Tugas Pertemuan 6
Praktikum Sistem Komputer dan Jaringan

Kosmas Rio Legowo
23/512012/PA/21863
Departemen Ilmu Komputer dan Elektronika
Universitas Gadjah Mada
kosmasriolegowo@mail.ugm.ac.id

Chapter 3 Sinkronisasi Proses

Aktivitas 1: Mensimulasikan Kondisi Balapan

```
kosmasrio_legowo@cloudshell:~ (innate-temple-436001-r1)$ gcc -o race race.c -lpthread
kosmasrio_legowo@cloudshell:~ (innate-temple-436001-r1)$ ./race
Utas 1 membaca nilai dari variabel bersama sebagai: 1
Pembaruan lokal oleh Utas 1: 2
Utas 2 membaca nilai dari variabel bersama sebagai: 1
Pembaruan lokal oleh Utas 2: 0
Nilai dari variabel bersama yang diperbarui oleh Utas 1 adalah: 2
Nilai dari variabel bersama yang diperbarui oleh Utas 2 adalah: 0
Nilai akhir dari variabel bersama adalah 0
```

- Apakah hasil akhir dari variabel bersama mirip dengan teman-temanmu?
  Jawab : Tidak, di beberapa teman saya ada yang mendapatkan nilai akhir variabel bersama bernilai 2.
- Mengapa ini bisa terjadi?
  Jawab : Ini terjadi karena kedua thread berjalan secara bersamaan dengan variabel bersama bernilai 1. Akan tetapi, jika pada kasus thread 1 selesai lebih dahulu, maka nilai akhir dari variabel bersama adalah 0. Mengapa? Karena thread 2 selesai terakhir dan hal ini menyebabkan nilai variabel bersama menyimpan hasil terbaru (dalam hal ini hasil dari thread 2), demikian juga sebaliknya.
- Bagian mana dari daftar kode sumber di atas yang merupakan bagian kritis?
  Jawab : Bagian kritis (critical section) pada source code ini terletak pada baris kode:
  1. x = shared
  2. x++
  3. y = shared
  4. y–

  yang terletak pada fungsi *increment() dan/atau *decrement(). Karena nilai variabel bersama akan digunakan dan diubah pada baris kode di atas.

Aktivitas 2: Sinkronisasi Proses Menggunakan Kunci Mutex

```
kosmasrio_legowo@cloudshell:~ (innate-temple-436001-r1)$ gcc -o mutex mutex.c -lpthread
kosmasrio_legowo@cloudshell:~ (innate-temple-436001-r1)$ ./mutex
Utas 1 mencoba untuk mendapatkan kunci
Utas 1 memperoleh kunci
Utas 1 membaca nilai dari variabel bersama sebagai: 1
Pembaruan lokal oleh Utas 1: 2
Utas 2 mencoba untuk mendapatkan kunci
Nilai dari variabel bersama yang diperbarui oleh Utas 1 adalah: 2
 Utas 1 melepaskan kunci
Utas 2 memperoleh kunci
Utas 2 membaca nilai dari variabel bersama sebagai: 2
Pembaruan lokal oleh Utas 2: 1
Nilai dari variabel bersama yang diperbarui oleh Utas 2 adalah: 1
 Utas 2 melepaskan kunci
Nilai akhir dari variabel bersama adalah 1
```

- Apa nilai akhir dari variabel bersama?
  Jawab : Nilai akhir variabel bersama (shared variable) adalah 1.
- Utas mana yang pertama kali memperoleh kunci?
  Jawab : Sesuai hasil yang saya dapatkan di atas, thread 1 lah yang berhasil mendapatkan lock terlebih dahulu
- Jika utas lain yang pertama kali memperoleh kunci, apa nilai akhir dari variabel bersama?
  Jawab : Jika thread 2 mendapatkan lock terlebih dahulu maka nilai akhir dari variabel bersama akan sama, yaitu 1. Karena jika thread 2 mendapatkan lock dan masuk ke critical section terlebih dahulu, maka variabel bersama akan di-increment dan bernilai 2, hingga akhirnya thread 2 melepaskan lock. Kemudian thread 1 akan mendapatkan lock, kemudian nilai dari variabel bersama akan di-decrement hingga nilainya menjadi 1 dan lock akan dilepas. Maka terbukti bahwa nilai akhir dari variabel bersama akan sama, yaitu 1.

Aktivitas 3: Sinkronisasi Proses Menggunakan Semaphore

```
kosmasrio_legowo@cloudshell:~ (innate-temple-436001-r1)$ gcc -o semaphore semaphore.c -lpthread
kosmasrio_legowo@cloudshell:~ (innate-temple-436001-r1)$ ./semaphore
Utas 1 mencoba untuk mengurangi semaphore
Utas 1 dapat masuk ke bagian kritisnya
Utas 1 membaca nilai dari variabel bersama sebagai: 1
Pembaruan lokal oleh Utas 1: 2
Utas 2 mencoba untuk mengurangi semaphore
Nilai dari variabel bersama yang diperbarui oleh Utas 1 adalah: 2
Utas 1 menambah semaphore
Utas 2 dapat masuk ke bagian kritisnya
Utas 2 membaca nilai dari variabel bersama sebagai: 2
Pembaruan lokal oleh Utas 2: 1
Nilai dari variabel bersama yang diperbarui oleh Utas 2 adalah: 1
Utas 2 menambah semaphore
Nilai akhir dari variabel bersama adalah: 1
```

- Apa nilai akhir dari variabel bersama?
  Jawab : Nilai akhir variabel bersama adalah 1.
- Dari nilai akhir yang Anda peroleh, utas mana yang pertama kali memperoleh akses ke variabel bersama?
  Jawab : Berdasarkan hasil yang saya dapatkan di atas. Terlihat bahwa thread 1 mendapatkan akses ke variabel bersama terlebih dahulu. Akan tetapi, apabila thread 2 mendapatkan akses terlebih dahulu, nilai akhir dari variabel

bersama tetap sama, yaitu 1. Karena konsep dari semaphore sendiri mirip dengan mutex pada activity sebelumnya, dimana semaphore disini bertindak sebagai syarat untuk suatu proses (dalam hal ini thread) untuk dapat masuk ke dalam critical section dari resource yang terbagi (dalam hal ini variabel bersama). Oleh karena itu, thread mana saja yang mendapatkan akses terlebih dahulu tidak akan merubah nilai akhir dari nilai akhir variabel bersama.

Chapter 5 Manajemen Memori

Aktivitas 1: Mengalokasikan Memori Secara Programatik dalam Proses

Awalnya saat di run program tersebut menghasilkan:

```
kosmasrio_legowo@cloudshell:~ (innate-temple-436001-r1)$ gcc -o malloc malloc.c
kosmasrio_legowo@cloudshell:~ (innate-temple-436001-r1)$ ./malloc &
[1] 2677
Memori telah berhasil dialokasikan.
Alamat awal: 0x5627190f12a0
Alamat akhir: 0x5627190f12b0
Elemen dari array adalah:
1
2
3
4
5
```

Kemudian dimodifikasi dengan menambahkan fungsi sleep(10000) hasil running di latar belakang adalah sebagai berikut:

malloc.c > ...

```c
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
int main(){
    int *ptr;

    ptr = (int *) malloc(size: 5 * sizeof(int));
    if (ptr != NULL) {
        printf(format: "Memori telah berhasil dialokasikan.\n");
        printf(format: "Alamat awal: %p\n", ptr);
        printf(format: "Alamat akhir: %p\n", ptr + 4);

        for (int i = 0; i < 5; i++) {
            ptr[i] = i + 1;
        }

        printf(format: "Elemen dari array adalah:\n");
        for (int i = 0; i < 5; i++) {
            printf(format: "%d\n", ptr[i]);
        }

    }
    sleep(seconds: 10000);
}
```

```
kosmasrio_legowo@cloudshell:~ (innate-temple-436001-r1)$ gcc -o malloc malloc.c
kosmasrio_legowo@cloudshell:~ (innate-temple-436001-r1)$ ./malloc &
[1] 2902
Memori telah berhasil dialokasikan.
Alamat awal: 0x5a630d4ed2a0
Alamat akhir: 0x5a630d4ed2b0
Elemen dari array adalah:
1
2
3
4
5
kosmasrio_legowo@cloudshell:~ (innate-temple-436001-r1)$ ps aux | grep ./malloc
kosmasr+    2902  0.0  0.0   2684  1176 pts/2    S<   02:50   0:00 ./malloc
kosmasr+    2906  0.0  0.0   6680  2196 pts/2    S<+  02:51   0:00 grep --color=auto ./malloc
```

```
kosmasrio_legowo@cloudshell:~ (innate-temple-436001-r1)$ cat /proc/2902/maps
5a630bcc4000-5a630bcc5000 r--p 00000000 08:11 131319                     /home/kosmasrio_legowo/malloc
5a630bcc5000-5a630bcc6000 r-xp 00001000 08:11 131319                     /home/kosmasrio_legowo/malloc
5a630bcc6000-5a630bcc7000 r--p 00002000 08:11 131319                     /home/kosmasrio_legowo/malloc
5a630bcc7000-5a630bcc8000 r--p 00002000 08:11 131319                     /home/kosmasrio_legowo/malloc
5a630bcc8000-5a630bcc9000 rw-p 00003000 08:11 131319                     /home/kosmasrio_legowo/malloc
5a630d4ed000-5a630d50e000 rw-p 00000000 00:00 0                          [heap]
7e7aaf824000-7e7aaf827000 rw-p 00000000 00:00 0
7e7aaf827000-7e7aaf84f000 r--p 00000000 08:01 5770702                    /usr/lib/x86_64-linux-gnu/libc.so.6
7e7aaf84f000-7e7aaf9d7000 r-xp 00028000 08:01 5770702                    /usr/lib/x86_64-linux-gnu/libc.so.6
7e7aaf9d7000-7e7aafa26000 r--p 001b0000 08:01 5770702                    /usr/lib/x86_64-linux-gnu/libc.so.6
7e7aafa26000-7e7aafa2a000 r--p 001fe000 08:01 5770702                    /usr/lib/x86_64-linux-gnu/libc.so.6
7e7aafa2a000-7e7aafa2c000 rw-p 00202000 08:01 5770702                    /usr/lib/x86_64-linux-gnu/libc.so.6
7e7aafa2c000-7e7aafa39000 rw-p 00000000 00:00 0
7e7aafa43000-7e7aafa45000 rw-p 00000000 00:00 0
7e7aafa45000-7e7aafa46000 r--p 00000000 08:01 5770681                    /usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2
7e7aafa46000-7e7aafa71000 r-xp 00001000 08:01 5770681                    /usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2
7e7aafa71000-7e7aafa7b000 r--p 0002c000 08:01 5770681                    /usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2
7e7aafa7b000-7e7aafa7d000 r--p 00036000 08:01 5770681                    /usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2
7e7aafa7d000-7e7aafa7f000 rw-p 00038000 08:01 5770681                    /usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2
7ffd156f5000-7ffd15717000 rw-p 00000000 00:00 0                          [stack]
7ffd15728000-7ffd1572c000 r--p 00000000 00:00 0                          [vvar]
7ffd1572c000-7ffd1572e000 r-xp 00000000 00:00 0                          [vdso]
ffffffffff600000-ffffffffff601000 --xp 00000000 00:00 0                  [vsyscall]
```

- Di bagian mana memori pointer menunjuk?
  Jawab : Address pertama dari pointer adalah 0x5a630d4ed2a0 dan menunjuk pada memory dengan address 5a630bcc4000, yang merupakan lokasi dari array dalam memory.
- Apa artinya dalam hal alokasi memori?
  Jawab : Pada kolom offset, terlihat bahwa terdapat jarak sebesar 00001000 dari tiap memorinya. Hal ini menandakan bahwa sistem operasi mengalokasikan ukuran dari tiap memori sekitar 4096 bits.

Aktivitas 2: Mengalokasikan Ulang Memori

Awalnya saat di run program tersebut menghasilkan:

```
kosmasrio_legowo@cloudshell:~ (innate-temple-436001-r1)$ gcc -o realloc realloc.c
kosmasrio_legowo@cloudshell:~ (innate-temple-436001-r1)$ ./realloc
Memori telah berhasil dialokasikan.
Alamat awal: 0x5758fbc572a0
Alamat akhir: 0x5758fbc572b0
Pointer telah berhasil dialokasikan ulang
Elemen dari array adalah:
1
2
3
4
5
6
7
8
9
10
```

Kemudian dimodifikasi dengan mengomentari fungsi realloc dan pencetakan pesannya yang sesuai sehingga menghasilkan sebagai berikut:

realloc.c > main

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

int main() {
    int *ptr;

    ptr = (int *) malloc(size: 5 * sizeof(int));
    if (ptr != NULL) {
        printf(format: "Memori telah berhasil dialokasikan.\n")
        printf(format: "Alamat awal: %p\n", ptr);
        printf(format: "Alamat akhir: %p\n", ptr + 4);

        for (int i = 0; i < 5; i++) {
            ptr[i] = i + 1;
        }
        /*
        ptr = realloc(ptr, 10 * sizeof(int));
        printf("Pointer telah berhasil dialokasikan ulang\n");
        */
        for (int i = 5; i < 10; i++) {
            ptr[i] = i + 1;
        }

        printf(format: "Elemen dari array adalah:\n");
        for (int i = 0; i < 10; i++) {
            printf(format: "%d\n", ptr[i]);
        }
    }
}
```

- Apa hasilnya?
  Jawab : Hasilnya ditunjukkan pada screenshot di bawah ini.

```
kosmasrio_legowo@cloudshell:~ (innate-temple-436001-r1)$ gcc -o realloc realloc.c
kosmasrio_legowo@cloudshell:~ (innate-temple-436001-r1)$ ./realloc
Memori telah berhasil dialokasikan.
Alamat awal: 0x58db434dd2a0
Alamat akhir: 0x58db434dd2b0
Elemen dari array adalah:
1
2
3
4
5
6
7
8
1835338296
842543923
```

- Mengapa itu terjadi?
  Jawab : Ketika kita meng-comment fungsi realloc maka memori (ptr) hanya akan meng alokasi sejumlah 5 * ukuran int, dan memori lain yang tidak di-alokasikan untuk ptr dapat digunakan oleh proses ataupun variabel lain.
  Berdasarkan sumber pada internet, secara default memori yang dialokasikan dari sizeof(int) adalah 8 byte. Hal ini berlaku untuk kelipatan dibawah 8. Namun, apabila kita mengatur memori diatas 8 dan dibawah kelipatan 4 setelah 8 maka kita akan mendapatkan ekstra 4 byte dari tiap kelipatan.
  Oleh karena itu, di awal kita mengatur ukuran dari malloc sebesar 5*sizeof(int) maka kita hanya mengalokasikan sebesar 8 byte saja. Hal ini terbukti pada hasil terminal yang kita dapatkan di atas bahwa 8 nilai awal tersimpan dengan baik sedangkan nilai setelah 8 sudah dipakai oleh proses atau variabel lain.
  Sumber: https://stackoverflow.com/questions/43660109/pointer-size-issue-in-malloc-in-c

Aktivitas 3: Membebaskan Memori

- Apa hasilnya?
  Jawab : Hasilnya ditunjukkan pada screenshot di bawah ini.

```
kosmasrio_legowo@cloudshell:~ (innate-temple-436001-r1)$ gcc -o free free.c
kosmasrio_legowo@cloudshell:~ (innate-temple-436001-r1)$ ./free
Memori telah berhasil dialokasikan.
Alamat awal: 0x59dd9f97a2a0
Alamat akhir: 0x59dd9f97a2b0
Elemen dari array adalah:
1
2
3
4
5
```

- Dapatkah pointer masih digunakan untuk menyimpan dan mencetak nilai? Mengapa?
  Jawab : Pointer masih dapat digunakan untuk menyimpan nilai. Hal ini disebabkan karena pada code kita meng-dealokasikan pointer padahal kita belum mengisi apapun pada pointer tersebut, sehingga tidak terjadi apa-apa. Namun, apabila kita telah mengisi nilai pada pointer kemudian memanggil fungsi free() maka kita tetap dapat menggunakan pointer tersebut, karena fungsi free() ini sendiri adalah untuk mengosongkan nilai pada pointer dan kita tetap dapat menggunakan pointer tersebut seperti biasa.