Sinkronisasi Thread

Praktikum Sistem Operasi

Ilmu Komputer IPB

2019

Critical Section

Critical Section

> *A **critical section** is a section of code that can be executed by at most **one thread at a time**. The critical section exists to protect shared resources from multiple access.*[1]

- contoh:
  - mengubah variabel global
  - mengubah *database*
  - menulis ke *file*
- proteksi dengan sinkronisasi

[1] Jones (2008), *GNU/Linux Application Programming*, hlm 264.

Sinkronisasi

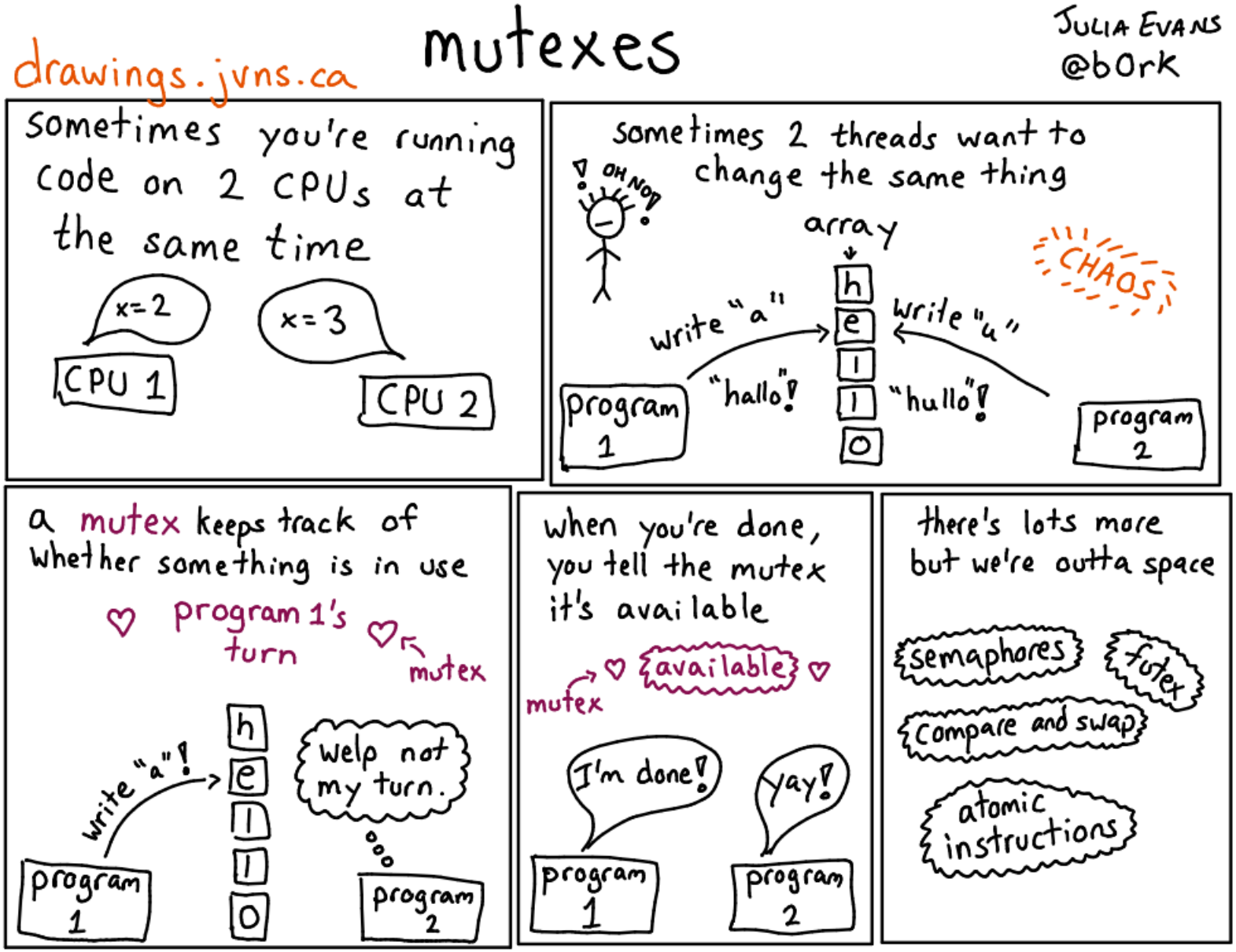
- melindungi (mengunci) sebuah *critical section*
  - hanya satu *thread* dalam satu waktu yang dapat masuk
- implementasi:
  - *mutex lock*
  - *semaphore*

Mutual Exclusion

Mutex

*Mutex is a **key** to a variable. One thread can have the key—modify the variable—at the time. When finished, the thread gives (frees) the key to the next thread in the group.*[2]

[2] http://koti.mbnet.fi/niclasw/MutexSemaphore.html


mutexes
drawings.jvns.ca
Julia Evans
@b0rk
sometimes you're running code on 2 CPUs at the same time
x=2
x=3
CPU 1
CPU 2
sometimes 2 threads want to change the same thing
Oh No!
array
CHAOS
write "a"
write "u"
"hallo"!
"hullo"!
program 1
program 2
h
e
l
l
o
a mutex keeps track of whether something is in use
♡ program 1's turn ♡
mutex
write "a"!
welp not my turn.
program 1
program 2
when you're done, you tell the mutex it's available
mutex
♡ available ♡
I'm done!
yay!
program 1
program 2
there's lots more but we're outta space
semaphores
futex
compare and swap
atomic instructions

Fungsi Mutex

```
#include <pthread.h>

pthread_mutex_t mutex;

pthread_mutex_init(&mutex, &attr);
pthread_mutex_lock(&mutex);
pthread_mutex_unlock(&mutex);
pthread_mutex_destroy(&mutex);
```

- `init`: inisialisasi `mutex`
  - `attr` *default* isi dengan `NULL`
- `lock`: mendapatkan kunci *critical section*
- `unlock`: melepas kunci *critical section*
- `destroy`: menghapus `mutex`

Latihan

Apa yang salah dengan kode berikut ini? Perbaiki dengan menggunakan *mutex*!

```c
// counting to one million
#include <stdio.h>
#include <pthread.h>

#define N 1000000
#define T 4

int count = 0;

void *counting(void *arg)
{
    int i;
    for (i = 0; i < N/T; i++)
        count++;        // critical section

    pthread_exit(NULL);
}
```

```c
int main()
{
    pthread_t t[T];
    int i;

    for (i = 0; i < T; i++)
        pthread_create(&t[i], NULL, counting, NULL);

    for (i = 0; i < T; i++)
        pthread_join(t[i], NULL);

    printf("%d\n", count);      // 1000000, no?
    return 0;
}
```

Semaphore

Semaphore

- nilai *semaphore* `S`: bilangan non-negatif
- terdapat dua operasi atomik yang bisa dilakukan pada *semaphore*, yaitu `wait` dan `post`[3]

```
wait(S) {
    while (S == 0)
        ; // busy wait
    S--;
}

post(S) {
    S++;
}
```

[3] Silberschatz *et al.* (2013), *Operating System Concepts*, hlm 214.

Jenis Semaphore

1. *Binary semaphore*
   - nilai awal *semaphore* $= 1$
   - sama fungsinya dengan *mutex*
2. *Counting semaphore*
   - nilai awal *semaphore* > 1

Fungsi Semaphore

```
#include <semaphore.h>

sem_t sem;

sem_init(&sem, pshared, value);
sem_wait(&sem);
sem_post(&sem);
sem_destroy(&sem);
```

- `init`: inisialisasi `sem` dengan nilai awal `value`
- `wait`:
  - selama `sem = 0` → *busy wait*
  - hingga `sem > 0` → `sem--`, *continue*
- `post`: `sem++`
- `destroy`: menghapus `sem`

Latihan

Perbaiki latihan sebelumnya dengan menggunakan *semaphore*!

Tugas

Big Array Sum

Identifikasi *critical section* dan perbaiki kode berikut ini supaya hasilnya benar. Kumpulkan di LMS paling lambat hingga praktikum berakhir.

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

#define N 100000
#define T 4

int sum = 0;

void *array_sum(void *arg)
{
    int *A = (int*)arg;          // cast void* --> int*
    int i;

    for (i = 0; i < N/T; i++)
        sum += A[i];

    pthread_exit(NULL);
}
```

```c
int main()
{
    pthread_t t[T];
    int A[N], i;

    for (i = 0; i < N; i++)
        A[i] = rand()%10;

    for (i = 0; i < T; i++)
        pthread_create(&t[i], NULL, array_sum, &A[i*N/T]);

    for (i = 0; i < T; i++)
        pthread_join(t[i], NULL);

    printf("%d\n", sum);     // 448706
    return 0;
}
```

Penilaian

- +80: keluaran selalu benar (jalankan min 10 kali)
- +20: jumlah *syscall* `clone`[4] ada 4

[4] cek dengan perintah `'strace -ce clone ./program'`