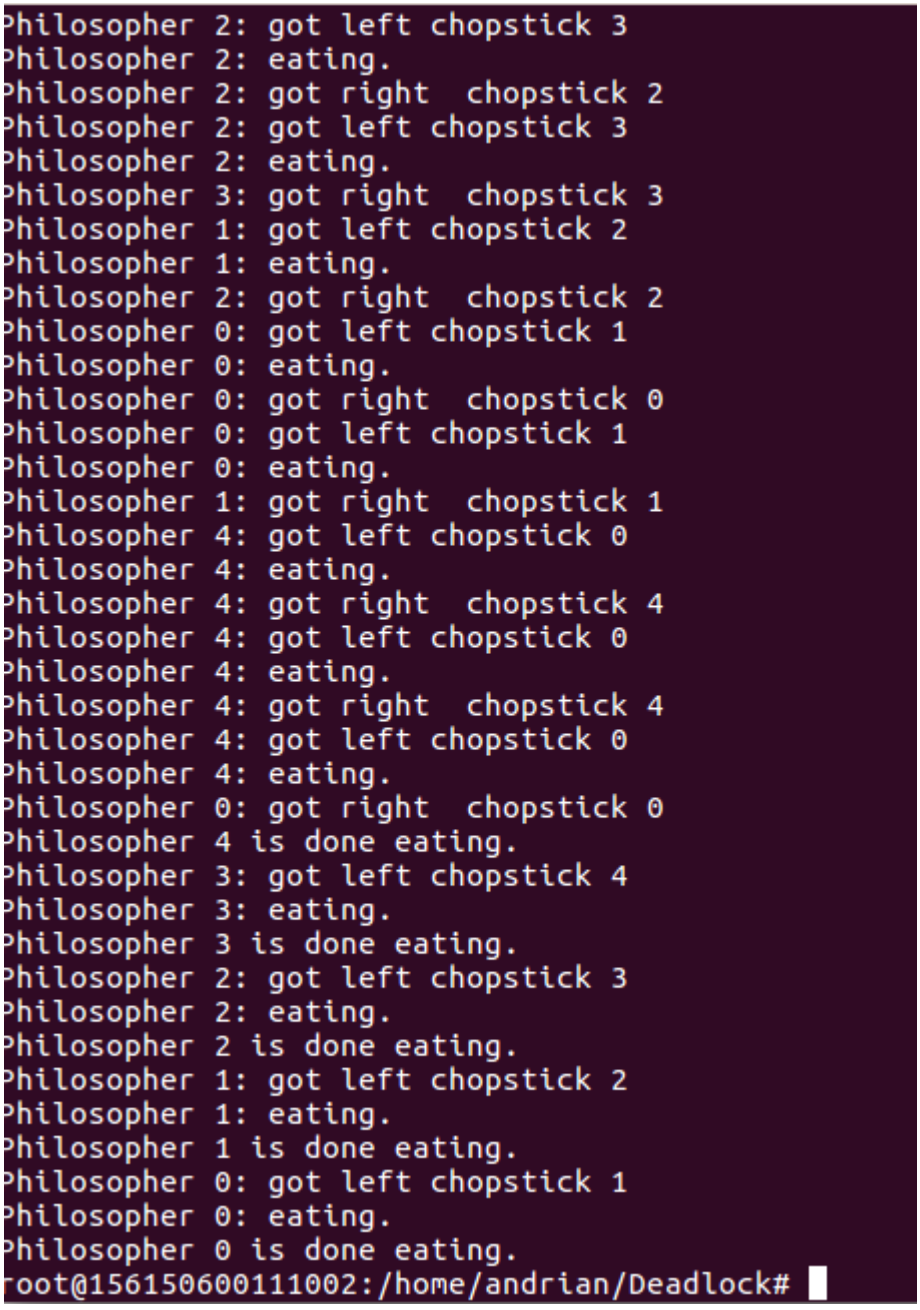
```
root@156150600111002:/home/andrian/Deadlock# gcc -pthread -o din_philo1 din_philo1.c

din_philo1.c: In function ‘main’:
din_philo1.c:41:55: warning: cast to pointer from integer of different size [-Wint-to
-pointer-cast]
         pthread_create (&philo[i], NULL, philosopher, (void *)i);
                                                       ^
din_philo1.c: In function ‘philosopher’:
din_philo1.c:53:10: warning: cast from pointer to integer of different size [-Wpointe
r-to-int-cast]
     id = (int)num;
          ^
```

3. Menjalankan file ./din_philo1

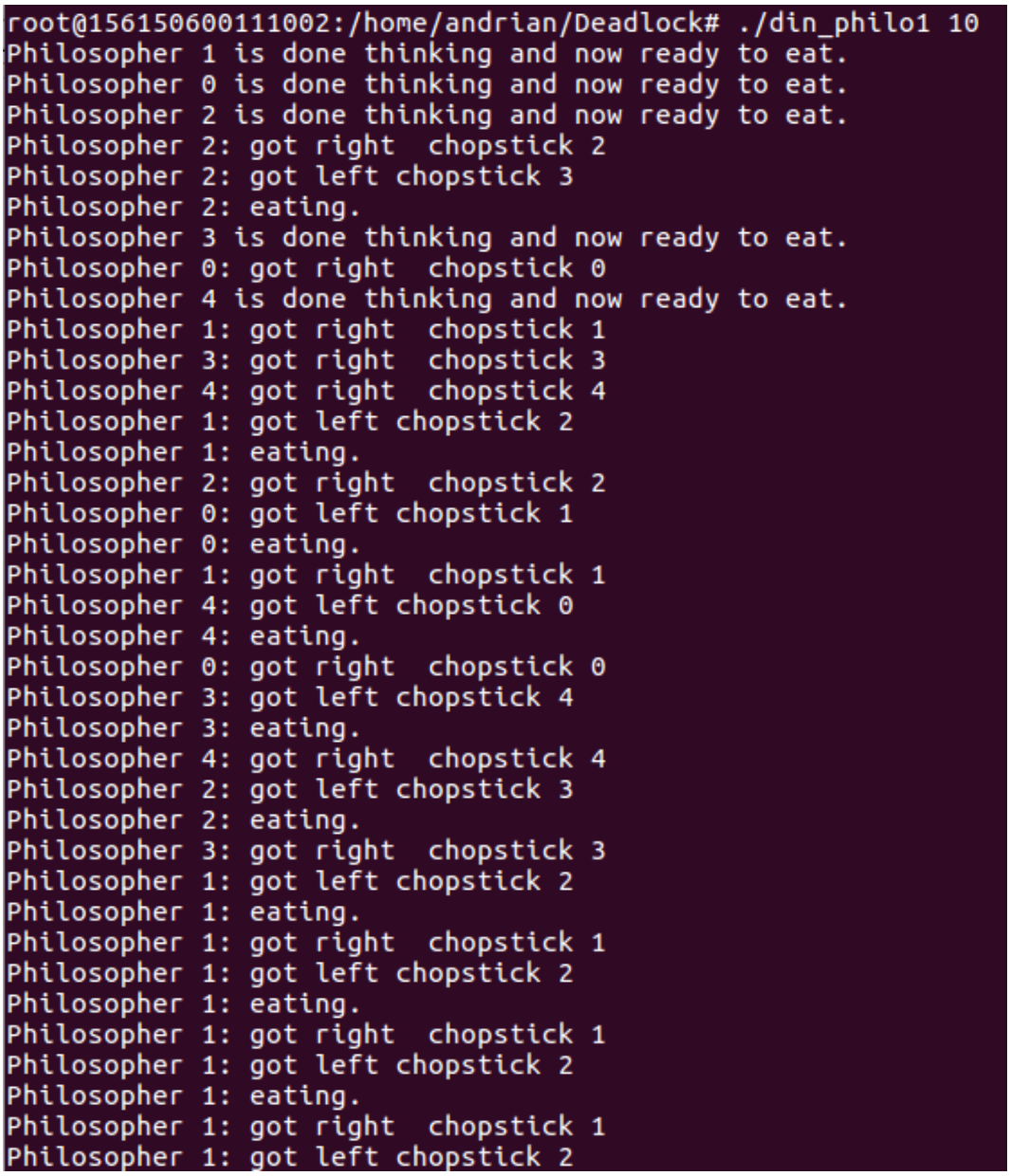
```
root@156150600111002:/home/andrian/Deadlock# ./din_philo1
Philosopher 0 is done thinking and now ready to eat.
Philosopher 0: got right  chopstick 0
Philosopher 0: got left chopstick 1
Philosopher 0: eating.
Philosopher 1 is done thinking and now ready to eat.
Philosopher 2 is done thinking and now ready to eat.
Philosopher 2: got right  chopstick 2
Philosopher 2: got left chopstick 3
Philosopher 2: eating.
Philosopher 3 is done thinking and now ready to eat.
Philosopher 4 is done thinking and now ready to eat.
Philosopher 0: got right  chopstick 0
Philosopher 1: got right  chopstick 1
Philosopher 4: got right  chopstick 4
Philosopher 3: got right  chopstick 3
Philosopher 1: got left chopstick 2
Philosopher 1: eating.
Philosopher 2: got right  chopstick 2
Philosopher 0: got left chopstick 1
Philosopher 0: eating.
Philosopher 1: got right  chopstick 1
Philosopher 4: got left chopstick 0
Philosopher 4: eating.
Philosopher 3: got left chopstick 4
Philosopher 3: eating.
Philosopher 0: got right  chopstick 0
Philosopher 3: got right  chopstick 3
Philosopher 3: got left chopstick 4
Philosopher 3: eating.
Philosopher 3: got right  chopstick 3
Philosopher 3: got left chopstick 4
Philosopher 3: eating.
Philosopher 3: got right  chopstick 3
Philosopher 3: got left chopstick 4
Philosopher 3: eating.
Philosopher 4: got right  chopstick 4
Philosopher 2: got left chopstick 3
```

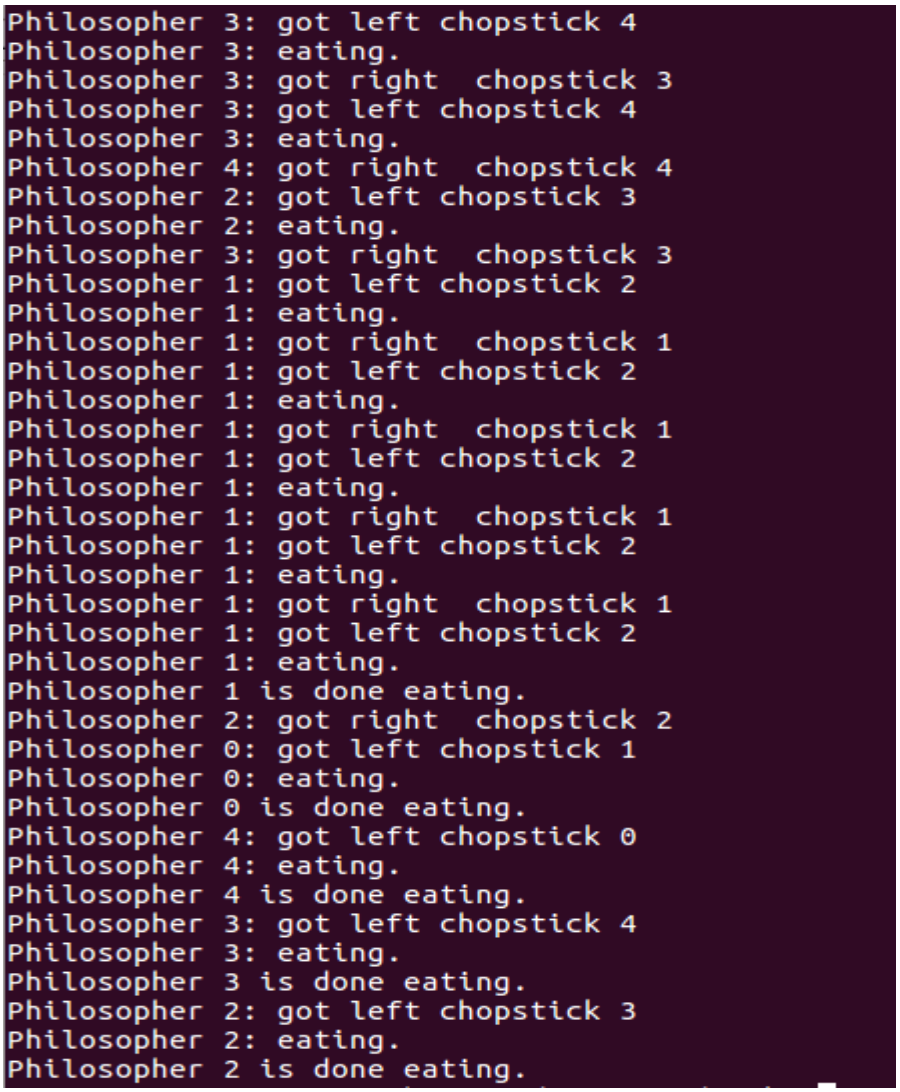
```
Philosopher 2: eating.
Philosopher 2: got right  chopstick 2
Philosopher 2: got left chopstick 3
Philosopher 2: eating.
Philosopher 3: got right  chopstick 3
Philosopher 1: got left chopstick 2
Philosopher 1: eating.
Philosopher 1: got right  chopstick 1
Philosopher 1: got left chopstick 2
Philosopher 1: eating.
Philosopher 1: got right  chopstick 1
Philosopher 1: got left chopstick 2
Philosopher 1: eating.
Philosopher 1: got right  chopstick 1
Philosopher 1: got left chopstick 2
Philosopher 1: eating.
Philosopher 1: got right  chopstick 1
Philosopher 1: got left chopstick 2
Philosopher 1: eating.
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 1: got right  chopstick 1
Philosopher 4: got left chopstick 0
Philosopher 4: eating.
Philosopher 0: got right  chopstick 0
Philosopher 3: got left chopstick 4
Philosopher 3: eating.
Philosopher 4: got right  chopstick 4
Philosopher 2: got left chopstick 3
Philosopher 2: eating.
Philosopher 3: got right  chopstick 3
Philosopher 1: got left chopstick 2
Philosopher 1: eating.
Philosopher 1: got right  chopstick 1
```

```
Philosopher 1: eating.
Philosopher 1: got right  chopstick 1
Philosopher 1: got left chopstick 2
Philosopher 1: eating.
Philosopher 2: got right  chopstick 2
Philosopher 0: got left chopstick 1
Philosopher 0: eating.
Philosopher 1: got right  chopstick 1
Philosopher 4: got left chopstick 0
Philosopher 4: eating.
Philosopher 4: got right  chopstick 4
Philosopher 4: got left chopstick 0
Philosopher 4: eating.
Philosopher 0: got right  chopstick 0
Philosopher 3: got left chopstick 4
Philosopher 3: eating.
Philosopher 3: got right  chopstick 3
Philosopher 3: got left chopstick 4
Philosopher 3: eating.
Philosopher 3: got right  chopstick 3
Philosopher 3: got left chopstick 4
Philosopher 3: eating.
Philosopher 3: got right  chopstick 3
Philosopher 3: got left chopstick 4
Philosopher 3: eating.
Philosopher 4: got right  chopstick 4
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 3: got right  chopstick 3
Philosopher 1: got left chopstick 2
Philosopher 1: eating.
Philosopher 2: got right  chopstick 2
Philosopher 0: got left chopstick 1
Philosopher 0: eating.
Philosopher 0: got right  chopstick 0
```



4. Penambahan parameter pada kode


root@156150600111002:/home/andrian/Deadlock# ./din_philo1 10
Philosopher 1 is done thinking and now ready to eat.
Philosopher 0 is done thinking and now ready to eat.
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 3 is done thinking and now ready to eat.
Philosopher 0: got right chopstick 0
Philosopher 4 is done thinking and now ready to eat.
Philosopher 1: got right chopstick 1
Philosopher 3: got right chopstick 3
Philosopher 4: got right chopstick 4
Philosopher 1: got left chopstick 2
Philosopher 1: eating.
Philosopher 2: got right chopstick 2
Philosopher 0: got left chopstick 1
Philosopher 0: eating.
Philosopher 1: got right chopstick 1
Philosopher 4: got left chopstick 0
Philosopher 4: eating.
Philosopher 0: got right chopstick 0
Philosopher 3: got left chopstick 4
Philosopher 3: eating.
Philosopher 4: got right chopstick 4
Philosopher 2: got left chopstick 3
Philosopher 2: eating.
Philosopher 3: got right chopstick 3
Philosopher 1: got left chopstick 2
Philosopher 1: eating.
Philosopher 1: got right chopstick 1
Philosopher 1: got left chopstick 2
Philosopher 1: eating.
Philosopher 1: got right chopstick 1
Philosopher 1: got left chopstick 2
Philosopher 1: eating.
Philosopher 1: got right chopstick 1
Philosopher 1: got left chopstick 2



5. Analisis perbedaan.

Bin_philo (5.2)	**Bin_philo1(5.3)**
Tidak akan pernah berhenti dan terjadi deadlock jika tidak diberikan parameter. Sedangkan pada program 5.2 semua philosopher tidak ada yang selesai makan karena pada program 5.2 masih terjadi deadlock, sehingga philosophernya saling menunggu atau tidak ada yang selesai makan.	langsung dicompile akan berhenti dan semua philosopher akan selesai makan. Hal ini terjadi karena source kedua program tersebut berbeda, pada program 5.3 di tambahan method token. didalam method ini terdapat proses sinkronisasi lock dan key sehingga program 5.3 akan terhindar dari **race condition** yang akan menyebabkan deadlock buktinya saat decompile program ini akan berhenti dan semua philosopher akan selesai makan (is done eating).

LAPORAN PRAKTIKUM SISTEM OPERASI
FAKULTAS ILMU KOMPUTER
UNIVERSITAS BRAWIJAYA

Nama : Moh. Arif Andrian
NIM : 156150600111002
Kesimpulan : BAB V
Asisten : Siska Permatasari
Zaenal Kurniawan

KESIMPULAN

Deadlock terjadi karena sekumpulan proses-proses yang diblok di mana setiap proses membawa sebuah sumber daya dan menunggu mendapatkan sumber daya yang dibawa oleh proses lain.

Penyebab Deadlock :

a. Mutual Exclusion :
   Suatu kondisi di mana setiap sumber daya diberikan tepat pada satu proses pada suatu waktu.
b. Hold and Wait
   Kondisi yang menyatakan proses-proses yang sedang memakai suatu sumber daya dapat meminta sumber daya yang lain.
c. Non-preemptive
   Kondisi di mana suatu sumber daya yang sedang berada pada suatu proses tidak dapat diambil secara paksa dari proses tersebut, sampai proses itu melepaskannya.
d. Circular Wait
   Kondisi yang menyatakan bahwa adanya rantai saling meminta sumber daya yang dimiliki oleh suatu proses oleh proses lainnya.