Thread

Praktikum Sistem Operasi

Ilmu Komputer IPB

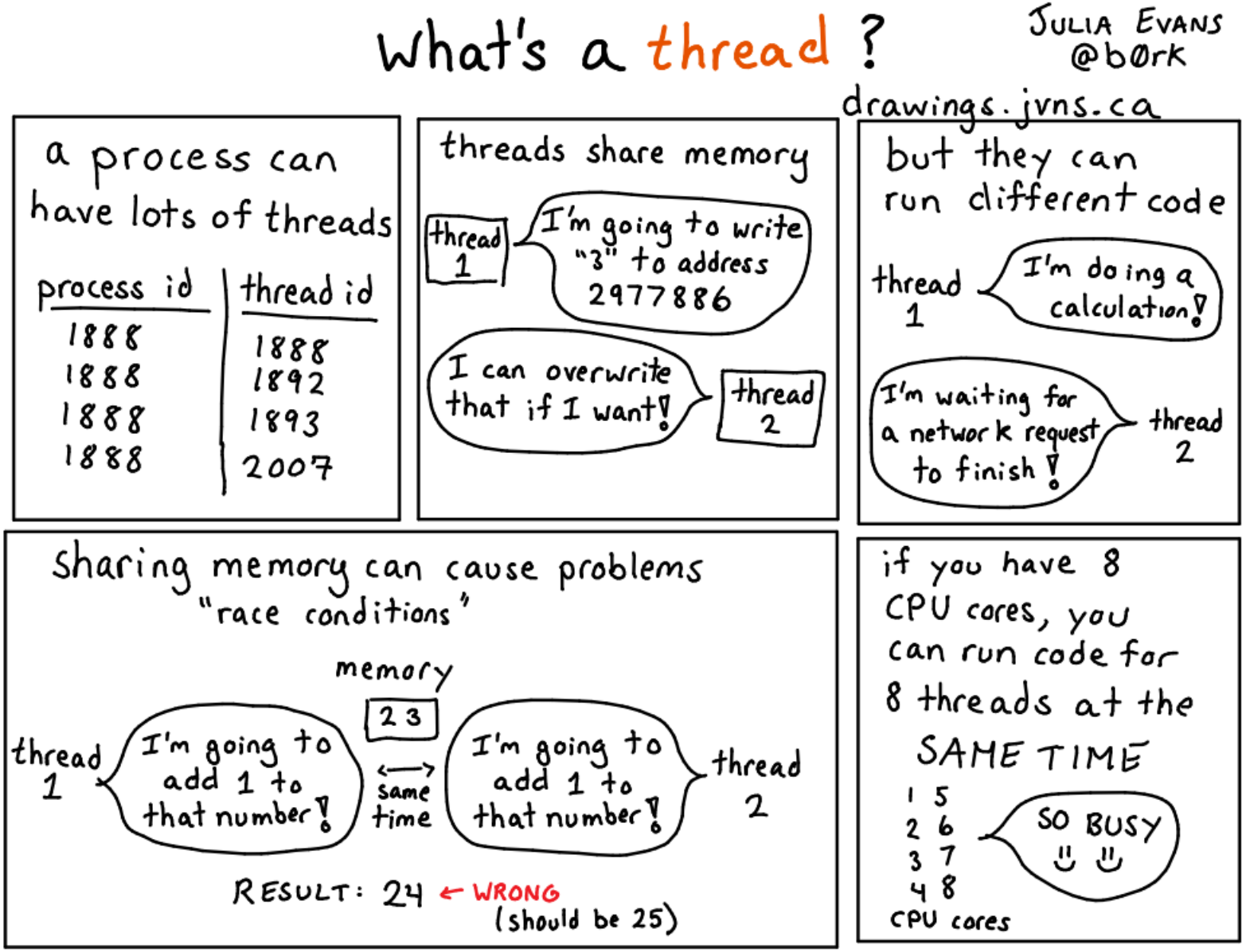
2023

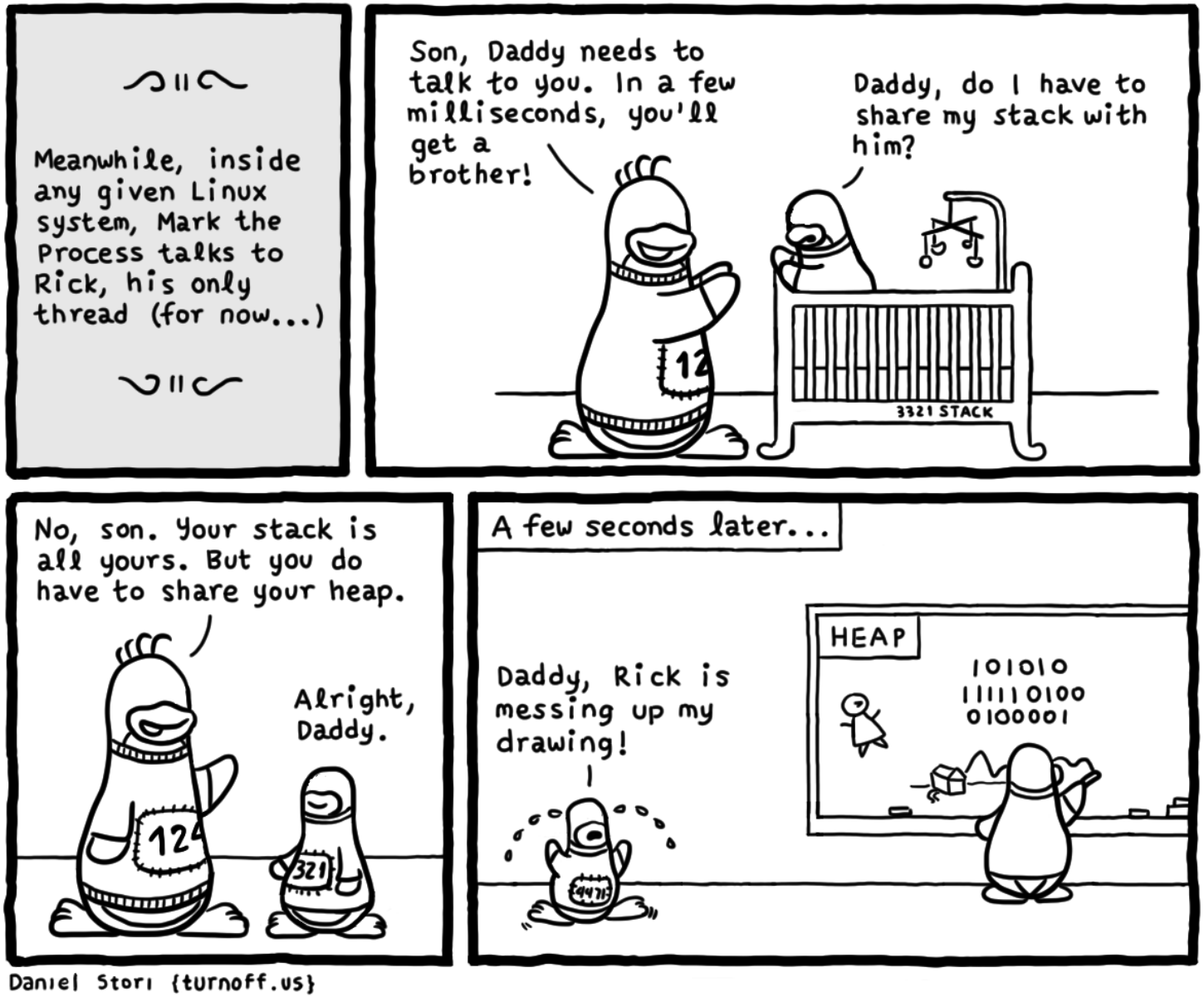
Thread

Thread

- *thread* adalah satuan dasar utilisasi CPU[1]
- tiap *thread* memiliki:
  - id, *program counter*, *register set*, dan *stack*
- dalam satu proses, *thread* berbagi:
  - segmen teks, data, dan *heap*, serta sumberdaya lain (mis. *file*)
- proses *multithreaded* memiliki beberapa *thread* yang dapat mengerjakan beberapa tugas secara bersamaan

[1] Silberschatz *et al.* (2013), *Operating System Concepts*, hlm 163.


What's a thread ?
Julia Evans
@b0rk
drawings.jvns.ca
a process can have lots of threads
process id
thread id
1888
1888
1888
1888
1888
1892
1893
2007
threads share memory
thread 1
I'm going to write "3" to address 2977886
I can overwrite that if I want!
thread 2
but they can run different code
thread 1
I'm doing a calculation!
I'm waiting for a network request to finish!
thread 2
Sharing memory can cause problems
"race conditions"
memory
23
thread 1
I'm going to add 1 to that number!
same time
I'm going to add 1 to that number!
thread 2
RESULT: 24 ← WRONG
(should be 25)
if you have 8 CPU cores, you can run code for 8 threads at the SAME TIME
1 5
2 6
3 7
4 8
CPU cores
SO BUSY


Meanwhile, inside any given Linux system, Mark the Process talks to Rick, his only thread (for now...)
Son, Daddy needs to talk to you. In a few milliseconds, you'll get a brother!
Daddy, do I have to share my stack with him?
12
3321 STACK
No, son. Your stack is all yours. But you do have to share your heap.
Alright, Daddy.
124
321
A few seconds later...
Daddy, Rick is messing up my drawing!
HEAP
101010
11111 0100
0100001
Daniel Stori {turnoff.us}

POSIX Thread

- Linux memakai *thread* standar POSIX[2] (`pthread`)
- saat kompilasi tambahkan *flag* `-pthread`

[2] The Portable Operating System Interface

Membuat Thread

```
pthread_create(thread, attr, func, arg);
```

- membuat satu `thread` dengan atribut `attr` yang akan menjalankan fungsi `func` dengan argumen `arg`[3]
- deklarasi fungsi tersebut:
  - `void *func(void *arg);`[4]

[3] lihat `'man pthread_create'`

[4] `void*`: tipe data *generic pointer*

Menunggu Thread

```
pthread_join(thread, retval);
```

- menunggu `thread` selesai dan menyimpan keluarannya ke variabel `retval`[5]

[5] lihat `'man pthread_join'`

Mengakhiri Thread

```
pthread_exit(retval);
```

- mengakhiri *thread* dengan nilai keluaran `retval`[6]

[6] lihat `'man pthread_exit'`

Contoh

Satu Thread

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

void *hello(void *arg) {
    puts("hello");
    pthread_exit(NULL);
}

int main() {
    pthread_t t;
    pthread_create(&t, NULL, hello, NULL);
    pthread_join(t, NULL);
    return 0;
}
```

Dua Thread

```
...

int main() {
    pthread_t t1, t2;

    pthread_create(&t1, NULL, hello, NULL);
    pthread_create(&t2, NULL, hello, NULL);
    pthread_join(t1, NULL);
    pthread_join(t2, NULL);

    return 0;
}
```

N Thread

```
...

#define N 8

int main() {
    pthread_t t[N];

    for (int i = 0; i < N; i++)
        pthread_create(&t[i], NULL, hello, NULL);
    for (int i = 0; i < N; i++)
        pthread_join(t[i], NULL);

    return 0;
}
```

Contoh Argumen

Satu Thread dengan Argumen

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

void* hello(void* arg) {
    printf("hello from thread %d\n", ((int*)arg)[0]);
    pthread_exit(NULL);
}

int main() {
    pthread_t t;
    int id[] = {1};
    pthread_create(&t, NULL, hello, &id[0]);
    pthread_join(t, NULL);
    return 0;
}
```

Dua Thread dengan Argumen

```
...

int main() {
    pthread_t t1, t2;
    int id[] = {1, 2};

    pthread_create(&t1, NULL, hello, &id[0]);
    pthread_create(&t2, NULL, hello, &id[1]);
    pthread_join(t1, NULL);
    pthread_join(t2, NULL);

    return 0;
}
```

N Thread dengan Argumen

```
...
#define N 8

int main() {
    pthread_t t[N];
    int id[N] = {1, 2, 3, 4, 5, 6, 7, 8};

    for (int i = 0; i < N; i++)
        pthread_create(&t[i], NULL, hello, &id[i]);
    for (int i = 0; i < N; i++)
        pthread_join(t[i], NULL);

    return 0;
}
```

Latihan

Jumlah Array

- lengkapi program berikut untuk menjumlahkan *array* `A`
- gunakan variabel global `sum` untuk menyimpan hasilnya

```
#include <stdio.h>
#define N 16

int sum = 0;

int main() {
    int A[N] = {68,34,64,95,35,78,65,93,
                51,67, 7,77, 4,73,52,91};
    /* TODO: array sum                    */
    printf("%d\n", sum);     /* 954       */
    return 0;
}
```

Jumlah Array (Satu Thread)

- sekarang, buat satu buah *thread* untuk menjumlahkan nilai semua elemen *array* `A` dengan fungsi `array_sum()`
- *thread* utama hanya membuat dan menunggu *thread* ini selesai

Jumlah Array (Dua Thread)

- oke?
- sekarang gunakan 2 buah *thread* untuk menjumlahkan nilai semua elemen *array* A
- pastikan pembagian kerja antara kedua *thread* seimbang, yaitu tiap *thread* memproses $\frac{N}{2}$ elemen

Jumlah Array (Empat Thread)

- bisa?
- sekarang gunakan 4 buah *thread* untuk menjumlahkan nilai semua elemen *array* A
- pastikan pembagian kerja antara keempat *thread* seimbang, yaitu tiap *thread* memproses $\frac{N}{4}$ elemen
- kumpulkan di LMS paling lambat hingga praktikum berakhir

Penilaian

- +60: keluaran benar: `954`
- +20: jumlah *syscall* `clone`[7] ada 4
- +20: variabel *array* A lokal di fungsi utama
- −20: pemanggilan fungsi `pthread` tanpa *looping*

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