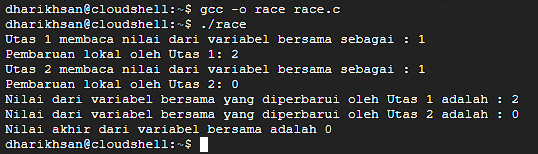
Nama: Daffa Harikhsan

NIM: 23/513044/PA/21918

**Tugas 6**

3.3.1 Activity 1





- Is the final result of the shared variable similiar to your friends?

Ya, hasilnya mirip dengan teman-temanku, variabel bersamanya = 0 di akhir

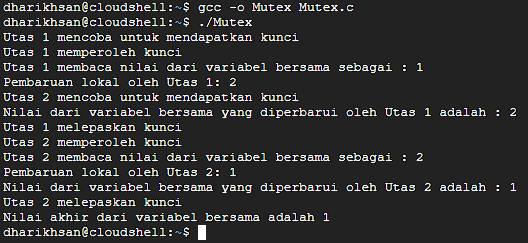
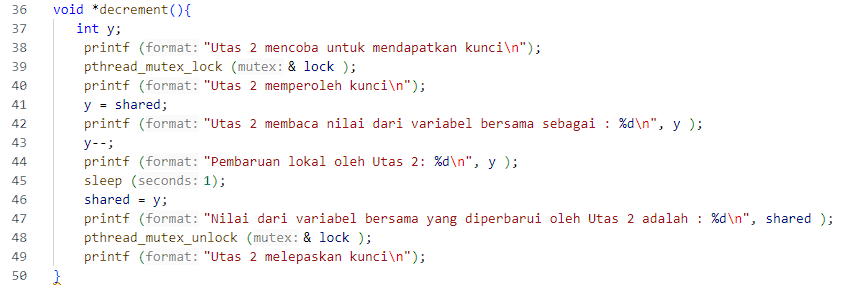
- Why does this happen?

Hal ini terjadi karena masalah race condition. Kedua thread, yaitu increment dan decrement, berjalan secara bersamaan tanpa sinkronisasi yang tepat, sehingga keduanya membaca dan menulis ke variabel shared secara bersamaan. Akibatnya, nilai shared bisa berubah dengan cara yang tidak diinginkan karena satu thread mungkin menimpa perubahan yang dilakukan oleh thread lainnya yang menyebabkan nilai shared = 0.

- Which part of the source code list above is the critical section?

Bagian yang menjadi critical section adalah blok kode yang mengakses dan memodifikasi variabel shared. Pada fungsi increment, bagian yang menjadi critical section adalah: (x = shared; setelah itu x++; dan yang terakhir shared = x), Thread ini membaca nilai dari shared, memodifikasi, dan menulis kembali nilainya. Sedangkan pada fungsi decrement, bagian yang menjadi critical section adalah: (y = shared; setelah itu y--; dan shared = y), thread ini membaca nilai shared, memodifikasi, dan menulis kembali.

3.4.1 Activity 2



- What is the final value of the shared variable?

Nilai akhir dari variable bersama yaitu = 1.

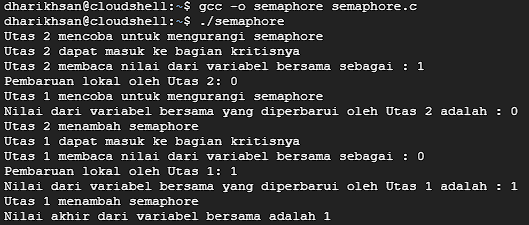
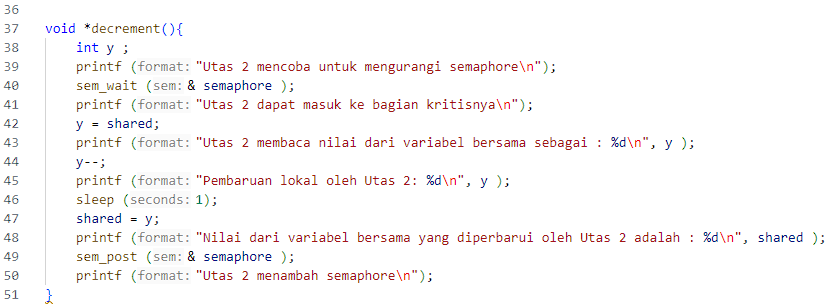
- Which thread obtains the lock first?

Thread yang memperoleh kunci terlebih dahulu adalah thread 1, karena ada pada bagian output.

- If the other thread first obtains the lock, what will be the final value of the shared variable?

Jika thread yang lain yang menerima/memperoleh kunci terlebih dahulu, maka urutan eksekusinya akan berbeda, dan nilai akhir dari variable bersama tetap 1 tidak berubah. Ini terjadi karena kedua thread melakukan modifikasi pada variable shared dengan menambahkan atau mengurangi nilai yang sama (satu increment, dan satu decrement).

3.5.1 Activity 3



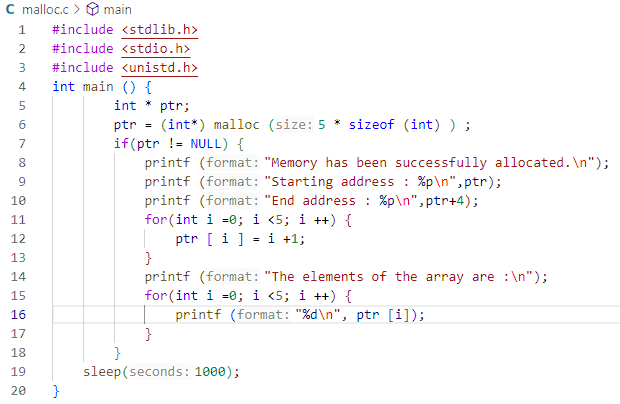
- What is the final value of the shared variable?

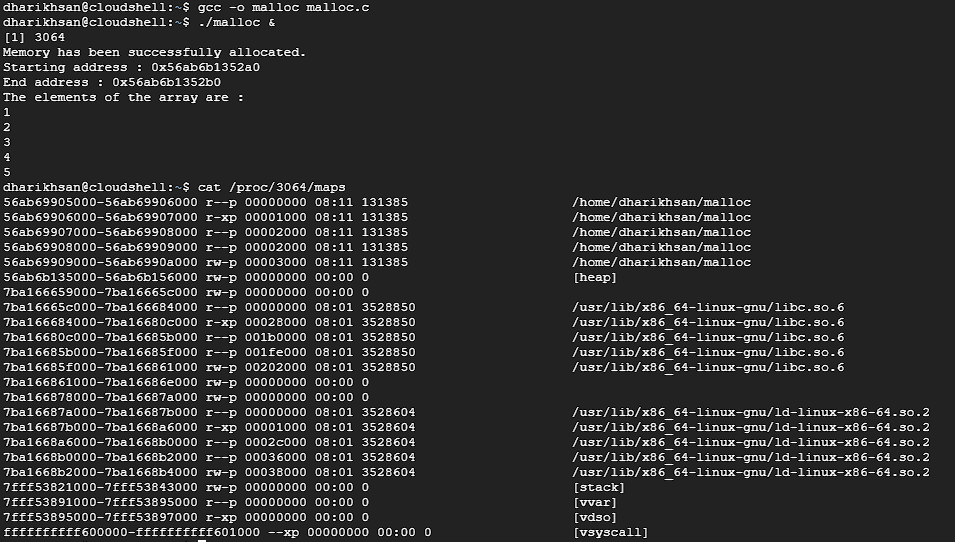
Nilai akhir dari variable bersama adalah 1.

- From the final value that you obtained, which thread first obtains access to the shared variable?

Thread 2 memperoleh pertama kali mengakses ke variabel bersama, setelah itu thread 2 melakukan decrement pada variable bersama menjadi 0, lalu thread 1 melakukan increment kembali sehingga nilai akhir variabel bersama adalah 1.

5.5.1 Activity 1





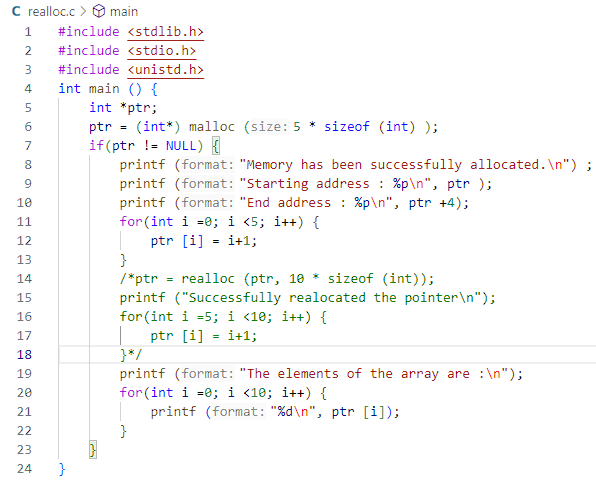
- In what part of the memory is the pointer pointed to?

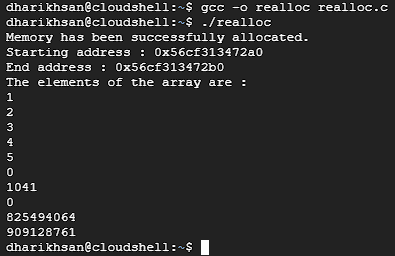
Pointer menunjuk ke alamat awal dari blok memori yang dialokasikan yakni 0x55aeb915a520. Alokasi memori terjadi pada alamat yang dimulai dari 0x55aeb915a520 dan berakhir di alamat 0x55aeb915a530. Pointer menunjuk ke blok memori yang dialokasikan secara dinamis menggunakan fungsi malloc (malloc()).

- What does it mean in terms of memory allocation?

Dalam program ini penggunaan fungsi malloc digunakan untuk mengalokasikan memori dinamis sebesar 5 elemen integer, serta menampilkan alamat awal dan akhir dari blok memori yang dialokasikan. Setelah dialokasikan, program mengisi array dengan nilai 1 hingga 5 dan menampilkannya

5.6.1 Activity 2





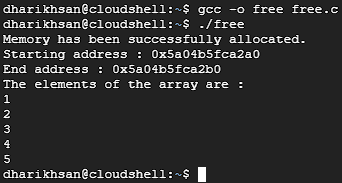
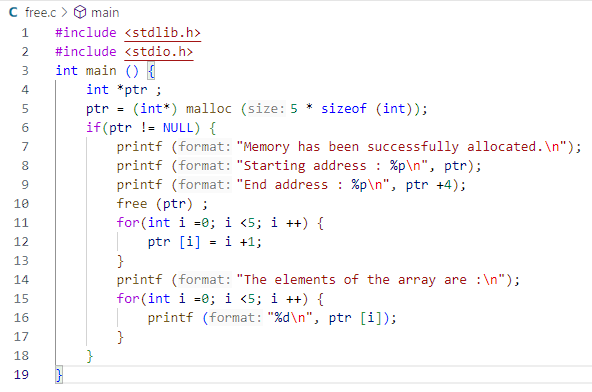
- What is the result?

Hasil program tersebut setelah dijalankan menampilkan 5 elemen array pertama dengan lancar, tetapi untuk 5 elemen berikutnya menunjukkan angka yang acak. Ini terjadi karena Alokasi memori awal berhasil, namun kode untuk alokasi ulang menggunakan realloc dikomentari sehingga tidak dijalankan.

- Why does that happen?

Yang terjadi karena meskipun malloc mengalokasikan memori untuk 5 elemen, program mencoba mencetak 10 elemen. Karena kode untuk alokasi ulang memori (realloc) dikomentari, hanya 5 elemen yang benar-benar dialokasikan, dan elemen di luar rentang tersebut berisi nilai acak dari memori yang tidak terinisialisasi atau tidak dialokasikan.

5.7.1 Activity



- What is the result?

Program menunjukkan bahwa memori dialokasikan untuk 5 elemen integer, dan alokasi memori tersebut berhasil. Setelah itu, memori dibebaskan dengan fungsi free(). Namun, program tetap berusaha untuk mengisi dan mencetak nilai dari pointer yang sudah dibebaskan. Dan juga hasilnya tetap sesuai dengan nilai yang diharapkan

- Can the pointer still be used to store and print values? Why?

Pointer tersebut tidak boleh digunakan untuk menyimpan dan mencetak nilai setelah memori dibebaskan dengan free(). Secara teknis, setelah memori dibebaskan, akses ke pointer tersebut adalah undefined behavior, artinya program bisa menampilkan hasil yang tidak terduga, crash, atau (seperti dalam kasus ini) kebetulan masih berfungsi dengan baik karena memori tersebut belum ditimpa. Namun, ini bukanlah praktik yang aman atau benar.