

KOMUNIKASI ANTAR PROSES

Saat banyak thread yang running secara concurrent, mereka butuh komunikasi satu sama lain (sinkronisasi). Berikut adalah contoh dua proses melakukan akses terhadap variabel yang sama .

<u>THREAD A</u>	<u>THREAD B</u>
<pre>a = r; a++; r = a; printf("r = %d\n", r);</pre>	<pre>b = r; b -- ; r = b; printf("r = %d\n", r);</pre>

Berapa hasil nilai r dari proses A dan B? Nilai r Sulit diprediksi karena kedua thread saling berebut menggunakan variable r sehingga hasil akhir dari r menjadi sulit diprediksi. Perlu adanya sinkronisasi untuk memastikan nilai r konsisten.

Terdapat beberapa cara sinkronisasi, dan dalam praktikum kali ini kita akan mencoba Sinkronisasi dengan Mutual Exclusion dan semaphore.

1. Sinkronisasi Mutual Exclusion

Jalankan program berikut, lakukan tracing program dan analisa hasilnya. Kenapa bisa seperti itu ?

```
#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

Berikut adalah solusi dari program diatas dengan mutual exclusion

```
#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

2. Sinkronisasi dengan 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 Operasi semaphore :

- Dua Operasi Semaphore
 - Operasi Wait (nama lainnya: P (Proberen) atau Down atau Lock)
 - Operasi Signal (nama lainnya: 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).

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

Ketikkan program berikut, buat flowchart dari program berikut dan analisa hasil program berikut :

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <pthread.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <sys/sem.h>
#include <string.h>

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

// THREAD
void *tambah(void *a) {
    int i,j; sem_wait(yosua);
    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 "yosua"
    if((yosua = semget(IPC_PRIVATE, 1, 0666|IPC_CREAT)) == -1){
        printf("\nError... Tidak bisa buat semaphore yosua");
        exit(1); }
    sem_create(yosua, 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(yosua);
    void* result;
    pthread_join(T0, &result);
    printf("Nilai Bilangan Akhir = %i\n", bilangan); return 0;
}

```

Implementasi kasus pertama dengan semaphore.

Jalankan program berikut, lakukan tracing program dan analisa hasilnya. Kenapa bisa seperti itu ?

```

#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

Berikut adalah solusi dari program diatas dengan semaphore. Jalankan program ini dan analisa hasilnya serta buat algoritma prinsip kerja semaphore pada program ini.

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<errno.h>
#include<pthread.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <sys/sem.h>
#include <string.h>
```

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

```
int yosua; /* 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;
    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

// THREAD
void *tambah(void *a) {
    int i,j;

    sem_wait(yosua);

    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 "yosua"
    if((yosua = semget(IPC_PRIVATE, 1, 0666|IPC_CREAT)) == -1){
        printf("\nError... Tidak bisa buat semaphore yosua");
        exit(1);
    }
    sem_create(yosua, 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(yosua);

    void* result;
    pthread_join(T0, &result);
    printf("Nilai Bilangan Akhir = %i\n", bilangan);
    return 0;
}

```