Program Studi Teknik Informatika
Institut Teknologi Sumatera

Nama: **Muhammad Khadziq (120140233)**
Mata Kuliah: **Sistem Operasi RD (IF2223)**

Tugas Ke: **2**
Tanggal: 11/04/2022

1 Tujuan HandsOn

Tujuan saya melakukan Hands On kedua adalah untuk memahami bagaimana sistem disinkronkan dan masalah apa yang ada, dan untuk memahami solusi saat menjalankan bagian penting. Ada beberapa implementasi yang harus disertakan dalam Hands On kedua ini, antara lain: join menggunakan Semaphores, Binary Semaphores, Produces Consumer, Reader/Writer Locks, dan Dining Philosophers.

2 Join

2.1 Code Snippets

```
    class SynthiaDataset(Dataset):
    #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;
    }

```

Kode 1: Captionnya tulis di sini class

2.2 Gambar

Semaphore adalah struktur data komputer yang sangat berguna untuk menyinkronkan proses dalam program kontrol untuk menjalankan proses. Contohnya adalah ketika utas menunggu daftar untuk daftar kosong atau kosong. Dari kondisi ini, semaphore akan didefinisikan dan diinisialisasi ke nol oleh Sem init. Tujuan dari proses ini adalah semaphore akan dibagikan di antara utas dari proses yang sama. Kemudian, ketika pembuatan utas selesai, ia akan terus memanggil fungsi semaphore child, yang akan memberi sinyal bahwa proses child selesai.

Gambar 1: join

3 Binary

3.1 Code Snippets

```
    class SynthiaDataset(Dataset):
    #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);
        Pthread_create(&c2, NULL, child, NULL);
        Pthread_join(c1, NULL);
        Pthread_join(c2, NULL);
        printf("result: %d (should be 20000000)\n", counter);
```

```
        return 0;
    }

```

Kode 2: Captionnya tulis di sini class

3.2 Gambar

Command sed digunakan untuk menampilkan teks dari sebuah file agar dihapus beberapa bagiannya, bisa dihapus depan, belakang atau depan dan belakangnya. Namun tidak mempengaruhi isi file tersebut.

```
khadziq@khadziq-VirtualBox:~/ostep-code/threads-sema$ ./binary
result: 20000000 (should be 20000000)
```

Gambar 2: Binary

4 Producer Consumer Works

4.1 Code Snippets

Berikut ini adalah contoh dari penggunaan \beginlstlisting untuk menulis potongan kode. Dalam kasus ini saya menggunakan bahasa Python. Jika anda menggunakan C atau yang lainnya, tinggal sesuaikan di bagian parameter dari \beginlstlisting. Anda dapat melihatnya pada code snipptes 6

```

    class SynthiaDataset(Dataset):
    #include <stdio.h>
    #include <unistd.h>
    #include <assert.h>
    #include <pthread.h>
    #include <stdlib.h>

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

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

    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++;
```

```
35        if (fill == max)
36      fill = 0;
37    }
38
39    int do_get() {
40        int tmp = buffer[use];
41        use++;
42       if (use == max)
43      use = 0;
44        return tmp;
45    }
46
47    void *producer(void *arg) {
48        int i;
49        for (i = 0; i < loops; i++) {
50      Sem_wait(&empty);
51      Sem_wait(&mutex);
52      do_fill(i);
53      Sem_post(&mutex);
54      Sem_post(&full);
55        }
56
57        // end case
58        for (i = 0; i < consumers; i++) {
59      Sem_wait(&empty);
60      Sem_wait(&mutex);
61      do_fill(-1);
62      Sem_post(&mutex);
63      Sem_post(&full);
64        }
65
66        return NULL;
67    }
68
69    void *consumer(void *arg) {
70        int tmp = 0;
71        while (tmp != -1) {
72      Sem_wait(&full);
73      Sem_wait(&mutex);
74      tmp = do_get();
75      Sem_post(&mutex);
76      Sem_post(&empty);
77      printf("%lld %d\n", (long long int) arg, tmp);
78        }
79        return NULL;
80    }
81
82    int main(int argc, char *argv[]) {
83        if (argc != 4) {
84      fprintf(stderr, "usage: %s <buffersize> <loops> <consumers>\n", argv[0]);
85      exit(1);
86        }
87        max   = atoi(argv[1]);
88        loops = atoi(argv[2]);
89        consumers = atoi(argv[3]);
90        assert(consumers <= CMAX);
91
92        buffer = (int *) malloc(max * sizeof(int));
93        assert(buffer != NULL);
94        int i;
95        for (i = 0; i < max; i++) {
96      buffer[i] = 0;
```

```
97        }
98
99        Sem_init(&empty, max); // max are empty
100       Sem_init(&full, 0);    // 0 are full
101       Sem_init(&mutex, 1);   // mutex
102
103       pthread_t pid, cid[CMAX];
104       Pthread_create(&pid, NULL, producer, NULL);
105       for (i = 0; i < consumers; i++) {
106     Pthread_create(&cid[i], NULL, consumer, (void *) (long long int) i);
107       }
108       Pthread_join(pid, NULL);
109       for (i = 0; i < consumers; i++) {
110     Pthread_join(cid[i], NULL);
111       }
112       return 0;
113   }
114
```

Kode 3: Captionnya tulis di sini class

4.2 Gambar

Pada program implementasi Producer/Consumer ini bisa kita sebut dengan bounded buffer. Isi program tersebut ialah memanggil, mengurangi, menghalangi konsumer, dan menunggu thread lain agar dapat memanggil Sem post saat terjadi full. Selanjutnya, program akan memulai fungsi procedure yang berguna dalam memanggil Sem wait(empty) dan Sem post(mutex). Pada fungsi procedur juga menjalankan terus sampai empty tadi menjadi max. Producer akan melakukan pengisian dengan fungsi do fill di entry pertama buffer setelah empty berkurang hingga mencapai nilai 0. Berikutnya, producer akan terus berjalan sampai suatu saat nanti memanggil Sem post(mutex) dan Sem post(full) yang mana akan mengganti nilai value full dari nilai -1 menjadi 0. Sehingga, Consumer akan melakukan fungsi looping ulang dan memblok dengan value empty semaphore bernilai kosong.

```
khadziq@khadziq-VirtualBox:~/ostep-code/threads-sema$ ./producer_consumer_works 1 1000 1
0 0
0 1
0 2
0 3
0 4
0 5
0 6
0 7
0 8
0 9
0 10
0 11
0 12
0 13
0 14
0 15
0 16
0 17
```

Gambar 3: Shell nano command

5 Reader or Writer Locks

5.1 Code Snippets

```
    class SynthiaDataset(Dataset):
```

```
#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

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;
```

```
59
60  void *reader(void *arg) {
61      int i;
62      int local = 0;
63      for (i = 0; i < read_loops; i++) {
64    rwlock_acquire_readlock(&mutex);
65    local = counter;
66    rwlock_release_readlock(&mutex);
67    printf("read %d\n", local);
68      }
69      printf("read done: %d\n", local);
70      return NULL;
71  }
72
73  void *writer(void *arg) {
74      int i;
75      for (i = 0; i < write_loops; i++) {
76    rwlock_acquire_writelock(&mutex);
77    counter++;
78    rwlock_release_writelock(&mutex);
79      }
80      printf("write done\n");
81      return NULL;
82  }
83
84  int main(int argc, char *argv[]) {
85      if (argc != 3) {
86    fprintf(stderr, "usage: rwlock readloops writeloops\n");
87    exit(1);
88      }
89      read_loops = atoi(argv[1]);
90      write_loops = atoi(argv[2]);
91
92      rwlock_init(&mutex);
93      pthread_t c1, c2;
94      Pthread_create(&c1, NULL, reader, NULL);
95      Pthread_create(&c2, NULL, writer, NULL);
96      Pthread_join(c1, NULL);
97      Pthread_join(c2, NULL);
98      printf("all done\n");
99      return 0;
100 }
101
```

Kode 4: Captionnya tulis di sini class

5.2 Gambar

Pada program implementasi Writer Locks, dapat dilihat bahwa terdapat classic problem dari flexible locking primitive yang menunjukkan bahwa akses struktur data berbeda perlu dibutuhkannya kunci spesial yang dibuat sebagai pembantu tipe operasi seperti reader/writer locks. Jika suatu saat thread memperbarui struktur datanya agar dapat memanggil pasangan operasi sinkronisasi rwlock acquire writelock yang mana berfungsi dalam mendapatkan writelock dan rwlock release writelock untuk melepaskannya. Secara umumnya, semaphore writelock untuk memastikan hanya satu writer saja yang mendapatkan lock dan memperbarui struktur datanya dengan masuknya ke critical section.

```
khadziq@khadziq-VirtualBox:~/ostep-code/threads-sema$ ./rwlock readloops writeloops
read done: 0
write done
all done
```

Gambar 4: Membuat shell script di nano

6 Dining Philosophers

6.1 Deadlock

6.1.1 Code Snippets

Berikut ini adalah contoh dari penggunaan \beginlstlisting untuk menulis potongan kode. Dalam kasus ini saya menggunakan bahasa Python. Jika anda menggunakan C atau yang lainnya, tinggal sesuaikan di bagian parameter dari \beginlstlisting. Anda dapat melihatnya pada code snipptes 6

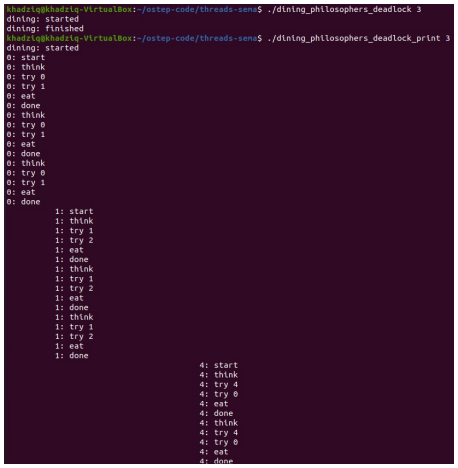
```

    class SynthiaDataset(Dataset):

    #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) {
```

```
45          space(p); printf("%d: try %d\n", p, left(p)); space_end();
46          Sem_wait(&forks[left(p)]);
47          space(p); printf("%d: try %d\n", p, right(p)); space_end();
48          Sem_wait(&forks[right(p)]);
49      }
50
51      void put_forks(int p) {
52          Sem_post(&forks[left(p)]);
53          Sem_post(&forks[right(p)]);
54      }
55
56      void think() {
57          return;
58      }
59
60      void eat() {
61          return;
62      }
63
64      void *philosopher(void *arg) {
65          arg_t *args = (arg_t *) arg;
66
67          space(args->thread_id); printf("%d: start\n", args->thread_id); space_end();
68
69          int i;
70          for (i = 0; i < args->num_loops; i++) {
71        space(args->thread_id); printf("%d: think\n", args->thread_id); space_end();
72        think();
73        get_forks(args->thread_id);
74        space(args->thread_id); printf("%d: eat\n", args->thread_id); space_end();
75        eat();
76        put_forks(args->thread_id);
77        space(args->thread_id); printf("%d: done\n", args->thread_id); space_end();
78          }
79          return NULL;
80      }
81
82      int main(int argc, char *argv[]) {
83          if (argc != 2) {
84        fprintf(stderr, "usage: dining_philosophers <num_loops>\n");
85        exit(1);
86          }
87          printf("dining: started\n");
88
89          int i;
90          for (i = 0; i < 5; i++)
91        Sem_init(&forks[i], 1);
92          Sem_init(&print_lock, 1);
93
94          pthread_t p[5];
95          arg_t a[5];
96          for (i = 0; i < 5; i++) {
97        a[i].num_loops = atoi(argv[1]);
98        a[i].thread_id = i;
99        Pthread_create(&p[i], NULL, philosopher, &a[i]);
100         }
101
102         for (i = 0; i < 5; i++)
103       Pthread_join(p[i], NULL);
104
105         printf("dining: finished\n");
106         return 0;
```

```
107     }
108
```

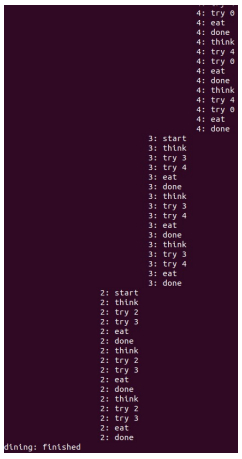
Kode 5: Captionnya tulis di sini class

6.1.2 Gambar



Gambar 5: Membuat teks dengan format txt

Scripting code bash di shell menggunakan nano open editor.



Gambar 6: Shell scripting

6.2 No Deadlock

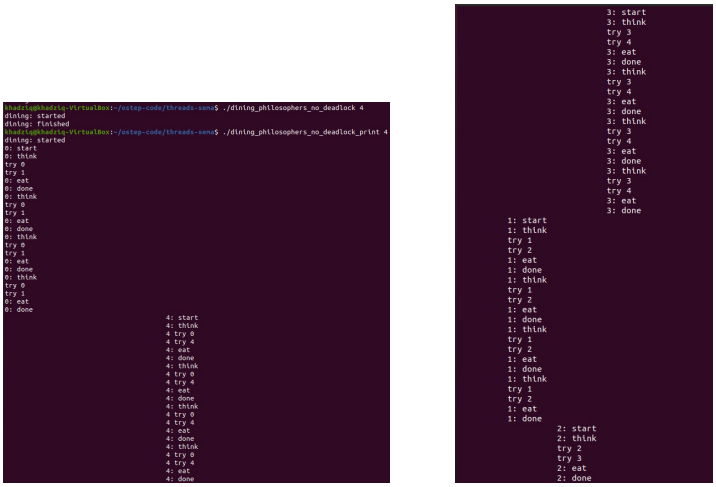
6.2.1 Code Snippets

```

    class SynthiaDataset(Dataset):

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

    #include "common.h"
```

```c
#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)]);
}

void think() {
    return;
}

void eat() {
    return;
}

```

```
71  void *philosopher(void *arg) {
72      arg_t *args = (arg_t *) arg;
73
74      space(args->thread_id); printf("%d: start\n", args->thread_id); space_end();
75
76      int i;
77      for (i = 0; i < args->num_loops; i++) {
78    space(args->thread_id); printf("%d: think\n", args->thread_id); space_end();
79    think();
80    get_forks(args->thread_id);
81    space(args->thread_id); printf("%d: eat\n", args->thread_id); space_end();
82    eat();
83    put_forks(args->thread_id);
84    space(args->thread_id); printf("%d: done\n", args->thread_id); space_end();
85      }
86      return NULL;
87  }
88
89  int main(int argc, char *argv[]) {
90      if (argc != 2) {
91    fprintf(stderr, "usage: dining_philosophers <num_loops>\n");
92    exit(1);
93      }
94      printf("dining: started\n");
95
96      int i;
97      for (i = 0; i < 5; i++)
98    Sem_init(&forks[i], 1);
99      Sem_init(&print_lock, 1);
100
101     pthread_t p[5];
102     arg_t a[5];
103     for (i = 0; i < 5; i++) {
104   a[i].num_loops = atoi(argv[1]);
105   a[i].thread_id = i;
106   Pthread_create(&p[i], NULL, philosopher, &a[i]);
107     }
108
109     for (i = 0; i < 5; i++)
110   Pthread_join(p[i], NULL);
111
112     printf("dining: finished\n");
113     return 0;
114 }
115
```

Kode 6: Captionnya tulis di sini class



(a) Augment Result 1 (b) Augment Result 2

Gambar 7: Augmentation Samples

6.2.2 Gambar

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

7 Kesimpulan

Menggunakan terminal linux secara langsung untuk pertama kali memang banyak kesulitan, namun ketika sudah terbiasa menjalankan program atau mengubah sebuah program lebih *powerful* atau lebih bebas dengan linux karena aksesnya yang luas. Kemudian banyaknya perintah di terminal linux tidak mungkin untuk dihafalkan, tapi jika sering diaplikasikan akan menjadi terbiasa dan tidak perlu lagi melihat dokumentasi. Dan yang saya dapatkan setelah mengerjakannya ialah saya dapat mengerti lebih dalam dengan materi Synchronisation and Deadlock dengan adanya pemberian kode program dari suatu programmer yang membuat programnya sesuai dengan implementasi materi tersebut. Dan begitu juga saya mengerti tentang adanya penggunaan Semaphore pada program yang telah dijalankan. Dengan demikian, saya dapat menjelajahi lebih dalam terkait dengan materi sinkronisasi dan deadlock ini atas tugas Hands On 2 yang telah diberikan.

8 github

https://github.com/Salmon-sky/Sistem-Operasi.git