

Laporan Praktikum

Sinkronisasi Proses/Thread

Manajemen Proses



Nama : Josep Phyto Napitupulu

NIM : 11421039

**Program Studi: DIV TEKNOLOGI REKAYASA
PERANGKAT LUNAK**

**INSTITUT TEKNOLOGI DEL
FAKULTAS VOKASI**

A. Teori

1. Jelaskanlah latar belakang perlunya sinkronisasi proses/ thread

Jawab:

Sinkronisasi sangat perlu karena akan dapat mengakibatkan akses secara bersamaan pada data yang di saring mengakibatkan data menjadi inkonsisten dan perlunya sinkronisasi ini juga dapat mengakibatkan maintain data yang konsisten membutuhkan mekanisme untuk memastikan bahwa eksekusi proses dari kooperating proses dilakukan secara berurutan

2. Apa yang dimaksud dengan Race Condition? Berikan contohnya!

Jawab:

Race kondision adalah situasi dimana beberapa peroses mengakses dan memanipulasi data bersama pada saat yang bersamaan. Nilai akhir dari data bersama tersebut tergantung pada proses terakhir selesai.

3. Sebutkan dan jelaskan masalah klasik pada sinkronisasi!

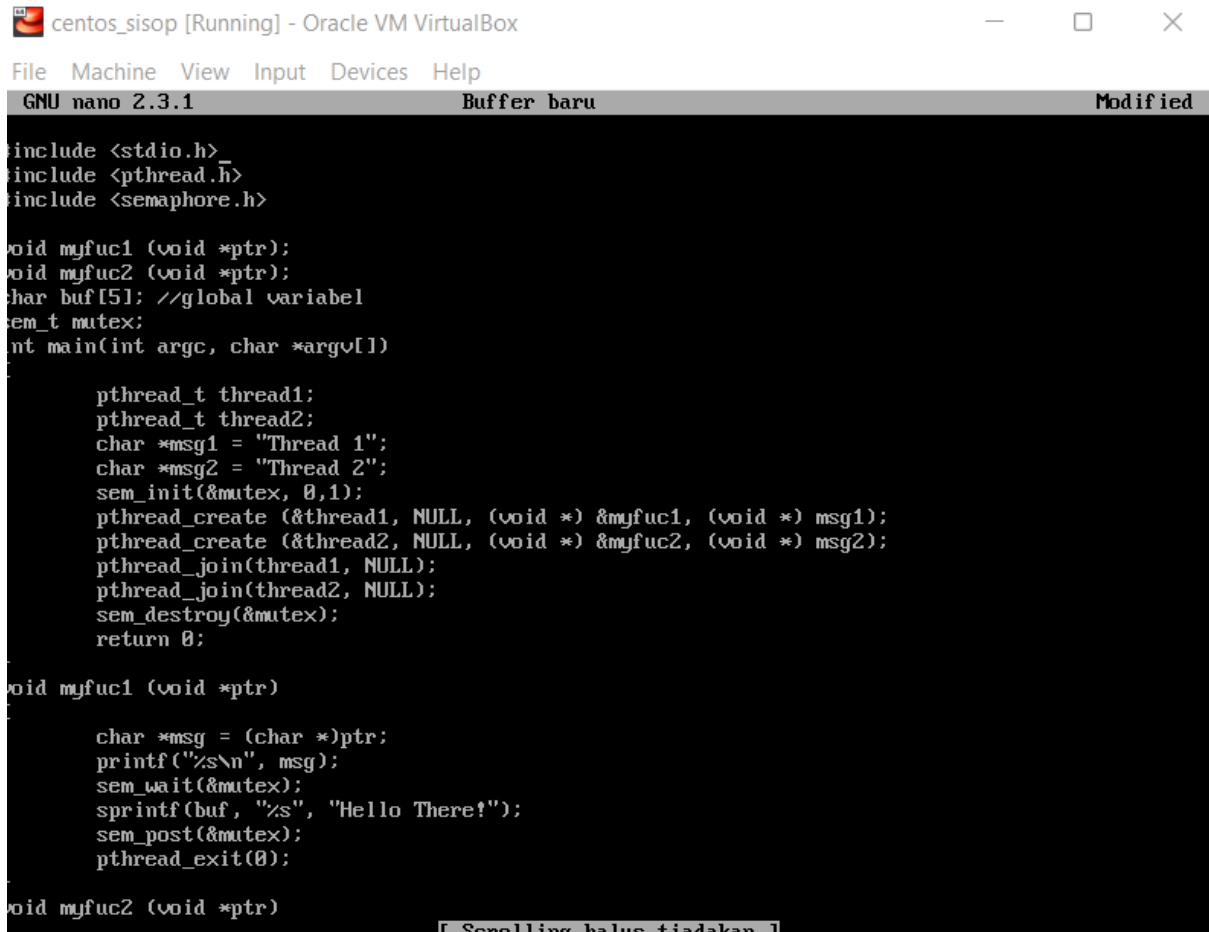
Jawab:

- Problem *Bounded Buffer*
Permasalahan : bagaimana jika dua proses berbeda, yaitu produsen dan konsumen, berusaha mengakses buffer tersebut dalam waktu bersamaan.
- Problem *Dining Philosopher*
Permasalahan : Dalam masalah Dining Philosophers, diketahui sejumlah (N) filsuf yang hanya memiliki tiga status, berpikir, lapar, dan makan. Semua filsuf berada di sebuah meja makan bundar yang ditata sehingga di depan setiap filsuf ada sebuah piring berisi mie dan di antara dua piring yang bersebelahan terdapat sebuah sumpit.
- Problem *Readers and Writers*
Permasalahan : masalah ini terjadi ketika ada beberapa pembaca dan penulis ingin mengakses suatu berkas pada saat bersamaan.

B. Pemograman

1. semaphore_basic.c

Input



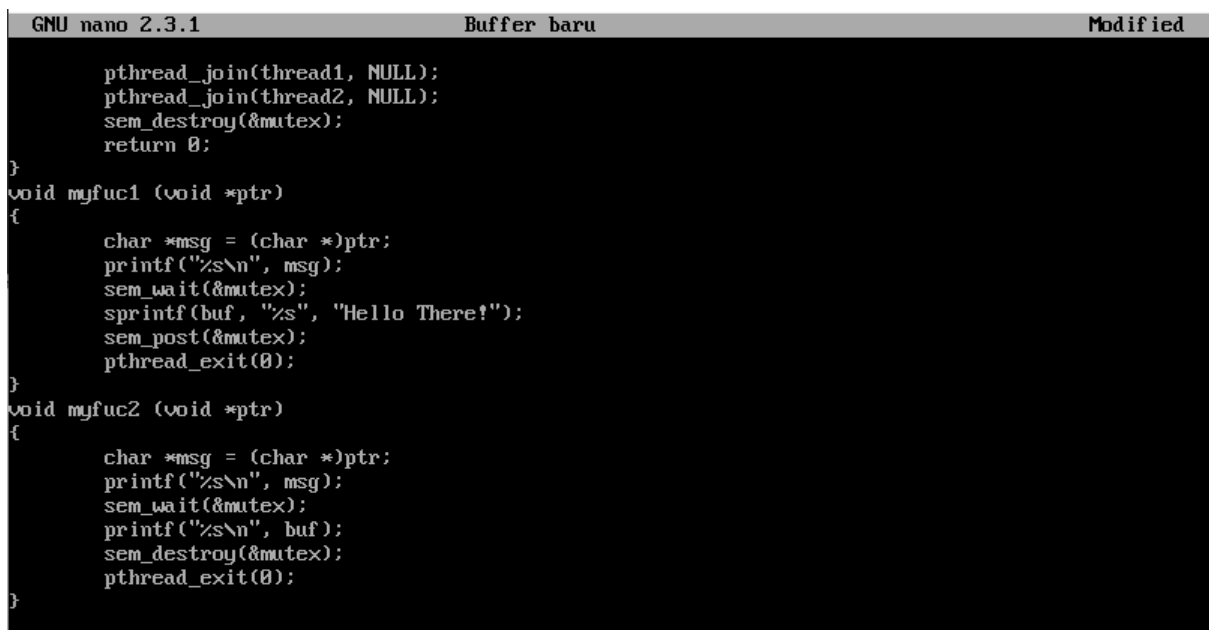
The screenshot shows a terminal window titled "centos_sisop [Running] - Oracle VM VirtualBox". The terminal is running the GNU nano 2.3.1 editor, editing a file named "Buffer baru". The code visible includes headers for stdio.h, pthread.h, and semaphore.h. It defines two functions, myfuc1 and myfuc2, and a main function. The main function creates two threads, thread1 and thread2, and initializes a semaphore. It then calls pthread_create to start the threads, pthread_join to wait for them to finish, and sem_destroy to destroy the semaphore. The myfuc1 function prints the thread ID and the semaphore value, and the myfuc2 function prints the thread ID and the semaphore value.

```
GNU nano 2.3.1 Buffer baru Modified
#include <stdio.h>
#include <pthread.h>
#include <semaphore.h>

void myfuc1 (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 *) &myfuc1, (void *) msg1);
    pthread_create (&thread2, NULL, (void *) &myfuc2, (void *) msg2);
    pthread_join(thread1, NULL);
    pthread_join(thread2, NULL);
    sem_destroy(&mutex);
    return 0;
}

void myfuc1 (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)
```



The screenshot shows the continuation of the code from the previous screenshot. It includes the pthread_join calls, sem_destroy, and the return statement for the main function. It also shows the myfuc1 and myfuc2 functions, which print the thread ID and the semaphore value.

```
    pthread_join(thread1, NULL);
    pthread_join(thread2, NULL);
    sem_destroy(&mutex);
    return 0;
}

void myfuc1 (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);
}
```

Output

```
[ Wrote 40 lines ]

[root@localhost ~]# cc semaphore_basic.c -o semaphore_basic
/tmp/ccrUakXX.o: Dalam fungsi 'main':
semaphore_basic.c:(.text+0x2f): referensi ke 'sem_init' tidak terdefinisi
semaphore_basic.c:(.text+0x4c): referensi ke 'pthread_create' tidak terdefinisi
semaphore_basic.c:(.text+0x69): referensi ke 'pthread_create' tidak terdefinisi
semaphore_basic.c:(.text+0x7a): referensi ke 'pthread_join' tidak terdefinisi
semaphore_basic.c:(.text+0x8b): referensi ke 'pthread_join' tidak terdefinisi
semaphore_basic.c:(.text+0x95): referensi ke 'sem_destroy' tidak terdefinisi
/tmp/ccrUakXX.o: Dalam fungsi 'myfuc1':
semaphore_basic.c:(.text+0xc6): referensi ke 'sem_wait' tidak terdefinisi
semaphore_basic.c:(.text+0xf2): referensi ke 'sem_post' tidak terdefinisi
/tmp/ccrUakXX.o: Dalam fungsi 'myfuc2':
semaphore_basic.c:(.text+0x126): referensi ke 'sem_wait' tidak terdefinisi
semaphore_basic.c:(.text+0x13a): referensi ke 'sem_destroy' tidak terdefinisi
collect2: error: ld menghasilkan status keluaran 1
[root@localhost ~]#
```

2. semaphore_PC.c

Input

```
GNU nano 2.3.1          Buffer baru          Modified
#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);
    }
    for(i = 0; i<Num_thread; i++){
        pthread_join(cons[i], NULL);
    }
}
```

```

GNU nano 2.3.1          Buffer baru          Modified
    }
    for(i = 0; i<Num_thread; i++){
        pthread_join(prod[i], NULL);
    }
    for(i = 0; i<Num_thread; i++){
        pthread_join(cons[i], NULL);
    }
    sem_destroy(&mutex);
    return 0;
}
void producer (void *ptr)
{
    char *msg = (char *)ptr;
    sem_wait(&mutex);
    int prod = (rand()) % 1000;
    buffer[in] = prod;
    printf("\n producer %d produce %d", in, buffer[in]);
    in = (in + 1) % Buffer_size;
    sem_post(&mutex);
    pthread_exit(0);
}
void consumer (void *ptr)
{
    char *msg = (char *)ptr;
    sem_wait(&mutex);
    int consumed;
    consumed = buffer[out];
    printf("\n Consumer %d consume %d", out, consumed);
    out = (out + 1) % Buffer_size;
    sem_post (&mutex);
    pthread_exit((0));
}

```

3. conditional_variable.c

Input

```

GNU nano 2.3.1          Buffer baru          Modifie
#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(consumes, NULL);
    return 0;
}

void put(int value){
    buffer[fillIndex] = value;
    fillIndex = (fillIndex + 1) % max;
    count++;
}

void get(){
    int tmp = buffer[useIndex];
    useIndex = (useIndex + 1) % max;
    [ Scrolling halus tiadakan ]
}

Bantuan    Tulis    Baca File    Hlm sebelumnya    Ptnng Teks    Pos Kursor
X Keluar    J Justifikasi    Di mana    Hlm berikutnya    UnCut Text    Mengeja

```

```
GNU nano 2.3.1 Buffer baru Modified

pthread_create(&prods, NULL, producer, NULL);
pthread_create(&consumers, NULL, consumer, NULL);
pthread_join(prods, NULL);
pthread_join(consumes, NULL);
return 0;
}

void put(int value){
    buffer[fillIndex] = value;
    fillIndex = (fillIndex + 1) % max;
    count++;
}

void get(){
    int tmp = buffer[useIndex];
    useIndex = (useIndex + 1) % max;
    count--;
    return tmp;
}

/*producer and consumer function*/
void *producer(void *arg){
    int i;
    for (i=0; i < loops; i++){
        pthread_mutex_lock(&mutex);
        while(count == max)
            ;
        put(i);
        //count = 0;
        pthread_cond_signal(&full);
        pthread_mutex_unlock(&mutex);
        //printf("%d",count);
        printf("%s", "Producer mengisi buffer ke = ");
        printf("%d\n", i);
    }
}
```

```
GNU nano 2.3.1 Buffer baru Modified

}

/*producer and consumer function*/
void *producer(void *arg){
    int i;
    for (i=0; i < loops; i++){
        pthread_mutex_lock(&mutex);
        while(count == max)
            ;
        put(i);
        //count = 0;
        pthread_cond_signal(&full);
        pthread_mutex_unlock(&mutex);
        //printf("%d",count);
        printf("%s", "Producer mengisi buffer ke = ");
        printf("%d\n", i);
    }
}

void *consumer(*arg){
    int i;
    for(i = 0; i < loops; i++){
        pthread_mutex_lock(&mutex);
        while(count == 0)
            ;
        pthread_cond_wait(&empty);
        int tmp = get();
        pthread_cond_signal(&empty);
        pthread_mutex_unlock(&mutex);
        printf("%s", "Consumer mengkonsumsi buffer ke = ");
        printf("%d\n",tmp);
    }
}
```

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!
 - Fungsi for loop dari baris ini adalah untuk mengecek perulangan yang mana akan menyimpan alamat dari variabel pthread_create dan di bentuk menjadi index[i].
 - b. Jelaskan fungsi dari For loop pada line 28-30!

- Fungsi for loop dari baris ini adalah untuk mengecek perulangan yang mana akan menyimpan alamat dari variabel pthread_Join dan di bentuk menjadi index[i].
- c. Jelaskan cara kerja fungsi producer pada line 37 dan consumer pada line 48!
- Kinerja dari Procedur ini ialah mendeklarasikan variable yang mana terdapat penjumlahan dan mencari hasil dari buffer.
- d. Jalankan program semaphore_PC.c lalu screenshot hasilnya!

```
producer 0 producer 41
producer 1 producer 41
producer 2 producer 41
Consumer 0 consume 41
Consumer 1 consume 41
Consumer 2 consume 41
-----
```

2. **[15 poin]** Pada program conditional_variable.c, jawablah poin-poin berikut:
- Jelaskan fungsi main pada line 14-22!
 - Fungsi main pada line tersebut adalah perintah yang mana merupakan inti dari program tersebut.
 - Jelaskan cara kerja fungsi producer pada line 36 dan consumer pada line 51!
 - Pada fungsi ini merupakan kinerja perulangan atau loop yang mana pada fungsi ini dilakukan pengecekan.jika dia bernilai true maka perulangan akan terus berlanjut sehingga pengecekan bernilai false.
 - Jalankan conditional_variable.c lalu screenshot hasilnya!

```
Producer mengisi buffer ke =0
Producer mengisi buffer ke =1
Producer mengisi buffer ke =2
Producer mengisi buffer ke =3
Consumer mengkonsumsi buffer ke = 0
Consumer mengkonsumsi buffer ke = 1
Consumer mengkonsumsi buffer ke = 2
Consumer mengkonsumsi buffer ke = 3
```

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!

```

#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;
}

void put(int value)
{
    buffer[fillIndex] = value;
    fillIndex = (fillIndex + 1) % max;
    count++;
}

int get()
{
    int tmp = buffer[userIndex];
    userIndex = (useIndex + 1) % max;
    count--;
}

pthread_join(prods, NULL);
pthread_join(consumers, NULL);
return 0;
}

void put(int value)
{
    buffer[fillIndex] = value;
    fillIndex = (fillIndex + 1) % max;
    count++;
}

int get()
{
    int tmp = buffer[userIndex];
    userIndex = (useIndex + 1) % max;
    count--;
    return tmp;
}

/*producer and consumer function*/
void *producer(void *arg)
{
    int i;
    for(i = 0; i < loops; i++)
    {
        pthread_mutex_lock(&mutex);
        while(count == max)
            pthread_cond_wait(&empty, &mutex);
        put(i);
        //count = 0
        pthread_cond_signal(&full);
        pthread_mutex_unlock(&mutex);
        //printf("%d", count);
        print("%s", "Producer mengisi buffer ke =");
    }
}

```



```

        printf("%s", "Producer mengisi buffer ke = ");
        printf("%d\n", i);
    }
}

void *consumer(void *arg)
{
    int i;
    for (i = 0; i < loops; i++)
    {
        pthread_mutex_lock(&mutex);
        while (count == 0);
        pthread_cond_wait(&full, &mutex);
        int tmp = get();
        pthread_cond_signal(&empty);
        pthread_mutex_unlock(&mutex);
        printf("%s", "Consumer mengkonsumsi buffer ke = ");
        printf("%d\n", tmp);
    }
}

```

```

}

void *consumer(void *arg)
{
    int i;
    for (i = 0; i < loops; i++)
    {
        pthread_mutex_lock(&mutex);
        while (count == 0);
        pthread_cond_wait(&full, &mutex);
        int tmp = get();
        pthread_cond_signal(&empty);
        pthread_mutex_unlock(&mutex);
        printf("%s", "Consumer mengkonsumsi buffer ke = ");
        printf("%d\n", tmp);
    }
}

```