SINKRONISASI PADA THREAD

MUTEX
SEMAPHORE

- Saat banyak thread yang running secara concurrent, mereka butuh komunikasi satu sama lain (sinkronisasi)

THREAD A	THREAD B

a = r;
a++;
r = a;
printf("r = %d\n", r);
	b = r;
	b -- ;
	r = b;
	printf("r = %d\n", r);

- Berapa nilai r ? Nilai r Sulit diprediksi karena kedua thread saling berebut menggunakan variable r sehingga hasil akhir dari r menjadi sulit diprediksi.
- Salah satu Sinkronisasi : Mutual Exclusion

Race Condition

- Sintaks #include<pthread.h> pada baris ke-5 adalah file header untuk mengakses POSIX thread library.
- Baris 24 pthread_create() digunakan untuk membuat thread baru.
- Function tambah pada baris ke 10 s/d 19 adalah aktifitas dari thread yang barusan dibuat atau thread T0.
- Baris 33 yaitu pthread_join() digunakan untuk menunggu thread T0 selesai dieksekusi.
- Ada variable global bernama bilangan yang awalnya bernilai nol (0).
- Baris 26 s/d 31 variabel bilangan ditambah satu sampai bilangan bernilai 20.
- Baris 12 s/d 17 variabel bilangan juga ditambah satu sampai bilangan bernilai 20.
- Berapa hasil akhir ?

```
student@debian8:~$ ./race_condition
Nilai Bilangan Awal = 0
Nilai Bilangan Akhir = 20
student@debian8:~$ ./race_condition
Nilai Bilangan Awal = 0
Nilai Bilangan Akhir = 20
```

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

int bilangan = 0;
pthread_t T0;

void *tambah(void *a) {
    int i,j;
    for (i = 0; i < 20; i++) {
        j = bilangan;
        j++;
        sleep(1);
        bilangan = j;
    }
    return NULL;
}

int main() {
    int i,j;
    printf("Nilai Bilangan Awal = %i\n", bilangan);
    if(pthread_create(&T0, NULL, tambah, NULL)==-1)
       error("thread tidak bisa dibuat");
    for ( i=0; i<20; i++) {
        j = bilangan;
        j++;
        sleep(1);
        bilangan = j;
    }
    void* result;
    pthread_join(T0, &result);
    printf("Nilai Bilangan Akhir = %i\n", bilangan);
    return 0;
}
```

Thread B

Thread A

Fungsi PTHREAD JOIN

- Fungsi Thread join untuk membuat menunggu thread default menunggu sementara waktu saat thread-thread lain sedang dieksekusi
- Hampir sama dengan fungsi wait

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

/* record yang berisi parameter yang akan dicetak. */
struct char_print_parms {
        /* karakter yang akan dicetak. */
        char character;

        /* jumlah pencetakan. */
        int count;
};

void* char_print (void* parameters) {
        /* lakukan casting cookie pointer agar memiliki tipe yang benar. */
        struct char_print_parms* p = (struct char_print_parms*) parameters;
        int i;
        for (i = 0; i < p->count; ++i){
                fputc (p->character, stderr);
                sleep(1); }
        return NULL;
}
int main (){
        pthread_t thread1, thread2;
        struct char_print_parms data_thread1, data_thread2;
        data_thread1.character = 'x';
        data_thread1.count = 5;
        pthread_create (&thread1, NULL, &char_print, &data_thread1);
        data_thread2.character = 'o';
        data_thread2.count = 5;
        pthread_create (&thread2, NULL, &char_print, &data_thread2);
        pthread_join (thread1, NULL);
        pthread_join (thread2, NULL);
        return 0;
}
```

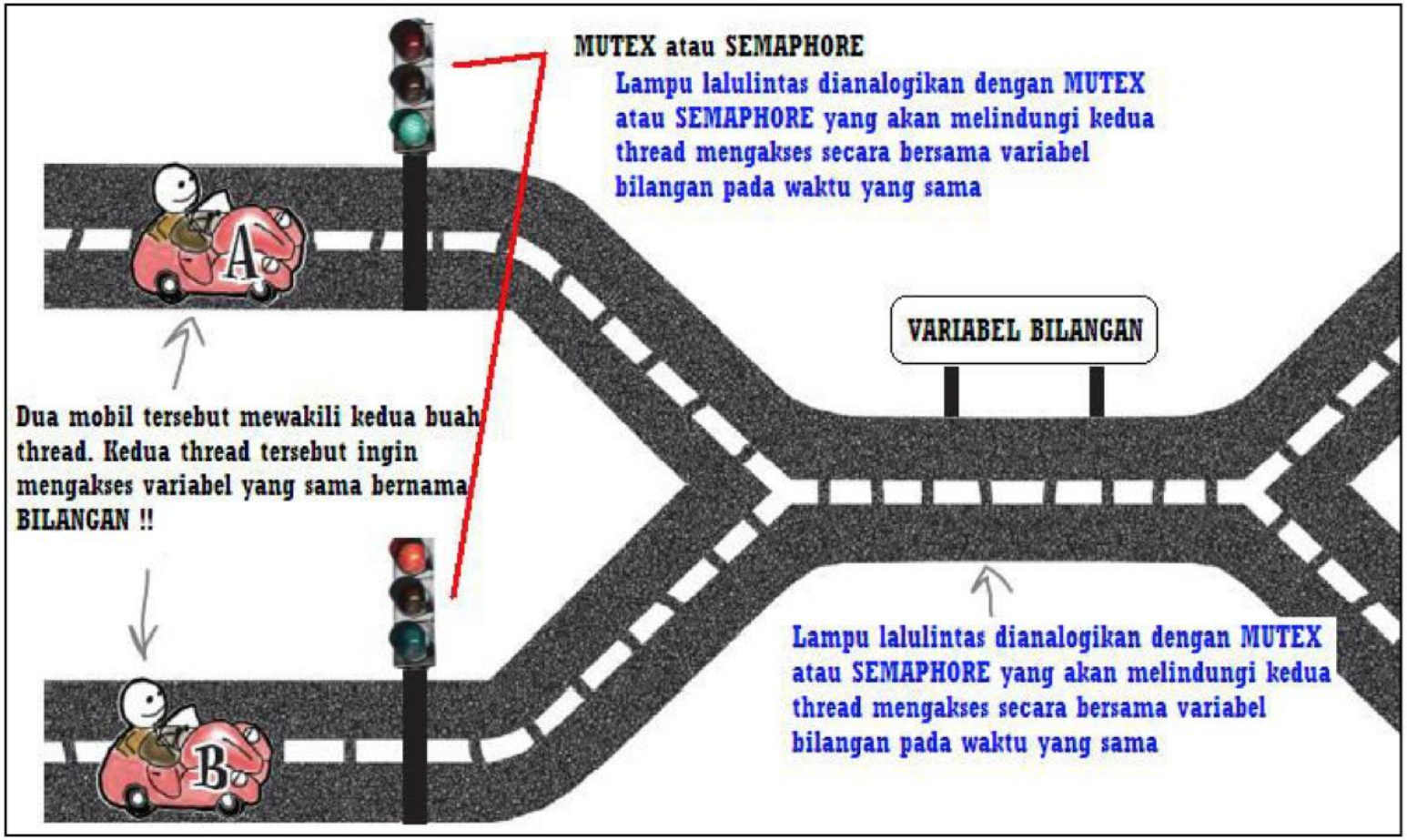
Tanpa Join

```
student@debian8:~$ ./thread_join
student@debian8:~$
```

Dengan Join

```
student@debian8:~$ ./thread_join
oxoxoxoxoxstudent@debian8:~$
```

MUTEX


MUTEX atau SEMAPHORE
Lampu lalulintas dianalogikan dengan MUTEX atau SEMAPHORE yang akan melindungi kedua thread mengakses secara bersama variabel bilangan pada waktu yang sama
VARIABEL BILANGAN
A
B
Dua mobil tersebut mewakili kedua buah thread. Kedua thread tersebut ingin mengakses variabel yang sama bernama BILANGAN !!
Lampu lalulintas dianalogikan dengan MUTEX atau SEMAPHORE yang akan melindungi kedua thread mengakses secara bersama variabel bilangan pada waktu yang sama

MUTEX

- mutex lock sebagai pengatur lalu lintas sinyal.

```
pthread_mutex_t a_lock = PTHREAD_MUTEX_INITIALIZER;
```

- Mutex harus diset sebagai variable global karena seluruh thread yang sedang running akan melihat/menggunakan mutex ini

Proses MUTEX

- tentukan dulu apa atau bagian mana dari kode program di atas yang harus menjadi critical section.

```
for (i = 0; i < 20; i++) {
    j = bilangan;
    j++;
    sleep(1);
    bilangan = j;
}
```

Critical Section

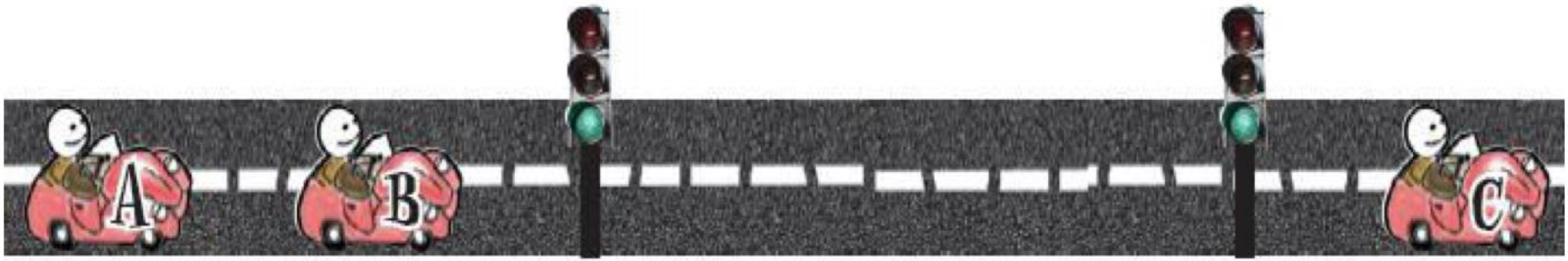
- Hanya satu proses yang bisa akses critical section


```
pthread_mutex_lock(&a_lock);
/* Critical Section ... */
```

Hanya satu thread saja yang akan masuk/mengakses bagian ini

Proses MUTEX

- Selesai Critical section, unlock signal




```
/*  .. Akhir dari Critical Section */
pthread_mutex_unlock(&a_lock);
```

Dengan MUTEX

```
x@x-laptop:~$ gcc bilangan_mutex.c -lpthread -o test
x@x-laptop:~$ ./test
Nilai Bilangan Awal = 0
Nilai Bilangan Akhir = 40
x@x-laptop:~$
```

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

pthread_mutex_t bilangan_lock = PTHREAD_MUTEX_INITIALIZER;
int bilangan = 0;
pthread_t T0;

void *tambah(void *a) {
    int i,j;
    pthread_mutex_lock(&bilangan_lock);
    for (i = 0; i < 20; i++) {
        j = bilangan;
        j++;
        sleep(1);
        bilangan = j;
    }
    pthread_mutex_unlock(&bilangan_lock);
    return NULL;
}

int main() {
    int i,j;
    printf("Nilai Bilangan Awal = %i\n", bilangan);
    if(pthread_create(&T0, NULL, tambah, NULL)==-1)
       error("thread tidak bisa dibuat");
    pthread_mutex_lock(&bilangan_lock);
    for ( i=0; i<20; i++) {
        j = bilangan;
        j++;
        sleep(1);
        bilangan = j;
    }
    pthread_mutex_unlock(&bilangan_lock);
    void* result;
    pthread_join(T0, &result);
    printf("Nilai Bilangan Akhir = %i\n", bilangan);
    return 0;
}
```

Critical Section

Critical Section

Semaphore

- Semaphore adalah counter atau penanda yang digunakan mengatur sinkronisasi saat proses atau thread berbagi (shared) sumber daya yang sama pada saat yang sama.
- Pada saat berbagi sumber daya bersama maka harus ada jaminan bahwa hanya satu proses yang mengakses sumber daya tersebut pada suatu waktu tertentu (mutual exclusion).

CARA KERJA SEMAPHORE

- Dua Operasi Semaphore
  - Operasi Wait (nama lainnya: P (Proberen) atau Down atau Lock)
  - Operasi Signal (nama lainya: V (Verhogen) atau Up atau Unlock atau Post)
- Wait maka nilai counter berkurang 1.
  - Jika counter bernilai negative maka proses/thread ini akan dipaksa berhenti (sleep).
  - Proses/thread yang diberi operasi Wait akan dipaksa berhenti sampai proses/thread tersebut diberi tanda Signal.
- Signal maka nilai counter bertambah 1.
  - Proses/thread yang diberi operasi/tanda Signal akan running dan harus berhenti jika diberi tanda Wait.
  - Kalo proses/thread lebih besar atau sama dengan nol maka akan running (unblock).

CARA KERJA SEMAPHORE

- Dengan menggunakan semaphore maka dapat diatur thread mana yang harus running terlebih dahulu sehingga kedua thread tidak berebutan menggunakan variable bilangan.

THREAD A

```
for (i = 0; i < 20; i++) {
   j = bilangan;
   j++;
   sleep(1);
   bilangan = j;
}
```

THREAD B

```
for (i = 0; i < 20; i++) {
   k = bilangan;
   k++;
   sleep(1);
   bilangan = k;
}
```

```
bilangan = 0
Init (&yosua, 0);
```

THREAD A

```
for (i = 0; i < 20; i++) {
   j = bilangan;
   j++;
   sleep(1);
   bilangan = j;
}

Signal(&yosua);
```

THREAD B

```
Wait(&yosua);
for (i = 0; i < 20; i++) {
   k = bilangan;
   k++;
   sleep(1);
   bilangan = k;
}
```

CONTOH SEMAPHORE

```
int bilangan = 0; // VARIABEL GLOBAL YANG DISHARE BERSAMA ANTARA
// THREAD 1 DAN THREAD TAMBAH
pthread_t T0;
int semaphore; /* variabel semaphore */

// FUNCTION DARI SEMAPHORE
void sem_p(int id, int value){ // untuk ganti nilai semaphore dengan -1 atau 1
        struct sembuf sem_b;
        int v;
        sem_b.sem_num = 0; sem_b.sem_op = -1; /* P() */
        sem_b.sem_flg = SEM_UNDO;
        if (semop(id, &sem_b, 1) == 1)
                fprintf(stderr, "\nError...Semaphore P Decrement Gagal");
}
void sem_v(int id, int value){ // untuk ganti nilai semaphore dengan -1 atau 1 struct sembuf sem_b;
        int v;
        struct sembuf sem_b;
        sem_b.sem_num = 0;
        sem_b.sem_op = 1; /* V() */
        sem_b.sem_flg = SEM_UNDO;
        if(semop(id, &sem_b, 1) == -1)
                fprintf(stderr, "\nError...Semaphore V Increment Gagal");
}
void sem_create(int semid, int initval){
        int semval;
        union semun { int val;
                struct semid_ds *buf; unsigned short *array;
        } s;

        s.val = initval;
        if((semval = semctl(semid, 0, SETVAL, s)) < 0)
                fprintf(stderr,"\nsemctl error....");
}

void sem_wait(int id){ // Decrement P
        int value = -1;
        sem_p(id, value);
}

void sem_signal(int id){ // Increment V
        int value = 1;
        sem_v(id, value);
}
// END FUNCTION SEMAPHORE
```

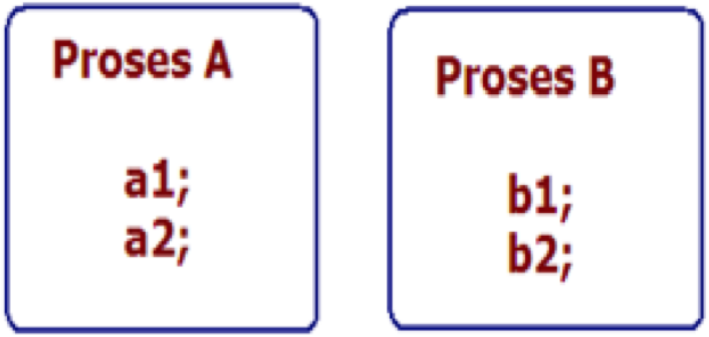
```
void *tambah(void *a) {
        int i,j; sem_wait(semaphore);
        for (i = 0; i < 20; i++) {
                j = bilangan;
                j++;
                sleep(1);
                bilangan = j;
        }
return NULL;
}

int main() {
        int i,j;
        printf("Nilai Bilangan Awal = %i\n", bilangan);
        // BUAT SEMAPHORE "semaphore"
        if((semaphore = semget(IPC_PRIVATE, 1, 0666|IPC_CREAT)) == -1){
                printf("\nError... Tidak bisa buat semaphore");
                exit(1);
        }
        sem_create(semaphore, 0);
        if(pthread_create(&T0, NULL, tambah, NULL)==-1) error("thread tidak bisa dibuat");
        // THREAD INI YANG RUNNING DULUAN KEMUDIAN THREAD TAMBAH
        for ( i=0; i<20; i++) {
                j = bilangan;
                j++; sleep(1);
                bilangan = j;
        }
        sem_signal(semaphore);
        void* result;
        pthread_join(T0, &result);
        printf("Nilai Bilangan Akhir = %i\n", bilangan); return 0;
}
```

```
x@x-laptop:~$ ./test
Nilai Bilangan Awal = 0
Nilai Bilangan Akhir = 40
x@x-laptop:~$
```

- Awalnya thread B **akan dipaksa berhenti** karena ada operasi Wait (nilai semaphore yang awalnya bernilai nol akan dikurangi 1 sehingga = -1)
- Thread A langsung bekerja. Saat thread A selesai, signal dijalankan yang mebuat nilai semaphore =0
- Setelah semaphore bernilai 0, maka thread B akan bekerja.

STUDI KASUS

- Terdapat 2 proses A dan B dimana masing-masing proses memiliki 2 buah thread
- Diatur agar Proses A thread a1 kerja lebih dahulu diikuti oleh Proses B thread b1 Selanjutnya thread b2 akan dieksekusi dan terakhir adalah eksekusi thread a2.



a1 → b1 → b2 → a2

STUDI KASUS (SOLUSI)

- Gunakan 2 buah semaphore

```
init (&yosua, 0);
init (&sir, 1);
```

Proses A

```
Wait (&sir);
a1;
Signal (&yosua);
Wait (&sir);
a2;
```

Proses B

```
Wait (&yosua);
b1;
b2;
Signal (&sir);
```