MODUL PRAKTIKUM #2
1031202/1041202 - Sistem Operasi

Sinkronisasi Proses/Thread

Minggu	**:**	4
Setoran	**:**	Lembar jawaban diserahkan kepada dosen pengampu untuk soal bagian A. Teori (nomor 1 sd. 4) B. Tugas Pemrograman [no 1 sd. 2] Kode Program diunggah ke e-course untuk soal bagian C. Tugas Pemrograman (no 3)
Batas Waktu Setoran	**:**	**04 Maret 2021 pukul 19.00 WIB**
Tujuan	**:**	Mampu menulis program untuk sinkronisasi thread pada Linux menggunakan Semaphore, Conditional Variable, dan Mutex.

Petunjuk Praktikum:

1. Sebelum memberikan pertanyaan silahkan baca buku yang ada pada referensi.
2. Mencontoh pekerjaan dari orang lain akan dianggap plagiarisme dan anda akan ditindak sesuai dengan sanksi akademik yang berlaku di IT Del atau sesuai dengan kebijakan saya dengan memberikan nilai 0.
3. Jawaban diketik menggunakan word processor kemudian dikonversi ke file berekstensi .pdf
4. Penamaan file HARUS sesuai dengan format NIM_Tugas- X_NamaTugas.pdf (contoh: 01_Tugas-2_Struktur_Sistem_Operasi.pdf).
5. Keterlambatan menyerahkan laporan tidak ditolerir dengan alasan apapun. Oleh karena itu, laporan harus dikumpul tepat waktu.
6. Gunakan Sistem Operasi Linux boleh menggunakan Distro apapun namun disarankan untuk mempermudah praktikum gunakan Ubuntu.

Referensi:

1. A. Silberschatz, P.B. Galvin, and G. Gagne, Operating System Concepts, 9th edition, Chapter 1 and 2, John Wiley & Sons, Inc., 2013.
2. M. Neil, S. Richard, Beginning Linux Programming, 4th edition, Wiley, 2008.

A. Teori [35 poin]

1. **[10 poin]** Jelaskanlah latar belakang perlunya sinkronisasi proses/ thread.
2. **[10 poin]** Apa yang dimaksud dengan Race Condition? Berikan contonya!
3. **[15 poin]** Sebutkan dan jelaskan masalah klasik pada sinkronisasi!

B. Pemrograman

Pada bagian ini anda akan menggunakan tiga metode dasar dalam sinkronisasi yaitu Semaphore, Conditional Variable, dan Mutual Exclusion (Mutex).

1. Sinkronisasi dengan Semaphore

Fungsi semaphore tidak dimulai dengan pthread_, seperti kebanyakan fungsi spesifik thread, tetapi dengan sem_. Empat fungsi dasar semaphore digunakan dalam thread diantaranya adalah sem_init, sem_wait, sem_post, dan sem_destroy.
Semaphore dibuat dengan fungsi sem_init dinyatakan sebagai berikut:

```
#include <semaphore.h>

int sem_init(sem_t *sem, int pshared, unsigned int value);
```

Parameter pertama berfungsi untuk menginisialisasi objek semaphore yang ditunjuk oleh sem. Parameter kedua menentukan bagaimana semaphore akan di-*sharing*. Jika bernilai 0, maka semaphore dapat di-*share* antar thread dalam satu proses. Tapi jika bernilai 1 maka semaphore dapat di-*share* antar proses. Parameter terakhir berfungsi untuk memberikan nilai awal ke objek semaphore yang diinisialisasi pada parameter pertama. Hasil dari sem_ini merupakan integer bernilai 0 jika inisialisasi sempahore berhasil dan bernilai 1 jika terdapat *error*.
Pasangan fungsi berikut berguna untuk mengontrol nilai semaphore dan dideklarasikan sebagai berikut:

```
#include <semaphore.h>

int sem_wait(sem_t * sem);

int sem_post(sem_t * sem);
```

Keduanya fungsi diatas mengambil pointer ke objek semaphore yang diinisialisasi pada sem_init. Fungsi sem_post secara atomik meningkatkan nilai semaphore sebesar 1. Sedangkan fungsi sem_wait secara atomik mengurangi nilai semaphore sebesar satu. Jika Anda memanggil sem_wait pada semaphore dengan nilai 2, maka thread akan dieksekusi sehingga nail semaphore menjadi 1. Jika sem_wait dipanggil lagi pada semaphore yang sama dan akan mengakibatkan nilainya menjadi 0, maka thread tersebut akan menunggu untuk dieksekusi sampai ada thread lain yang meningkatkan nilai semaphore tersebut sehingga tidak lagi bernilai 0 melalui sem_post.

Fungsi semaphore terakhir adalah sem_destroy. Fungsi ini digunakan saat Anda semaphore yang anda buat selesai yang dinyatakan sebagai berikut:

```
#include <semaphore.h>

int sem_destroy(sem_t * sem);
```

Fungsi ini mengambil pointer ke semaphore dan merapikan sumber daya yang mungkin dimiliki. Seperti kebanyakan fungsi Linux, semua fungsi ini mengembalikan nilai 0 jika berhasil dan selain 0 jika terdapat error.

Selanjutnya anda diminta untuk membuat program menggunakan metode sempahore dengan mengikuti prosedur di bawah ini.

Prosedur:

1. Buatlah file dengan nama semaphore_basic.c lalu tambahkan code di bawah ini pada file tersebut.

```c
#include <stdio.h>
#include <pthread.h>
#include <semaphore.h>
void myfucl (void *ptr);
void myfuc2 (void *ptr);
char buf[5]; //global variabel
sem_t mutex;
int main(int argc, char *argv[])
{
    pthread_t thread1;
    pthread_t thread2;
    char *msg1 = "Thread 1";
    char *msg2 = "Thread 2";
    sem_init(&mutex, 0,1);
    pthread_create (&thread1, NULL, (void *) &myfucl, (void *) msg1);
    pthread_create (&thread2, NULL, (void *) &myfuc2, (void *) msg2);
    pthread_join(thread1, NULL);
    pthread_join(thread2, NULL);
    sem_destroy(&mutex);
    return 0;
}
void myfucl (void *ptr)
{
    char *msg = (char *)ptr;
    printf("%s\n", msg);
    sem_wait(&mutex);
    sprintf(buf, "%s", "Hello There!");
    sem_post(&mutex);
    pthread_exit(0);
}
void myfuc2 (void *ptr)
{
    char *msg = (char *)ptr;
    printf("%s\n", msg);
    sem_wait(&mutex);
    printf("%s\n", buf);
    sem_destroy(&mutex);
    pthread_exit(0);
}
```

2. Pada program semaphore_basic.c di atas, terdapat suatu global variabel yaitu buf[4] yang akan di akses oleh thread1 dan thread2 melalui fungsi berikut:

```
17      pthread_create (&thread1, NULL, (void *) &myfuc1, (void *) msg1);
18      pthread_create (&thread2, NULL, (void *) &myfuc2, (void *) msg2);
```

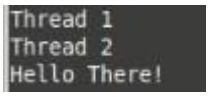
3. Thread1 akan memanggil fungsi myfuc1 dimana fungsi tersebut akan mengisi buf dengan “Hello There!” melalui pernyataan berikut:

```
34      sprintf(buf, "%s", "Hello There!");
```

4. Sedangkan thread2 akan memanggil funsgi myfuc2 dimana fungsi tersbut akan menampilkan isi dari buf memalui pernyataan berikut:

```
45      printf("%s\n", buf);
```

5. Berikut adalah hasil setelah program semaphore_basic.c di jalankan:



Selanjutnya anda diminta untuk membuat program menggunakan metode sempahore dengan kasus producer consumer dengan mengikuti prosedur di bawah ini.

Prosedur:

1. Buatlah file dengan nama semaphore_PC.c lalu tambahkan code di bawah ini pada file tersebut.

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

#define Num_thread 3
#define Buffer_size 10

void producer (void *ptr);
void consumer (void *ptr);
int buffer[Buffer_size];
int in = 0;
int out = 0;
sem_t mutex;
int main(int argc, char *argv[])
{
    pthread_t prod[Num_thread], cons[Num_thread];
    int i;
    char *msg1 = "Thread 1";
    char *msg2 = "Thread 2";
    sem_init(&mutex, 0,1);
    for(i = 0; i< Num_thread; i++){
        pthread_create (&prod[i], NULL, (void *) &producer, (void *) msg1);
    }
        for(i = 0; i< Num_thread; i++){
        pthread_create (&cons[i], NULL, (void *) &consumer, (void *) msg2);
    }
    for(i = 0; i< Num_thread; i++){
        pthread_join(prod[i], NULL);
    }
```

```
31      for(i = 0; i< Num_thread; i++){
32          pthread_join(cons[i], NULL);
33      }
34      sem_destroy(&mutex);
35      return 0;
36  }
37  void producer (void *ptr)
38  {
39      char *msg = (char *)ptr;
40      sem_wait(&mutex);
41      int prod = (rand()) % 1000;
42      buffer[in] = prod;
43      printf("\n producer %d produce %d", in, buffer[in]);
44      in = (in + 1) % Buffer_size;
45      sem_post(&mutex);
46      pthread_exit(0);
47  }
48  void consumer (void *ptr)
49  {
50      char *msg = (char *)ptr;
51      sem_wait(&mutex);
52      int consumed;
53      consumed = buffer[out];
54      printf("\n Consumer %d consume %d", out, consumed);
55      out = (out + 1) % Buffer_size;
56      sem_post(&mutex);
57      pthread_exit(0);
58  }
```

2. Program di atas terdapat beberapa thread yang akan dibuat untuk mengakses fungsi producer dan consumer. fungsi producer yang akan memproduksi suatu nilai pada buffer dan fungsi consumer yang akan mengkonsumsi nilai pada buffer yang telah terisi.

2. Sinkronisasi dengan Conditional Variable

Metode dengan Conditional Variable memungkinkan sejumlah thread menunggu hingga thread lain selesai. Anda dapat mendeklarasikan variabel kondisi menggunakan :

```
pthread_cond_t  myConVar
```

dan deklarasi variabel mutex menggunakan:

```
pthread_mutex_t mymutex;
```

Variabel kondisi selalu digunakan bersama dengan mutex untuk mengontrol aksesnya. Berikut adalah fungsi yang anda gunakan jika ingin membuat variabel kondisi dalam keadaan wait (menunggu).

```
pthread_cond_wait(&myConVar , &mymutex);
```

Anda perlu mengunci mutex sebelum memanggil fungsi di atas. Semua thread yang menunggu pada variabel kondisi akan ditunda hingga thread lain menggunakan fungsi sinyal yang dinyatakan sebagai berikut:

```
pthread_cond_signal(&myConVar);
```

Sama seperti sebelumnya, mutex harus dikunci sebelum memanggil fungsi di atas dan dibuka setelahnya. Fungsi sinyal menyebabkan setidaknya satu utas yang menunggu untuk dimulai kembali (restarted).

Selanjutnya anda diminta untuk membuat program menggunakan metode conditional variable dengan kasus producer consumer dengan mengikuti prosedur di bawah ini.

Prosedur:

1. Buatlah file dengan nama conditional_variable.c lalu tambahkan code di bawah ini pada file tersebut.

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

int max, loops, consumers;
int buffer[4];
int fillIndex, useIndex, count = 0;
pthread_mutex_t mutex;
pthread_cond_t empty, full;

void *producer(void *arg);
void *consumer(void *arg);

int main(int argc, char *argv[]){
    max = 4;
    loops = 4;
    pthread_t prods, consumers;
    pthread_create(&prods, NULL, producer, NULL);
    pthread_create(&consumers, NULL, consumer, NULL);
    pthread_join(prods, NULL);
    pthread_join(consumers, NULL);
    return 0;
}
```

```
24  void put(int value){
25      buffer[fillIndex] = value;
26      fillIndex = (fillIndex + 1) % max;
27      count++;
28  }
29  int get(){
30      int tmp = buffer[useIndex];
31      useIndex = (useIndex + 1) % max;
32      count--;
33      return tmp;
34  }
35  /* producer and consumer function */
36  void *producer(void *arg)
37  {   int i;
38      for (i = 0; i < loops; i++){
39          pthread_mutex_lock(&mutex);
40          while(count == max)
41              pthread_cond_wait(&empty, &mutex);
42          put(i);
43          //count = 0;
44          pthread_cond_signal(&full);
45          pthread_mutex_unlock(&mutex);
46          //printf("%d", count);
47          printf("%s", "Producer mengisi buffer ke = ");
48          printf("%d\n", i);
49      }
50  }
51  void *consumer(void *arg)
52  {   int i;
53      for (i = 0; i < loops; i++){
54          pthread_mutex_lock(&mutex);
55          while(count == 0)
56              pthread_cond_wait(&full, &mutex);
57          int tmp = get();
58          pthread_cond_signal(&empty);
59          pthread_mutex_unlock(&mutex);
60          printf("%s", "Consumer mengkonsumsi buffer ke = ");
61          printf("%d\n", tmp);
62      }
63  }
```

2. Program di atas terdapat beberapa thread yang akan dibuat untuk mengakses fungsi producer dan consumer. fungsi producer yang akan memproduksi suatu nilai pada buffer hingga penuh lalu fungsi consumer yang akan mengkonsumsi nilai pada buffer yang telah terisi penuh hingga kosong.

3. Sinkronisasi dengan MUTEX.

Cara lain untuk menyinkronkan akses dalam program multithreaded adalah dengan mutex (Mutula Exclusion), yang bertindak dengan memungkinkan pemrogram untuk "mengunci" objek sehingga hanya satu thread saja yang dapat mengakses

critical secction. Untuk mengontrol akses ke bagian critical section, anda perlu mengunci mutex terlebih dahulu dan kemudian buka kunci itu setelah selesai. Fungsi dasar yang diperlukan untuk menggunakan mutex sangat mirip dengan yang dibutuhkan untuk semaphores yang dinyatakan sebagai berikut:

```
#include <pthread.h>

int pthread_mutex_init(pthread_mutex_t *mutex, const pthread_mutexattr_t
*mutexattr);

int pthread_mutex_lock(pthread_mutex_t *mutex));

int pthread_mutex_unlock(pthread_mutex_t *mutex);

int pthread_mutex_destroy(pthread_mutex_t *mutex);
```

Seperti biasa, akan mengembalikan nilai 0 jika sukses, dan selain 0 jika terjadi error. Sama seperti semaphores, mutex mengambil pointer ke objek yang dinyatakan sebelumnya, dalam hal ini pthread_mutex_t.
Parameter atribut ekstra pthread_mutex_init memungkinkan Anda untuk menyediakan atribut untuk mutex, yang mengontrol perilakunya.

C. Tugas Pemrograman [65 Poin]

1. **[20 poin]** Pada program semaphore_PC.c, jawablah poin-poin berikut:
   a. Jelaskan fungsi dari For loop pada line 22-24!
   b. Jelaskan fungsi dari For loop pada line 28-30!
   c. Jelaskan cara kerja fungsi producer pada line 37 dan consumer pada line 48!
   d. Jalankan program semaphore_PC.c lalu *screenshot* hasilnya!
2. **[15 poin]**Pada program conditional_variable.c, jawablah poin-poin berikut:
   a. Jelaskan fungsi main pada line 14-22
   c. Jelaskan cara kerja fungsi producer pada line 36 dan consumer pada line 51!
   d. Jalankan conditional_variable.c lalu *screenshot* hasilnya!
3. **[30 poin]** Anda telah menjalankan program semaphore_PC.c dan conditional_variable.c. Sekarang buatlah 2 program menggunakan metode Mutex dimana satu programnya digunakan untuk menggantikan metode semaphore pada program semaphore_PC.c dan program lainnya digunakan untuk menggantikan metode conditional variable pada program conditional_variable.c!