

Tugas Praktikum SKJ ke-6

Nama : Bagus Cipta Pratama

NIM : 23/516539/PA/22097

Kelas : KOMC

Pembahasan Bab 3 :

A. Aktivitas 1 :

```
bagus-cipta-pratama@bagus:~  
bagus-cipta-pratama@bagus:~$ nano race.c  
bagus-cipta-pratama@bagus:~$ gcc -o race race.c -lpthread  
bagus-cipta-pratama@bagus:~$ ./race  
Utas 1 membaca nilai variabel bersama: 1  
Pembaruan lokal oleh Utas 1: 2  
Utas 2 membaca nilai variabel bersama: 1  
Pembaruan lokal oleh Utas 2: 0  
Nilai variabel bersama diperbarui oleh Utas 1: 2  
Nilai variabel bersama diperbarui oleh Utas 2: 0  
Nilai akhir dari variabel bersama adalah 0  
bagus-cipta-pratama@bagus:~$
```

```
GNU nano 7.2 race.c  
#include <pthread.h>  
#include <stdio.h>  
#include <unistd.h>  
  
void *increment();  
void *decrement();  
  
int shared = 1;  
  
int main() {  
    pthread_t thread1, thread2;  
    pthread_create(&thread1, NULL, increment, NULL);  
    pthread_create(&thread2, NULL, decrement, NULL);  
    pthread_join(thread1, NULL);  
    pthread_join(thread2, NULL);  
  
    printf("Nilai akhir dari variabel bersama adalah %d\n", shared);  
}  
  
void *increment() {  
    int x;  
    x = shared;  
    printf("Utas 1 membaca nilai variabel bersama: %d\n", x);  
    x++;  
    printf("Pembaruan lokal oleh Utas 1: %d\n", x);  
    sleep(1);  
    shared = x;  
    printf("Nilai variabel bersama diperbarui oleh Utas 1: %d\n", shared);  
}  
  
void *decrement() {  
    int y;  
    y = shared;  
    printf("Utas 2 membaca nilai variabel bersama: %d\n", y);  
    y--;  
    printf("Pembaruan lokal oleh Utas 2: %d\n", y);  
    sleep(1);  
}
```

- Apakah hasil akhir dari variable Bersama mirip dengan teman temanmu ?

Hasil akhir dari variable Bersama berbeda dan tidak konsisten setiap kali menjalankan program . ini karena program belum menggunakan mekanisme sinkronisasi yang mencegah kedua thread dari mengakses variable Bersama secara bersamaan , yang menyebabkan ‘race condition’.

- Mengapa ini bisa terjadi ?

Hasil yang berbeda muncul karena dua thread, yaitu increment dan decrement, mengakses dan memodifikasi variabel shared secara bersamaan tanpa adanya pengaturan siapa yang seharusnya mengakses duluan. Dalam kasus ini, kedua thread membaca dan menulis variabel bersama, dan urutan operasi ini tidak dijamin, yang menyebabkan hasil akhirnya tergantung pada urutan eksekusi masing-masing thread

- Bagian mana dari daftar kode sumber diatas yang merupakan bagian kritis ?

Bagian kritis terletak pada bagian Dimana variabel shared dibaca dan ditulis oleh kedua thread . ini terjadi dalam fungsi increment dan decrement

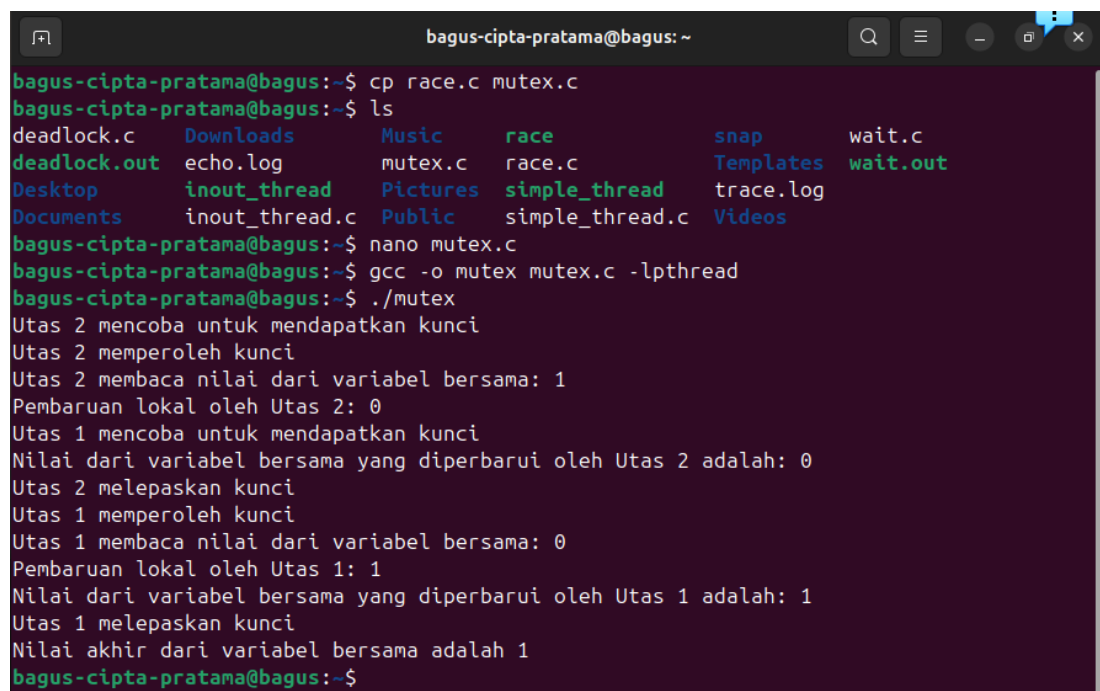
Increment : $x = \text{shared}$; dan $\text{shared} = x$;

Decrement : $y = \text{shared}$; dan $\text{shared} = y$;

B. Aktivitas 2 :

Pertama , saya melakukan penyalinan terhadap kode `rece.c` menjadi `mutex.c` . untuk menjamin hal tersebut saya menggunakan `ls` untuk memverifikasi apakah Salinan file sudah dibuat . setelah itu , saya melakukan modifikasi untuk menggunakan kunci mutex dengan menambahkan `pthread_mutex_t lock` untuk mendefinisikan variabel `mutex` , inisialisasi `mutex` dengan `pthread_mutex_init(&lock,null)` di fungsi `main` dan menggunakan fungsi `pthread_mutex_lock(&lock);` untuk mengunci akses ke bagian kritis dan `unlock` untuk membuka akses setelah selesai .

Berikut adalah kode yang saya jalankan di terminal linux :



```
bagus-cipta-pratama@bagus: ~  
bagus-cipta-pratama@bagus:~$ cp race.c mutex.c  
bagus-cipta-pratama@bagus:~$ ls  
deadlock.c  Downloads  Music  race  snap  wait.c  
deadlock.out  echo.log  mutex.c  race.c  Templates  wait.out  
Desktop  inout_thread  Pictures  simple_thread  trace.log  
Documents  inout_thread.c  Public  simple_thread.c  Videos  
bagus-cipta-pratama@bagus:~$ nano mutex.c  
bagus-cipta-pratama@bagus:~$ gcc -o mutex mutex.c -lpthread  
bagus-cipta-pratama@bagus:~$ ./mutex  
Utas 2 mencoba untuk mendapatkan kunci  
Utas 2 memperoleh kunci  
Utas 2 membaca nilai dari variabel bersama: 1  
Pembaruan lokal oleh Utas 2: 0  
Utas 1 mencoba untuk mendapatkan kunci  
Nilai dari variabel bersama yang diperbarui oleh Utas 2 adalah: 0  
Utas 2 melepaskan kunci  
Utas 1 memperoleh kunci  
Utas 1 membaca nilai dari variabel bersama: 0  
Pembaruan lokal oleh Utas 1: 1  
Nilai dari variabel bersama yang diperbarui oleh Utas 1 adalah: 1  
Utas 1 melepaskan kunci  
Nilai akhir dari variabel bersama adalah 1  
bagus-cipta-pratama@bagus:~$
```

```
bagus-cipta-pratama@bagus: ~  
GNU nano 7.2 mutex.c *  
#include <pthread.h>  
#include <stdio.h>  
#include <unistd.h>  
  
void *increment();  
void *decrement();  
  
int shared = 1;  
pthread_mutex_t lock;  
  
int main() {  
    pthread_t thread1, thread2;  
  
    pthread_mutex_init(&lock, NULL);  
  
    pthread_create(&thread1, NULL, increment, NULL);  
    pthread_create(&thread2, NULL, decrement, NULL);  
  
    pthread_join(thread1, NULL);  
    pthread_join(thread2, NULL);  
  
    printf("Nilai akhir dari variabel bersama adalah %d\n", shared);  
  
    pthread_mutex_destroy(&lock);  
}  
  
void *increment() {  
    int x;  
  
    printf("Utas 1 mencoba untuk mendapatkan kunci\n");  
    pthread_mutex_lock(&lock);  
    printf("Utas 1 memperoleh kunci\n");  
  
    x = shared;  
    printf("Utas 1 membaca nilai dari variabel bersama: %d\n", x);  
    x++;  
    printf("Pembaruan lokal oleh Utas 1: %d\n", x);
```

```
    x = shared;  
    printf("Utas 1 membaca nilai dari variabel bersama: %d\n", x);  
    x++;  
    printf("Pembaruan lokal oleh Utas 1: %d\n", x);  
    sleep(1);  
  
    shared = x;  
    printf("Nilai dari variabel bersama yang diperbarui oleh Utas 1 adalah: %d\n", shared);  
  
    pthread_mutex_unlock(&lock);  
    printf("Utas 1 melepaskan kunci\n");  
}  
  
void *decrement() {  
    int y;  
  
    printf("Utas 2 mencoba untuk mendapatkan kunci\n");  
    pthread_mutex_lock(&lock);  
    printf("Utas 2 memperoleh kunci\n");  
  
    y = shared;  
    printf("Utas 2 membaca nilai dari variabel bersama: %d\n", y);  
    y--;  
    printf("Pembaruan lokal oleh Utas 2: %d\n", y);  
    sleep(1);  
  
    shared = y;  
    printf("Nilai dari variabel bersama yang diperbarui oleh Utas 2 adalah: %d\n", shared);  
  
    pthread_mutex_unlock(&lock);  
    printf("Utas 2 melepaskan kunci\n");  
}
```

- Apa nilai akhir dari variabel Bersama ?

Nilai akhir dari variabel bersama adalah 1. Hal ini menunjukkan bahwa kedua thread berhasil mengakses variabel bersama secara sinkron menggunakan mutex, sehingga tidak ada 'Race condition' yang menyebabkan nilai menjadi tidak konsisten.

- Utas mana yang pertama kali memperoleh kunci?

Dari screenshot diatas dapat dilihat bahwa utas 2 adalah thread yang pertama kali memperoleh kunci . ini terlihat dari pesan 'utas 2 memperoleh kunci' yang muncul sebelum pesan dari utas 1.

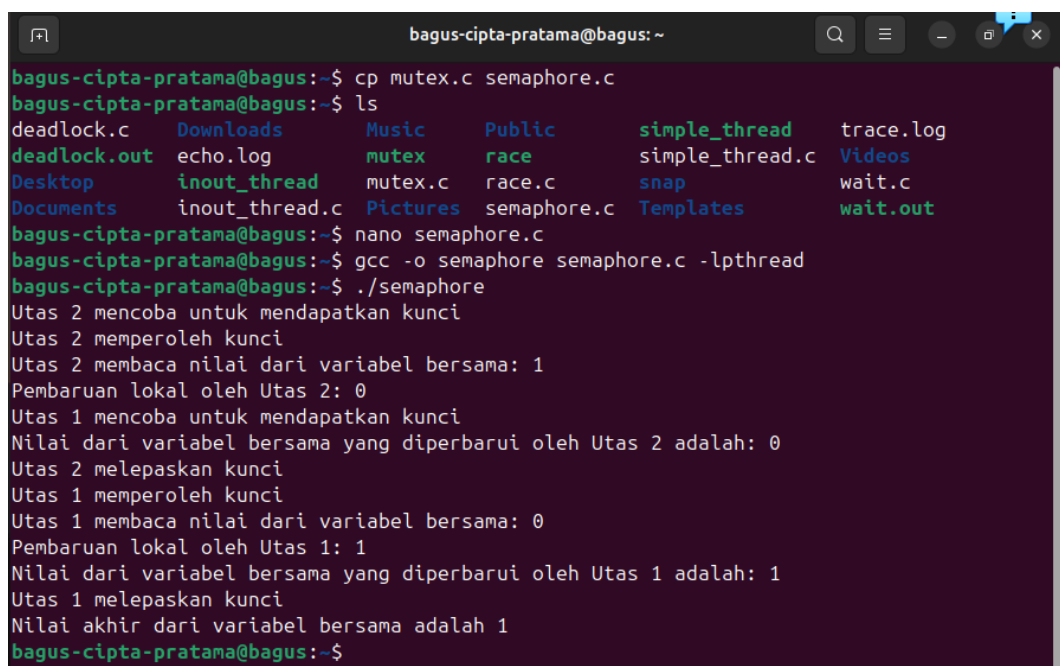
- Jika utas lain yang pertama kali memperoleh kunci , apa nilai akhir dari variabel Bersama ?

Jika utas 1 yang pertama kali memperoleh kunci , nilai akhir dari variabel Bersama tetap akan menjadi 1 . ini karena mutex memastikan bahwa hanya satu thread yang mengakses variabel Bersama pada satu waktu . jadi , urutan eksekusi thread tidak akan mempengaruhi hasil akhir , dan 'race condition' bisa dihindari .

C. Aktivitas 3 :

Tujuan dari aktivitas ketiga adalah menggunakan semaphore untuk mengontrol akses ke variabel Bersama sehingga hanya satu utas yang dapat mengakses variabel pada satu waktu , sekaligus mencegah ‘race condition’.

Pertama , saya menyalin kode mutex.c menjadi semaphore.c , lalu saya memodifikasi kode untuk bisa menggunakan semaphore . disini ditambahkan package `#include<semaphore.h>` untuk menggunakannya . saya mendeklarasikan `sem_t semaphore` dan `sem_init()` dan menggunakan `sem_wait()` untuk menggantikan fungsi mutex sebelumnya dan `sem_post()` untuk sebaliknya juga . terakhir digunakan `sem_destroy()` untuk menghancurkan semaphore setelah program selesai .



```
bagus-cipta-pratama@bagus: ~  
bagus-cipta-pratama@bagus:~$ cp mutex.c semaphore.c  
bagus-cipta-pratama@bagus:~$ ls  
deadlock.c  Downloads  Music  Public  simple_thread  trace.log  
deadlock.out echo.log  mutex  race  simple_thread.c Videos  
Desktop  inout_thread  mutex.c  race.c  snap  wait.c  
Documents  inout_thread.c  Pictures  semaphore.c  Templates  wait.out  
bagus-cipta-pratama@bagus:~$ nano semaphore.c  
bagus-cipta-pratama@bagus:~$ gcc -o semaphore semaphore.c -lpthread  
bagus-cipta-pratama@bagus:~$ ./semaphore  
Utas 2 mencoba untuk mendapatkan kunci  
Utas 2 memperoleh kunci  
Utas 2 membaca nilai dari variabel bersama: 1  
Pembaruan lokal oleh Utas 2: 0  
Utas 1 mencoba untuk mendapatkan kunci  
Nilai dari variabel bersama yang diperbarui oleh Utas 2 adalah: 0  
Utas 2 melepaskan kunci  
Utas 1 memperoleh kunci  
Utas 1 membaca nilai dari variabel bersama: 0  
Pembaruan lokal oleh Utas 1: 1  
Nilai dari variabel bersama yang diperbarui oleh Utas 1 adalah: 1  
Utas 1 melepaskan kunci  
Nilai akhir dari variabel bersama adalah 1  
bagus-cipta-pratama@bagus:~$
```

```
bagus-cipta-pratama@bagus: ~
GNU nano 7.2 semaphore.c *
#include <pthread.h>
#include <stdio.h>
#include <unistd.h>
#include <semaphore.h>

void *increment();
void *decrement();

int shared = 1;
sem_t semaphore;

int main() {
    pthread_t thread1, thread2;

    sem_init(&semaphore, 0, 1);

    pthread_create(&thread1, NULL, increment, NULL);
    pthread_create(&thread2, NULL, decrement, NULL);

    pthread_join(thread1, NULL);
    pthread_join(thread2, NULL);

    printf("Nilai akhir dari variabel bersama adalah %d\n", shared);

    sem_destroy(&semaphore);
}

void *increment() {
    int x;

    printf("Utas 1 mencoba untuk mendapatkan kunci\n");
    sem_wait(&semaphore);
    printf("Utas 1 memperoleh kunci\n");

    x = shared;
    printf("Utas 1 membaca nilai dari variabel bersama: %d\n", x);
    x++;
}
```

```
bagus-cipta-pratama@bagus: ~
GNU nano 7.2 semaphore.c *
    printf("Utas 1 mencoba untuk mendapatkan kunci\n");
    sem_wait(&semaphore);
    printf("Utas 1 memperoleh kunci\n");

    x = shared;
    printf("Utas 1 membaca nilai dari variabel bersama: %d\n", x);
    x++;
    printf("Pembaruan lokal oleh Utas 1: %d\n", x);
    sleep(1);

    shared = x;
    printf("Nilai dari variabel bersama yang diperbarui oleh Utas 1 adalah: %d\n", shared);

    sem_post(&semaphore);
    printf("Utas 1 melepaskan kunci\n");
}

void *decrement() {
    int y;

    printf("Utas 2 mencoba untuk mendapatkan kunci\n");
    sem_wait(&semaphore);
    printf("Utas 2 memperoleh kunci\n");

    y = shared;
    printf("Utas 2 membaca nilai dari variabel bersama: %d\n", y);
    y--;
    printf("Pembaruan lokal oleh Utas 2: %d\n", y);
    sleep(1);

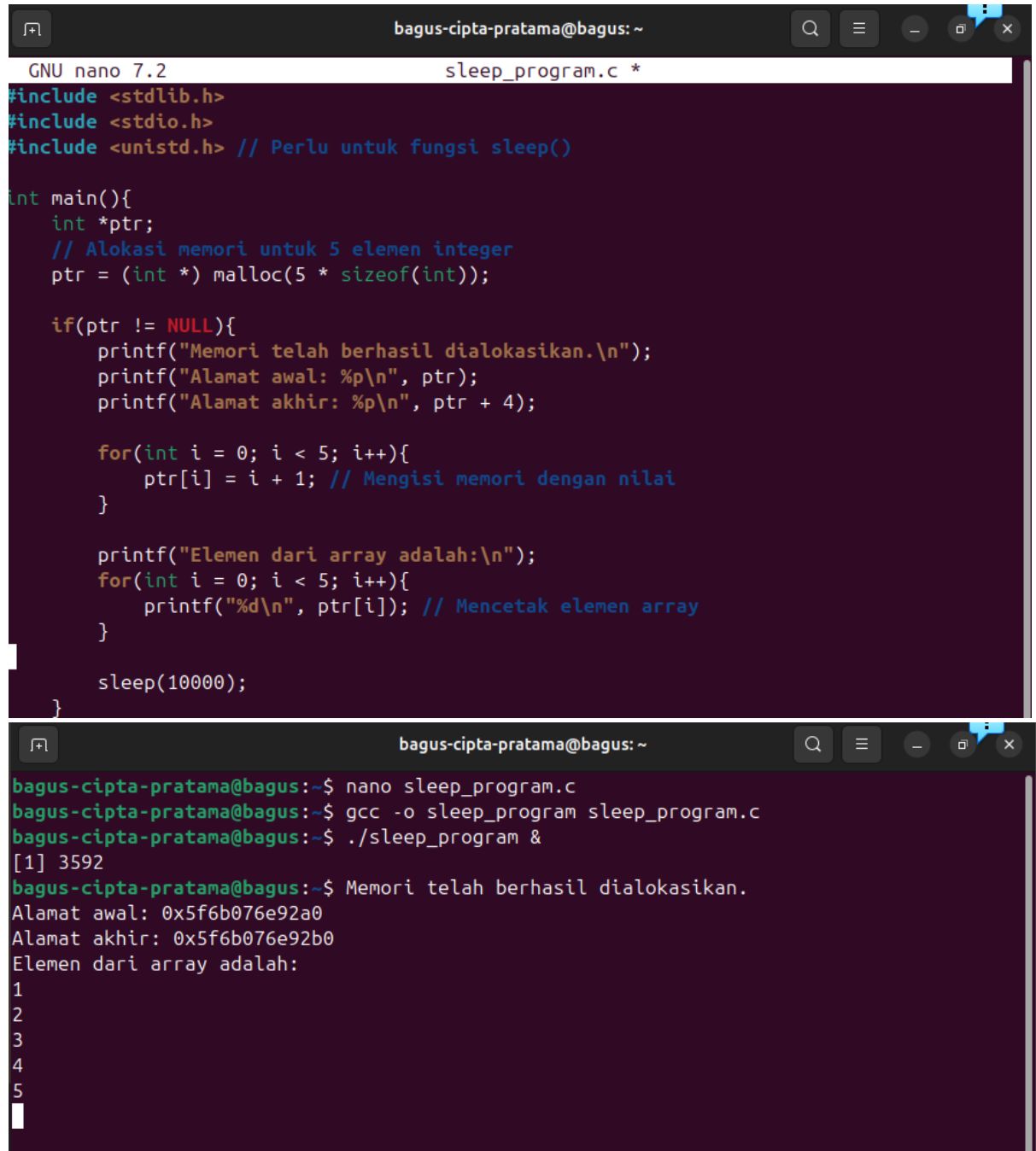
    shared = y;
    printf("Nilai dari variabel bersama yang diperbarui oleh Utas 2 adalah: %d\n", shared);

    sem_post(&semaphore);
    printf("Utas 2 melepaskan kunci\n");
}
```


- Apa nilai akhir dari variabel Bersama ?
Nilai akhir dari bvariabel Bersama adalah 1. Semaphore berhasil mengatur akses thread sehingga tidak ada 'race condition' , dan nilai variabel Bersama diperbarui secara sinkron oleh kedua utas .
- Thread mana yang pertama kali memperoleh semaphore ?
Thread / utas 2 adalah yang pertama kali memperoleh semaphore . ini terlihat dari output pada 'utas 2 memperoleh kunci' yang muncul sebelum pesan dari utas 1.
- Jika thread lain yang pertama kali memperoleh semaphore, apa nilai akhir dari variabel bersama?
Jika Utas 1 yang pertama kali memperoleh semaphore, nilai akhir dari variabel bersama tetap akan menjadi 1. Hal ini karena semaphore memastikan bahwa hanya satu thread yang bisa mengakses variabel bersama pada satu waktu, sehingga urutan eksekusi tidak mempengaruhi hasil akhir.

Pembahasan Bab 5 :

A. Aktivitas 1 :



```
bagus-cipta-pratama@bagus: ~
GNU nano 7.2 sleep_program.c *
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h> // Perlu untuk fungsi sleep()

int main(){
    int *ptr;
    // Alokasi memori untuk 5 elemen integer
    ptr = (int *) malloc(5 * sizeof(int));

    if(ptr != NULL){
        printf("Memori telah berhasil dialokasikan.\n");
        printf("Alamat awal: %p\n", ptr);
        printf("Alamat akhir: %p\n", ptr + 4);

        for(int i = 0; i < 5; i++){
            ptr[i] = i + 1; // Mengisi memori dengan nilai
        }

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

        sleep(10000);
    }
}
```

```
bagus-cipta-pratama@bagus: ~
bagus-cipta-pratama@bagus:~$ nano sleep_program.c
bagus-cipta-pratama@bagus:~$ gcc -o sleep_program sleep_program.c
bagus-cipta-pratama@bagus:~$ ./sleep_program &
[1] 3592
bagus-cipta-pratama@bagus:~$ Memori telah berhasil dialokasikan.
Alamat awal: 0x5f6b076e92a0
Alamat akhir: 0x5f6b076e92b0
Elemen dari array adalah:
1
2
3
4
5
```

```

bagus-cipta-pratama@bagus:~$ cat /proc/3592/maps
5f6b05b96000-5f6b05b97000 r--p 00000000 08:02 2098261 /home/bagus-cipta-pratama/sleep_program
5f6b05b97000-5f6b05b98000 r-xp 00001000 08:02 2098261 /home/bagus-cipta-pratama/sleep_program
5f6b05b98000-5f6b05b99000 r--p 00002000 08:02 2098261 /home/bagus-cipta-pratama/sleep_program
5f6b05b99000-5f6b05b9a000 r--p 00002000 08:02 2098261 /home/bagus-cipta-pratama/sleep_program
5f6b05b9a000-5f6b05b9b000 rw-p 00003000 08:02 2098261 /home/bagus-cipta-pratama/sleep_program
5f6b076e9000-5f6b0770a000 rw-p 00000000 00:00 0 [heap]
7f87f1a00000-7f87f1a28000 r--p 00000000 08:02 2370691 /usr/lib/x86_64-linux-gnu/libc.so.6
7f87f1a28000-7f87f1bb0000 r-xp 00028000 08:02 2370691 /usr/lib/x86_64-linux-gnu/libc.so.6
7f87f1bb0000-7f87f1bfb000 r--p 001b0000 08:02 2370691 /usr/lib/x86_64-linux-gnu/libc.so.6
7f87f1bfb000-7f87f1c03000 r--p 001fe000 08:02 2370691 /usr/lib/x86_64-linux-gnu/libc.so.6
7f87f1c03000-7f87f1c05000 rw-p 00202000 08:02 2370691 /usr/lib/x86_64-linux-gnu/libc.so.6
7f87f1c05000-7f87f1c12000 rw-p 00000000 00:00 0
7f87f1c3a000-7f87f1c3d000 rw-p 00000000 00:00 0
7f87f1c4d000-7f87f1c4f000 rw-p 00000000 00:00 0
7f87f1c4f000-7f87f1c50000 r--p 00000000 08:02 2370509 /usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2
7f87f1c50000-7f87f1c7b000 r-xp 00001000 08:02 2370509 /usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2
7f87f1c7b000-7f87f1c85000 r--p 0002c000 08:02 2370509 /usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2
7f87f1c85000-7f87f1c87000 r--p 00036000 08:02 2370509 /usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2
7f87f1c87000-7f87f1c89000 rw-p 00038000 08:02 2370509 /usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2
7ffe645a7000-7ffe645c8000 rw-p 00000000 00:00 0 [stack]
7ffe645cc000-7ffe645d0000 r--p 00000000 00:00 0 [vvar]
7ffe645d0000-7ffe645d2000 r-xp 00000000 00:00 0 [vdso]
ffffffff600000-ffffffff601000 --xp 00000000 00:00 0 [vsyscall]

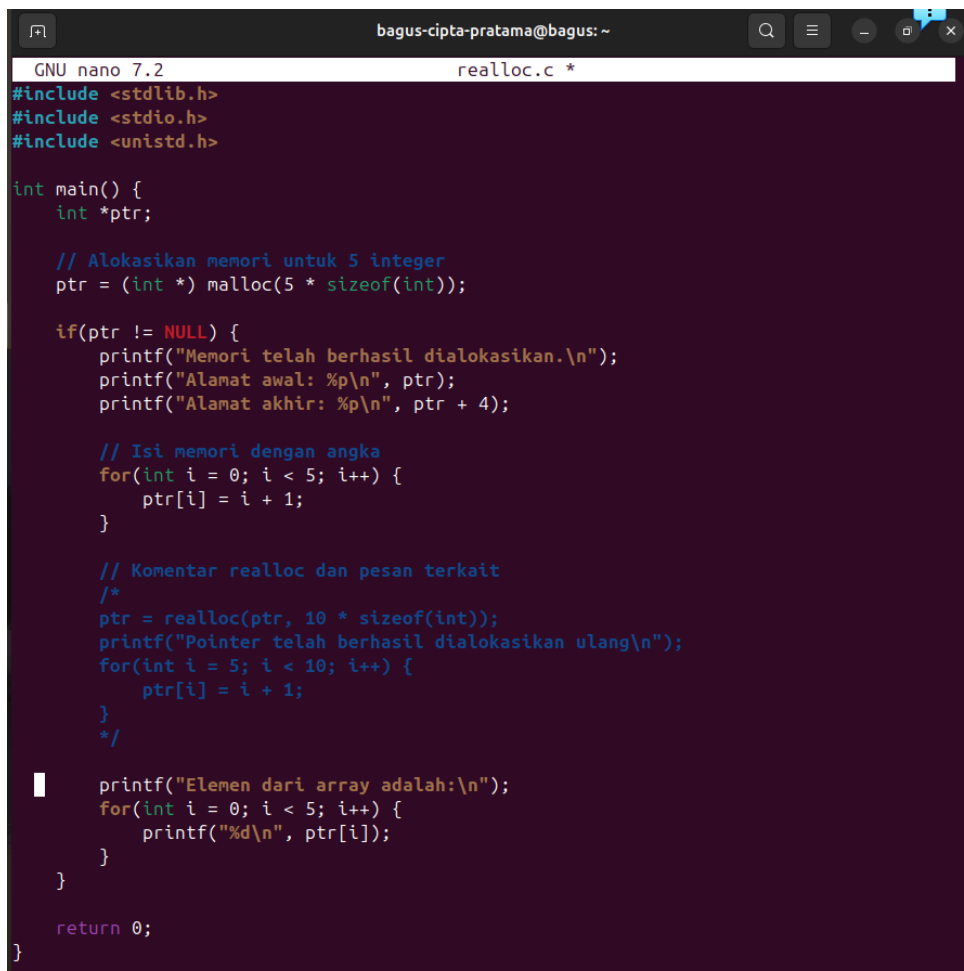
```

- Di bagian mana memori pointer menunjuk ?
 Alamat awal pointer yang ditunjukkan dalam output adalah 0x5f6b076e92a0, dan alamat akhir adalah 0x5f6b076e92b0. Ini menunjukkan bahwa memori dialokasikan secara dinamis oleh malloc() pada segmen heap. Segmen heap adalah bagian dari memori yang digunakan untuk alokasi memori dinamis dalam sebuah proses, di mana memori dialokasikan dan dibebaskan secara manual (misalnya menggunakan malloc() dan free() dalam C).
- Apa artinya dalam hal alokasi memori ?
 alamat memori yang dialokasikan (0x5f6b076e92a0 hingga 0x5f6b076e92b0), ini menunjukkan bahwa alokasi memori telah berhasil dilakukan di segmen heap. Dalam alokasi dinamis, program meminta sistem operasi untuk menyediakan sejumlah blok memori yang dapat digunakan, dan memori ini dialokasikan dari segmen heap. Memori ini bersifat fleksibel, bisa bertambah atau berkurang selama masa

eksekusi program. Memori yang dialokasikan berada dalam segmen heap, yang umumnya berada di atas segmen data statis dan dapat tumbuh saat program memerlukan lebih banyak memori (selama masih ada ruang di memori fisik atau virtual).

B. Aktivitas 2 :

pada aktivitas 2 , kita melakukan ini dengan tujuan memahami pengalokasian ulang memori dinamis menggunakan fungsi `realloc()` dalam bahasa c . secara singkat , aktivitas ini bertujuan untuk memberikan pemahaman bagaimana memori dialokasikan , dikelola dan dialokasikan ulang .



```
bagus-cipta-pratama@bagus: ~  
GNU nano 7.2      realloc.c *  
#include <stdlib.h>  
#include <stdio.h>  
#include <unistd.h>  
  
int main() {  
    int *ptr;  
  
    // Alokasikan memori untuk 5 integer  
    ptr = (int *) malloc(5 * sizeof(int));  
  
    if(ptr != NULL) {  
        printf("Memori telah berhasil dialokasikan.\n");  
        printf("Alamat awal: %p\n", ptr);  
        printf("Alamat akhir: %p\n", ptr + 4);  
  
        // Isi memori dengan angka  
        for(int i = 0; i < 5; i++) {  
            ptr[i] = i + 1;  
        }  
  
        // Komentar realloc dan pesan terkait  
        /*  
        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("Elemen dari array adalah:\n");  
        for(int i = 0; i < 5; i++) {  
            printf("%d\n", ptr[i]);  
        }  
    }  
  
    return 0;  
}
```

```
bagus-cipta-pratama@bagus: ~  
bagus-cipta-pratama@bagus:~$ nano realloc.c  
bagus-cipta-pratama@bagus:~$ gcc realloc.c -o realloc  
bagus-cipta-pratama@bagus:~$ ./realloc  
Memori telah berhasil dialokasikan.  
Alamat awal: 0x6038400942a0  
Alamat akhir: 0x6038400942b0  
Elemen dari array adalah:  
1  
2  
3  
4  
5  
bagus-cipta-pratama@bagus:~$
```

Penjelasan :

1. Apa hasil yang diperoleh setelah realloc dinonaktifkan ?

Output program menunjukkan bahwa memori telah berhasil dialokasikan untuk 5 elemen awal, yaitu angka dari 1 hingga 5. Memori dialokasikan dengan alamat awal 0x6038400942a0 dan alamat akhir 0x6038400942b0. Namun, karena fungsi realloc() telah dinonaktifkan, hanya 5 elemen pertama yang diisi dan dicetak.

2. Mengapa hasil tersebut terjadi ?

Dalam program ini, karena fungsi realloc() telah dikomentari atau dinonaktifkan, hanya alokasi awal memori untuk 5 elemen yang dilakukan. Memori tidak dialokasikan ulang untuk menambah kapasitas array dari 5 menjadi 10 elemen. Akibatnya, hanya 5 elemen awal yang dicetak, dan elemen tambahan (yang akan dialokasikan jika realloc() diaktifkan) tidak ada.

C. Aktivitas 3 :

```
bagus-cipta-pratama@bagus: ~  
bagus-cipta-pratama@bagus:~$ nano free.c  
bagus-cipta-pratama@bagus:~$ gcc -o free_program free.c  
bagus-cipta-pratama@bagus:~$ ./free_program  
Memori telah berhasil dialokasikan.  
Alamat awal: 0x5e69aa2372a0  
Alamat akhir: 0x5e69aa2372b0  
Elemen dari array adalah:  
1  
2  
3  
4  
5  
bagus-cipta-pratama@bagus:~$
```

```
GNU nano 7.2 free.c  
#include <stdlib.h>  
#include <stdio.h>  
  
int main(){  
    int *ptr;  
    ptr = (int *) malloc (5 * sizeof(int));  
  
    if(ptr != NULL){  
        printf("Memori telah berhasil dialokasikan.\n");  
        printf("Alamat awal: %p\n", ptr);  
        printf("Alamat akhir: %p\n", ptr +4);  
  
        free(ptr);  
  
        for(int i=0; i<5; i++){  
            ptr[i] = i+1;  
        }  
  
        printf("Elemen dari array adalah:\n");  
        for(int i=0; i<5; i++){  
            printf("%d\n",ptr[i]);  
        }  
    }  
}
```

jawaban pertanyaan :

- Apakah pointer masih bisa digunakan untuk menyimpan dan mencetak nilai setelah dipanggil free()?

Berdasarkan output diatas , Setelah ``free()`` dipanggil, pointer masih bisa digunakan untuk menyimpan dan mencetak nilai, seperti terlihat pada output. Namun, ini adalah undefined behavior dalam C, dan tidak aman. Meskipun berhasil sekarang, akses ke memori yang sudah dibebaskan bisa menyebabkan masalah di masa depan.