Tugas 13
Sistem Operasi

Nama : Rheza Dewangga Rendragraha

Kelas: 1 D4 Teknik Informatika B

NRP: 2110191044

OPERATING SYSTEMS - THREADS

1. Pthread

a. Source Code

```
1.  #include <stdio.h>
2.  #include <stdlib.h>
3.  #include <pthread.h>
4.
5.  void *print_message_function( void *ptr );
6.
7.  main()
8.  {
9.       pthread_t thread1, thread2;
10.      char *message1 = "Thread 1";
11.      char *message2 = "Thread 2";
12.      int  iret1, iret2;
13.
14.     /* Create independent threads each of which will execute function */
15.
16.      iret1 = pthread_create( &thread1, NULL, print_message_function, (void*) messag
    e1);
17.      iret2 = pthread_create( &thread2, NULL, print_message_function, (void*) messag
    e2);
18.
19.      /* Wait till threads are complete before main continues. Unless we  */
20.      /* wait we run the risk of executing an exit which will terminate   */
21.      /* the process and all threads before the threads have completed.   */
22.
23.      pthread_join( thread1, NULL);
24.      pthread_join( thread2, NULL);
25.
26.      printf("Thread 1 returns: %d\n",iret1);
27.      printf("Thread 2 returns: %d\n",iret2);
28.      exit(0);
29. }
30.
31. void *print_message_function( void *ptr )
32. {
33.      char *message;
34.      message = (char *) ptr;
35.      printf("%s \n", message);
36. }
```

b. Output

```
rheza@rheza-VirtualBox:~$ gcc -o pthread1 pthread1.c -lpthread
rheza@rheza-VirtualBox:~$ ./pthread1
Thread 2
Thread 1
Thread 1 returns: 0
Thread 2 returns: 0
```

c. Analisa (list angka menunjukkan baris yang di maksud pada source code)

1) file header <stdio.h> digunakan dideklarasikan untuk mengakses berbagai fungsi standar input dan output.
2) file header <stdlib.h> digunakan untuk operasi pembanding dan operasi konversi.

3) file header <pthread.h> digunakan untuk mendefinisikan beberapa tipe data, fungsi, dan konstanta dari C untuk thread, semisal pada program ini dibutuhkan tipe pthread_t, dan misal fungsi pthread_create dan pthread_join.
4) -
5) deklarasi fungsi *print_message_function(void *ptr); pada fungsi print message mempunyai parameter pointer *ptr yang bertipe void.
6) -
7) fungsi main, yaitu fungsi utama dalam program. Fungsi ini akan dieksekusi pertama kali saat program dijalankan.
8) -
9) Deklarasi variabel thread1 dan thread 2 yang bertipe data pthread_t.
10) deklarasi pointer *message1 yang bertipe char dan diassign dengan "Thread 1".
11) deklarasi pointer *message2 yang bertipe char dan diassign dengan "Thread 2".
12) deklarasi pointer iret1 dan iret2 yang bertipe int.
13) -
14) -
15) -
16) pthread_create adalah sebuah fungsi yang digunakan untuk menciptakan thread baru dalam suatu proses dengan atribut yang ditentukan. Fungsi ini mempunyai 4 parameter. Pada parameter pertama, jika pemanggilan fungsi ini berhasil, fungsi akan menyimpan ID dari thread ciptaan pada rujukan yang ditunjuk thread yaitu variable thread1, pada parameter kedua diisi NULL apabila variabel yang digunakan merupakan variable default. Parameter yang ketiga digunakan untuk menunjuk fungsi yang akan dikerjakan oleh thread. Pada parameter keempat, digunakan untuk mengarahkan ke fungsi yang akan di threaded yaitu message1 (pointer to void).
17) pthread_create adalah sebuah fungsi yang digunakan untuk menciptakan thread baru dalam suatu proses dengan atribut yang ditentukan. Fungsi ini mempunyai 4 parameter. Pada parameter pertama, jika pemanggilan fungsi ini berhasil, fungsi akan menyimpan ID dari thread ciptaan pada rujukan yang ditunjuk thread yaitu variable thread1, pada parameter kedua diisi NULL apabila variabel yang digunakan merupakan variable default. Parameter yang ketiga digunakan untuk menunjuk fungsi yang akan dikerjakan oleh thread. Pada parameter keempat, digunakan untuk mengarahkan ke fungsi yang akan di threaded yaitu message2 (pointer to void).
18) -
19) -
20) -
21) -
22) -
23) pthread_join adalah fungsi untuk melakukan penggabungan dengan thread lain yang telah di-terminasi (telah di exit).Bila thread yang ingin di-join belum diterminasi,fungsi ini mempunyai 2 parameter. fungsi ini akan menunggu hingga thread yang diinginkan telah terminated

24) pthread_join adalah fungsi untuk melakukan penggabungan dengan thread lain yang telah di-terminasi (telah di exit).Bila thread yang ingin di-join belum diterminasi,fungsi ini mempunyai 2 parameter. fungsi ini akan menunggu hingga thread yang diinginkan telah terminated
25) -
26) -printf("Thread 1 returns: %d\n",iret1); Fungsi printf digunakan untuk menampilkan output daripada kalimat yang ada di dalam kurung dan tanda petik tersebut. Sementara *iret1* berfungsi untuk menampilkan nilai yang ada pada *iret1* untuk ditampilkan pada *%d.*
27) -printf("Thread 2 returns: %d\n",iret2); Fungsi printf digunakan untuk menampilkan output daripada kalimat yang ada di dalam kurung dan tanda petik tersebut. Sementara *iret2* berfungsi untuk menampilkan nilai yang ada pada *iret2* untuk ditampilkan pada *%d.*
28) -
29) -
30) -
31) Pendefinisian fungsi void bernama print_message_function().
32) -
33) Pendeklarasian variabel yang bertipe data char yang merupakan pointer to char yang bernama message
34) Void ptr dicasting kedalam bentuk tipe data char lalu diassign ke variabel message
35) Mencetak isi dari variabel message
36) -

2. Mutex

a. Source Code

```
1. #include <stdio.h>
2. #include <stdlib.h>
3. #include <pthread.h>
4.
5. void *functionC();
6. pthread_mutex_t mutex1 = PTHREAD_MUTEX_INITIALIZER;
7. int  counter = 0;
8.
9. main()
10.     {
11.        int rc1, rc2;
12.        pthread_t thread1, thread2;
13.
14.        /* Create independent threads each of which will execute functionC */
15.
16.        if( (rc1=pthread_create( &thread1, NULL, &functionC, NULL)) )
17.        {
18.           printf("Thread creation failed: %d\n", rc1);
19.        }
20.
21.        if( (rc2=pthread_create( &thread2, NULL, &functionC, NULL)) )
22.        {
23.           printf("Thread creation failed: %d\n", rc2);
24.        }
25.
26.        /* Wait till threads are complete before main continues. Unless we  */
```

```
27.         /* wait we run the risk of executing an exit which will terminate   */
28.         /* the process and all threads before the threads have completed.   */
29.
30.         pthread_join( thread1, NULL);
31.         pthread_join( thread2, NULL);
32.
33.         exit(0);
34.     }
35.
36.     void *functionC()
37.     {
38.         pthread_mutex_lock( &mutex1 );
39.         counter++;
40.         printf("Counter value: %d\n",counter);
41.         pthread_mutex_unlock( &mutex1 );
42.     }
```

b. Output

```
rheza@rheza-VirtualBox:~$ gcc -o mutex1 mutex1.c -lpthread
rheza@rheza-VirtualBox:~$ ./mutex1
Counter value: 1
Counter value: 2
```

c. Analisa (list angka menunjukkan baris yang di maksud pada source code)

1) #include<stdio.h> merupakan file header yang berisi deklarasi fungsi-fungsi dasar untuk membat program C
2) #include <stdlib.h> Merupakan file header yang berfungsi untuk operasi pembanding dan operasi konversi.
3) #include <pthread.h> digunakan agar kita dapat menggunakan fungsi - fungsi dari library pthread. Dan untuk mengeksekusi file tersebut, maka kita harus menggunakan -pthread atau -lpthread dalam command line
4) -
5) deklarasi fungsi *functionC();
6) deklarasi mutex1 dengan tipe pthread_mutex_t yang berisi konstanta PTHREAD_MUTEX_INITIALIZER.
7) Mendeklarasikan variabel counter bernilai 0 dengan tipe data int.
8) -
9) main() adalah fungsi utama dalam program, dimana fungsi ini akan dieksekusi pertama kali saat program dijalankan.
10) -
11) Mendeklarasikan variable rc1 dan rc2 dengan tipe data int.
12) Mendeklarasikan variable thread1 dan thread2 dengan tipe data pthread_t. pthread_t adalah sebuah tipe data yang digunakan untuk mengidetifikasi sebuah thread
13) -
14) Memberikan comment, membuat *independent threads* yang masing-masing akan menjalankan fungsi.
15) -
16) pthread_create adalah sebuah fungsi yang digunakan untuk membuat sebuah thread baru. fungsi ini memiliki 4 parameter. Parameter yang pertama berfungsi

sebagai nilai kembalian berupa thread id dari thread yang berhasil dibuat. parameter kedua berisi NULL karena menggunakan variabel default. Parameter yang ketiga digunakan untuk menunjuk fungsi yang akan dikerjakan oleh thread. Pada parameter keempat, digunakan untuk mengarahkan ke fungsi yang akan di threaded. Karena berisi NULL maka tidak akan mengarah.

17) Syntax untuk membuka kondisi pertama.
18) Printf akan menampilkan “Thread creation failed: %d\n“ dengan %d adalah return value dari rc1 yang bertipe data int.
19) Syntax untuk menutup kondisi pertama.
20) -
21) pthread_create adalah sebuah fungsi yang digunakan untuk membuat sebuah thread baru. fungsi ini memiliki 4 parameter. Parameter yang pertama berfungsi sebagai nilai kembalian berupa thread id dari thread yang berhasil dibuat. parameter kedua berisi NULL karena menggunakan variabel default. Parameter yang ketiga digunakan untuk menunjuk fungsi yang akan dikerjakan oleh thread. Pada parameter keempat, digunakan untuk mengarahkan ke fungsi yang akan di threaded. Karena berisi NULL maka tidak akan mengarah.
22) Syntax untuk membuka selain kondisi pertama.
23) Printf akan menampilkan “Thread creation failed: %d\n“ dengan %d adalah return value dari rc2 yang bertipe data int.
24) Syntax untuk menutup selain kondisi pertama.
25) -
26) Memberikan comment, menunggu *threads* selesai, sebelum melanjutkan proses utama. Kecuali jika kita tiadak,
27) Memberikan comment, menunggu kita beresiko keluar dari proses yang dieksekusi yang akan,
28) Memberikan comment, menghentikan proses dan semua *threads* sebelum *threads* selesai.
29) -
30) pthread_join digunakan untuk melakukan wait sebuah thread untuk dihentikan, kemudian dua parameter tersebut adalah, parameter pertama adalah thread id dari thread yang sedang melakukan waiting. Kemudian parameter kedua adalah exit status dari thread yang melakukan waiting.
31) pthread_join digunakan untuk melakukan wait sebuah thread untuk dihentikan, kemudian dua parameter tersebut adalah, parameter pertama adalah thread id dari thread yang sedang melakukan waiting. Kemudian parameter kedua adalah exit status dari thread yang melakukan waiting.
32) -
33) exit digunakan untuk keluar dari shell dimana ia sedang berjalan. Pada baris ini, proses akan keluar dan mengembalikan nilai 0.
34) ...
35) -
36) Pendefinisian fungsi void bernama fuctionC().
37) ...

38) pthread_mutex_lock digunakan untuk mengunci object mutex yang direferensikan dengan mutex. Dan dalam kasus ini adalah variabel mutex1. Jika mutex sudah dikunci, maka pthread_mutex_lock akan menghalanginya hingga mutex tersedia kembali. Operasi ini kembali dengan object mutex yang direferensikan oleh mutex yang ada dalam kondisi terkunci dengan calling thread sebagai pemiliknya.
39) Nilai variable counter ditambahkan dengan 1.
40) Printf akan menampilkan “Counter value: %d\n“ dengan %d adalah return value dari counter yang bertipe data int.
41) pthread_mutex_unlock digunakan untuk melepaskan object mutex yang direferensikan oleh mutex. Dan membuat mutex kembali tersedia.
42) –

3. Joins

a. Source Code

```
1.  #include <stdio.h>
2.  #include <pthread.h>
3.
4.  #define NTHREADS 10
5.  void *thread_function(void *);
6.  pthread_mutex_t mutex1 = PTHREAD_MUTEX_INITIALIZER;
7.  int  counter = 0;
8.
9.  main()
10. {
11.    pthread_t thread_id[NTHREADS];
12.    int i, j;
13.
14.    for(i=0; i < NTHREADS; i++)
15.    {
16.       pthread_create( &thread_id[i], NULL, thread_function, NULL );
17.    }
18.
19.    for(j=0; j < NTHREADS; j++)
20.    {
21.       pthread_join( thread_id[j], NULL);
22.    }
23.
24.    /* Now that all threads are complete I can print the final result.     */
25.    /* Without the join I could be printing a value before all the threads */
26.    /* have been completed.                                                */
27.
28.    printf("Final counter value: %d\n", counter);
29. }
30.
31. void *thread_function(void *dummyPtr)
32. {
33.    printf("Thread number %ld\n", pthread_self());
34.    pthread_mutex_lock( &mutex1 );
35.    counter++;
36.    pthread_mutex_unlock( &mutex1 );
37. }
```

b. Output

```
rheza@rheza-VirtualBox:~$ gcc -o join join.c -lpthread
rheza@rheza-VirtualBox:~$ ./join
Thread number 139802494777088
Thread number 139802503169792
Thread number 139802486384384
Thread number 139802511562496
Thread number 139802477991680
Thread number 139802519955200
Thread number 139802469598976
Thread number 139802528347904
Thread number 139802536740608
Thread number 139802545133312
Final counter value: 10
```

c. Analisa (list angka menunjukkan baris yang dimaksud pada Source Code)

1) Sebuah header yang berisi fungsi-fungsi yang digunakan untuk operasi input output.
2) file header <pthread.h> digunakan untuk mendefinisikan beberapa tipe data, fungsi, dan konstanta dari C untuk thread, semisal pada program ini dibutuhkan tipe pthread_t dan pthread_mutex_t, dan misal fungsi pthread_create, pthread_join, pthread_mutex_unlock, pthread_mutex_lock.
3) -
4) Mendefinisikan nilai NTHREADS adalah 10
5) Pendeklarasian fungsi void *thread_function(void *) yang akan dijalankan oleh thread untuk mencetak ID thread dan mengincrement nilai counter.
6) deklarasi mutex1 dengan tipe pthread_mutex_t yang berisi konstanta PTHREAD_MUTEX_INITIALIZER.
7) Menginisialisasi counter yang memiliki tipe data integer dengan nilai 0.
8) -
9) main() adalah fungsi utama dalam program, dimana fungsi ini akan dieksekusi pertama kali saat program dijalankan.
10) Syntax untuk membuka deklarasi fungsi main().
11) Pthread_t merupakan tipe data int yang berisi ID dari thread. Pada baris ini, dideklarasikan variabel bertipe data array of pthread_t bernama thread_id dengan array sebesar NTHREADS.
12) Mendeklarasikan variabel i dan j bertipe integer
13) -
14) Melakukan loop dengan nilai awal i-0 dan i diincrement. Selama i < NTHREADS, kerjakan baris 15-17.
15) Syntax untuk membuka kondisi for (i=0; i<NTHREADS; i++).
16) pthread_create adalah sebuah fungsi yang digunakan untuk membuat sebuah thread baru. fungsi ini memiliki 4 parameter. Parameter yang pertama berisi pointer yang mengembalikan nilai thread id dari thread yang berhasil dibuat. parameter kedua berisi NULL karena menggunakan variabel default. Parameter yang ketiga digunakan untuk menunjuk fungsi yang akan dikerjakan oleh

thread. Parameter keempat merupakan pointer argumen dari fungsi yang akan di threaded. Berisi NULL karena tidak ada argumen yang di-passing-kan ke fungsi tersebut.

17) Syntax untuk menutup kondisi for (i=0; i<NTHREADS; i++).
18) –
19) Melakukan loop dengan nilai awal j=0 dan j diincrement. Selama j < NTHREADS, kerjakan baris 15-17.
20) Syntax untuk membuka kondisi for (j=0; j<NTHREADS; j++).
21) pthread_join digunakan untuk menunggu berhentinya thread yang dispesifikkan. Parameter pertama diisi dengan thread id yang akan ditunggu. Parameter kedua diisi dengan pointer menuju lokasi dimana exit status dari thread yang disebutkan disimpan. Karena tidak dibutuhkan, maka pada baris ini diisi dengan NULL.
22) Syntax untuk menutup kondisi for (j=0; j<NTHREADS; j++).
23) -
24) komentar
25) komentar
26) komentar
27) -
28) printf akan menampilkan "Counter value: %d\n" dengan %d adalah return value dari counter yang bertipe data int.
29) Syntax untuk menutup deklarasi fungsi main().
30) -
31) Fungsi thread_function dengan parameter `void *dummyPtr`
32) Syntax untuk membuka deklarasi fungsi void().
33) memanggil sebuah fungsi bernama pthread_self() untuk menghasilkan return value berupa ID thread kemudian mencetaknya dengan fungsi printf().
34) pthread_mutex_lock digunakan untuk mengunci object mutex yang direferensikan dengan mutex. Dan dalam kasus ini adalah variabel mutex1. Jika mutex sudah dikunci, maka pthread_mutex_lock akan menghalanginya hingga mutex tersedia kembali. Operasi ini kembali dengan object mutex yang direferensikan oleh mutex yang ada dalam kondisi terkunci dengan calling thread sebagai pemiliknya.
35) Nilai variable counter ditambahkan dengan 1.
36) pthread_mutex_unlock digunakan untuk melepaskan object mutex yang direferensikan oleh mutex. Dan membuat mutex kembali tersedia.
37) Syntax untuk membuka deklarasi fungsi void().

4. Conditional Variables

a. Source Code

```
1.  #include <stdio.h>
2.  #include <stdlib.h>
3.  #include <pthread.h>
4.
5.  pthread_mutex_t count_mutex     = PTHREAD_MUTEX_INITIALIZER;
6.  pthread_mutex_t condition_mutex = PTHREAD_MUTEX_INITIALIZER;
7.  pthread_cond_t  condition_cond  = PTHREAD_COND_INITIALIZER;
8.
9.  void *functionCount1();
10. void *functionCount2();
11. int  count = 0;
12. #define COUNT_DONE  10
13. #define COUNT_HALT1  3
14. #define COUNT_HALT2  6
15.
16. main()
17. {
18.    pthread_t thread1, thread2;
19.
20.    pthread_create( &thread1, NULL, &functionCount1, NULL);
21.    pthread_create( &thread2, NULL, &functionCount2, NULL);
22.    pthread_join( thread1, NULL);
23.    pthread_join( thread2, NULL);
24.
25.    exit(0);
26. }
27.
28. void *functionCount1()
29. {
30.    for(;;)
31.    {
32.       pthread_mutex_lock( &condition_mutex );
33.       while( count >= COUNT_HALT1 && count <= COUNT_HALT2 )
34.       {
35.          pthread_cond_wait( &condition_cond, &condition_mutex );
36.       }
37.       pthread_mutex_unlock( &condition_mutex );
38.
39.       pthread_mutex_lock( &count_mutex );
40.       count++;
41.       printf("Counter value functionCount1: %d\n",count);
42.       pthread_mutex_unlock( &count_mutex );
43.
44.       if(count >= COUNT_DONE) return(NULL);
45.     }
46. }
47.
48. void *functionCount2()
49. {
50.     for(;;)
51.     {
52.        pthread_mutex_lock( &condition_mutex );
53.        if( count < COUNT_HALT1 || count > COUNT_HALT2 )
54.        {
55.           pthread_cond_signal( &condition_cond );
56.        }
57.        pthread_mutex_unlock( &condition_mutex );
58.
59.        pthread_mutex_lock( &count_mutex );
60.        count++;
61.        printf("Counter value functionCount2: %d\n",count);
62.        pthread_mutex_unlock( &count_mutex );
```

```
63.
64.         if(count >= COUNT_DONE) return(NULL);
65.     }
66.
67. }
```

b. Output

```
rheza@rheza-VirtualBox:~$ gcc -o cond1 cond1.c -lpthread
rheza@rheza-VirtualBox:~$ ./cond1
Counter value functionCount2: 1
Counter value functionCount2: 2
Counter value functionCount2: 3
Counter value functionCount2: 4
Counter value functionCount2: 5
Counter value functionCount2: 6
Counter value functionCount2: 7
Counter value functionCount2: 8
Counter value functionCount2: 9
Counter value functionCount2: 10
Counter value functionCount1: 11
```

c. Analisa (list angka menunjukkan baris yang dimaksud pada Source Code)

1) #include <studio.h> adalah library dalam bahasa pemrograman C yang digunakan untuk input output, tanpa menggunakan library tersebut maka program C yang kita buat tidak bisa digunakan untuk menampilkan atau menginput sesuatu.
2) #include <stdlib.h> digunakan untuk melakukan berbagai operasi, termasuk konversi, pseudo-acak nomor, alokasi memori, kontrol proses, lingkungan, sinyal, pencarian, dan sortasi.
3) #include <pthread.h> : Untuk menggunakan fungsi-fungsi dan tipe data yang ada pada thread
4) -
5) pthread_mutex_t count_mutex = PTHREAD_MUTEX_INITIALIZER; baris ini meng-*aasign* value yaitu “PTHREAD_MUTEX_INITIALIZER” kedalam variabel “count_mutex” yang memiliki tipe data *pthread_mutex_t* . “PTHREAD_MUTEX_INITIALIZER” adalah makro yang bisa digunakan untuk inisialisasi mutex yang dialokasikan secara statis.
6) pthread_mutex_t condition_mutex = PTHREAD_MUTEX_INITIALIZER; baris ini untuk meng-*assign* valaue yaitu “PTHREAD_MUTEX_INITIALIZER” ke dalam variabel “conditional_mutex” yang memiliki tipe data pthread_mutex_t. “PTHREAD_MUTEX_INITIALIZER” adalah makro yang bisa digunakan untuk inisialisasi mutex yang dialokasikan secara statis.
7) pthread_cond_t condition_cond = PTHREAD_COND_INITIALIZER;

8) baris ini meng-*assign* value yaitu “PTHREAD_COND_INITIALIZER” kedalam variabel “condition_cond” yang memiliki tipe data *pthread_cond_t* . “PTHREAD_COND_INITIALIZER adalah makro untuk menginisialisasi kondisi variabel secara statis ke default atributnya.

9) -
10) void *functionCount1(); adalah pendeklarasian fungsi functionCount1 yang nantinya akan digunakan oleh thread1.
11) void *functionCount2(); deklarasi fungsi functionCount2 yang nantinya akan digunakan oleh thread2.
12) int count = 0, merupakan pendeklarasian variabel “count” yang bernilai 0 dengan tipe data *int*.
13) #define COUNT_DONE 10 : mendefinisikan COUNT DONE bernilai 10
14) #define COUNT_HALT1 3 mendefinisikan COUNT_HALT1 bernilai 3
15) #define COUNT_HALT2 6 : mendefinisikan sebuah konstanta bernama COUNT_HALT2 sebesar 6
16) -
17) Fungsi main().
18) {. Kurung kurawal buka digunakan sebagai pembuka seluruh perintah yang akan dieksekusi di dalam fungsi main()
19) pthread_t thread1, thread2; merupakan pendeklerasian variabel thread1 dan thread2 yang bertipe data pthread_t yang ada pada main.
20) -
21) pthread_create(&thread1, NULL, &functionCount1, NULL); membuat thread pada thread identifier thread1, parameter kedua menunjukan parameter attr bernilai NULL yang artinya thread dideklarasi secara default, thread1 akan menjalankan fungsi functionCount1. karena fungsi functionCount1 tidak memiliki parameter, paka parameter keempat adalah NULL.
22) pemanggilan fungsi pthread_create dengan 4 parameter, yaitu &thread2, NULL, &functionCount2, NULL. Parameter ke-1, &thread2, sebuah variabel yang digunakan untuk menampung alamat dari thread ID yang nantinya dibuat. Parameter ke-2, NULL, artinya thread yang dibuat akan diisi dengan default atribut. Parameter ke-3, &functionCount2, merupakan fungsi utama untuk sebuah thread yang mulai mengeksekusi user code pada alamat tersebut. Parameter ke-4, NULL, nilai awal untuk memulai eksekusi. Tujuan dari fungsi ini adalah untuk membuat thread baru.
23) pthread_join(thread1, NULL); memanggil fungsi join yang artinya menunggu thread1 selesai menjalankan fungsi functionCount1(terminated) sebelum berlanjut ke program selanjutnya. karena tidak ada callback function setelah thread1 terminated, maka parameter kedua adalah NULL.
24) Pemanggilan fungsi pthread_join(thread2, NULL) dengan 2 parameter. Parameter ke-1, thread2, merupakan thread yang akan ditunggu. Parameter ke-2, NULL, lokasi untuk menyimpan exit status dari joined thread, diset NULL karena exit status tidak dibutuhkan. Pemanggilan fungsi ini digunakan untuk menunggu thread thread2 selesai diterminasi.
25) -
26) exit(0). Program akan selesai dieksekusi dengan retval 0.
27) }. Kurung kurawal tutup sebagai akhir dari fungsi main().
28) -

29) void *functionCount1() : pemanggilan fungsi functionCount1 yang akan digunakan oleh thread1.
30) tanda kurung kurawal buka ({) untuk membuka statement untuk mengeksekusi uang adad di dalam fungsi *functionCount() bertipe data void
31) for(;;) : for infinity loop yang digunakan dalam C/C++.
32) {. Kurung kurawal buka untuk memulai code pada statement for().
33) pthread_mutex_lock(&condition_mutex); adalah pemanggilan fungsi pthread_mutex_lock dengan parameter alamat dari condition_mutex. fungsi ini akan mengunci condition_mutex hingga proses yang melibatkan mutex telah selesai dieksekusi.
34) while(count >= COUNT_HALT1 && count <= COUNT_HALT2) : Kondisional while ketika nilai dari count lebih dari atau sama dengan COUNT_HALT1 dan nilai dari count kurang dari atau sama dengan COUNT_HALT2. Jika kondisional terpenuhi, maka block kode dibawahnya akan dijalankan.
35) tanda kurung kurawal buka ({) untuk membuka statement eksekusi looping while
36) pthread_cond_wait(&condition_cond, &condition_mutex) : adalah memanggil fungsi *pthread_cond_wait* dengan 2 parameter yaitu *condition_cond & condition mutex* dengan passing by reference. fungsi ini digunakan untuk memblok/menghentikan kondisi variabel dengan thread yang dipanggil melalui *condition_cond* melepaskan mutex melalui *condition_mutex*
37) Tanda kurung kurawal tutup **(})** menunjukkan bahwa eksekusi yang dilakukan sudah selesai jika condition pada while sudah tidak terpenuhi lagi.
38) pthread_mutex_unlock(&condition_mutex); : Memanggil fungsi phthread_mutex_unlock dengan parameter alamat dari condition_mutex. Fungsi ini digunakan untuk melepas status lock/terkunci pada mutex tersebut.
39) -
40) pthread_mutex_lock(&count_mutex); : Pemanggilan fungsi pthread_mutex_lock dengan parameter alamat dari count_mutex. fungsi ini akan mengunci count_mutex hingga proses yang melibatkan mutex telah selesai dieksekusi.
41) count ++ merupakan kependekan dari count = count + 1. Artinya nilai dari variabel count ditambah 1 dari nilai sebelumnya. count ++ berfungsi untuk mencatat berapa kali looping terjadi, jika looping sudah mencapai batas yang ditentukan, maka looping akan berhenti.
42) printf("Counter value functionCount1: %d\n",count); : Mencetak nilai dari counter pada fungsi functionCount1 (thread 1) yang sesuai dengan count yang didapat berdasarkan kondisional jika terpenuhi.
43) pthread_mutex_unlock(&count_mutex);. Pemanggilan fungsi pthread_mutex_unlock dengan parameter alamat dari count_mutex akan membebaskan count_mutex sehingga penjadwalan akan menentukan thread mana yang akan menggunakan mutex tersebut.
44) -
45) if(count >= COUNT_DONE) return(NULL); merupakan suatu kondisi (if) untuk menjalankan (mengeksekusi suatu program) yang apabila syaratnya atau syarat tertentu telah terpenuhi. Pada program ini kondisinya if (count >=

COUNT_DONE) yang mana maksudnya syarat untuk mengeksekusi program tersebut jika count sudah lebih besar sama dengan COUNT_DONE maka akan dieksekusi return (NULL) karena syarat sudah terpenuhi.

46) }. Kurung kurawal tutup sebagai penanda akhir dari statement for().
47) } sebagai tanda akhir blok fungsi functionCount1()
48) -
49) void *functionCount2() pemanggilan fungsi functionCount2 yang akan digunakan oleh thread2..
50) Tanda kurung kurawal buka ({) merupakan tanda awal dari blok kode *functionCount2() yang bertipe data void.
51) for(;;) merupakan for infinity loop yang digunakan dalam C/C++.
52) Tanda kurung kurawal buka **({)** merupakan tanda awal dari block kode loop for
53) pthread_mutex_lock(&condition_mutex); adalah pemanggilan fungsi pthread_mutex_lock dengan parameter alamat dari condition_mutex. fungsi ini akan mengunci condition_mutex. Hal ini akan memblok pemanggilan thread sampai mutex tidak di-lock.
54) kondisional if akan mengecek apakah nilai dari count kurang dari COUNT_HALT1 atau nilai dari count lebih dari COUNT_HALT2. Jika kondisional ini terpenuhi maka, block kode dibawah akan dijalankan.
55) Tanda kurung kurawal buka **({)** merupakan tanda awal dari blok kode if(count < COUNT_HALT1 || count > COUNT_HALT2)
56) pthread_cond_signal(&condition_cond); adalah pemanggilan fungsi pthread_cond_signal dengan parameter alamat dari condition_cond. Fungsi phtread_cond_signal sendiri dipanggil thread berdasarkan kondisi
57) }. Kurung kurawal tutup sebagai akhir dari statement if.
58) pthread_mutex_unlock(&condition_mutex); adalah pemanggilan fungsi pthread_mutex_unlock dengan parameter alamat dari condition_mutex. fungsi ini akan membuka kunci conditional_mutex sehingga mutex kembali tersedia.
59) -
60) pthread_mutex_lock(&count_mutex); digunakan untuk mengunci mutex count_mutex ketika thread2 ingin menjalankan program count++ agar tidak ada thread lain yang menginterupsi.
61) count ++ merupakan kependekan dari count = count + 1. Artinya nilai dari variabel count ditambah 1 dari nilai sebelumnya
62) printf("Counter value functionCount2: %d\n",count); digunakan untuk mencetak string "Counter value functionCount2: " dan isi dari variabel count dengan format integer (%d).
63) pthread_mutex_unlock(&count_mutex); adalah pemanggilan fungsi pthread_mutex_unlock dengan parameter alamat dari count_mutex. fungsi ini akan membuka kunci count_mutex sehingga mutex kembali tersedia.
64) -
65) if(count >= COUNT_DONE) return(NULL); merupakan suatu kondisi (if)untuk menjalankan (mengeksekusi suatu program) yang apabila syaratnya atau syarat tertentu telah terpenuhi. Pada program ini kondisinya if (count >=

COUNT_DONE) yang mana maksudnya syarat untuk mengeksekusi program tersebut jika count sudah lebih besar samadengan COUNT_DONE maka akan dieksekusi return (NULL) karena syarat sudah terpenuhi.

66) kurung kurawal tutup (}) untuk menutup statement eksekusi for(;;). for(;;) melakukan perulangan proses tak terhingga karena tidak ada inisialisasi awal, kondisi berhenti, dan iterasi.

67) -

68) tanda kurung kurawal **(})** merupakan tanda akhir dari block kode *functionCount2()

5. Single Thread

a. Source Code

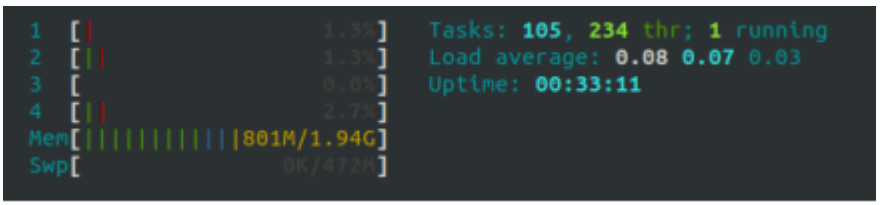
```
1.  // CPP Program to multiply two matrix using pthreads
2.  #include <bits/stdc++.h>
3.  using namespace std;
4.  using namespace chrono;
5.  // maximum size of matrix
6.  #define MAX 1000
7.
8.
9.  int matA[MAX][MAX];
10. int matB[MAX][MAX];
11. int matC[MAX][MAX];
12. int step_i = 0;
13.
14. void* multi(void* arg)
15. {
16.     for (int i = 0; i <  MAX ; i++)
17.         for (int j = 0; j < MAX; j++)
18.             for (int k = 0; k < MAX; k++)
19.                 matC[i][j] += matA[i][k] * matB[k][j];
20. }
21.
22. // Driver Code
23. int main()
24. {
25.     // Generating random values in matA and matB
26.     for (int i = 0; i < MAX; i++) {
27.         for (int j = 0; j < MAX; j++) {
28.             matA[i][j] = rand() % 10;
29.             matB[i][j] = rand() % 10;
30.         }
31.     }
32.
33.     // Displaying matA
34.     //cout << endl
35.     //  << "Matrix A" << endl;
36.     //for (int i = 0; i < MAX; i++) {
37.     //  for (int j = 0; j < MAX; j++)
38.     //      cout << matA[i][j] << " ";
39.     //  cout << endl;
40.     //}
41.
42.     // Displaying matB
43.     //cout << endl
44.     //  << "Matrix B" << endl;
45.     //for (int i = 0; i < MAX; i++) {
46.     //  for (int j = 0; j < MAX; j++)
47.     //      cout << matB[i][j] << " ";
```

```
48.     //  cout << endl;
49.     //}
50.
51.     auto a = high_resolution_clock::now();
52.     int* p;
53.     multi((void*)(p));
54.
55.     // Displaying the result matrix
56.     //cout << endl
57.     //  << "Multiplication of A and B" << endl;
58.     //for (int i = 0; i < MAX; i++) {
59.     //  for (int j = 0; j < MAX; j++)
60.     //      cout << matC[i][j] << " " ;
61.     //  cout << endl;
62.     //}
63.
64.     auto b = high_resolution_clock::now();
65.     cout << "Took " << duration_cast<milliseconds>(b -
    a).count() << " milliseconds" <<  endl;
66.     return 0;
67. }
```

ANALISA OUTPUT:

Source code diatas yaitu menjalankan program matriks dengan hanya satu thread saja. Dan ketika program tersebut dijalankan oleh hanya satu thread (Single Thread), maka prosesor yang terpakai pada satu thread ini bisa satu saja. Dan itulah yang dimaksud dengan single thread, menjalankan suatu program hanya dengan satu thread saja. Hal ini tentu akan membuang-buang banyak waktu karena semisal terdapat banyak program yang kita jalankan tetapi hanya satu thread saja yang berjalan, maka program-program tersebut dijalankan bergantian. Tentu hal tersebut tidak efisien. Maka berikut adalah hasil output dari program matriks satu thread.

Dan sebelum kita jalankan program matriks diatas maka kita jalankan perintah htop, dan pada awal dijalankan, tampilannya akan seperti ini:



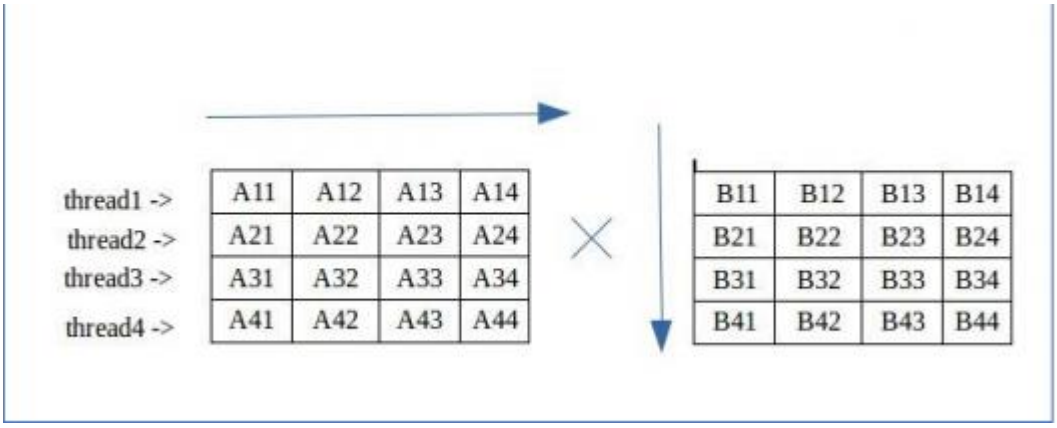
Nomor 1 sampai 4 pada gambar diatas yaitu adalah menandakan terdapat 4 CPU yang semestinya bisa menjalankan 4 threads.

Terdapat 4 CPU/prosesor yang telah tersedia pada sistem operasi ini. Dan ketika kita eksekusi programnya, maka akan menjadi seperti berikut:

```
rheza@rheza-VirtualBox:~$ g++ -o matrices-single-global-var
 matrices-single-global-var.cpp
rheza@rheza-VirtualBox:~$ ./matrices-single-global-var
Took 20874 milliseconds
```

Setelah kita menjalankan program matriksnya, ternyata didapatkan pada salah satu dari CPU tersebut penuh kapasitasnya. Hal ini tandanya adalah sebuah thread (Single thread) hanya menggunakan satu prosesor saja dalam menjalankan program matriks. Dan itulah yang terjadi ketika kita hanya memakai single thread pada setiap program yang kita jalankan. CPU/prosesor lainnya akan idle/menganggur karena hanya cukup satu prosesor saja dalam menjalankan satu thread. Dan tentunya hal ini bukanlah sesuatu yang efisien.

Strategi yang dipakai adalah menggunakan hanya satu thread saja untuk menjalankan program perkalian matriks tersebut. Maka jika hanya ada satu thread, yang terjadi adalah thread tersebut akan menghitung perkalian antar baris dan kolom secara bergantian mulai dari baris dan kolom pertama.



Jika melihat gambar diatas maka yang aktif hanyalah thread 1 saja. Anggap thread 2, thread 3, thread 4 tidak ada. Maka thread 1 akan menjalankan perkalian baris 1 matriks A dan kolom 1 matriks B. selanjutnya dilakukan perkalian baris 1 matriks A dan kolom 2 matriks B. dan akan berlanjut secara bergantian sampai semua baris dan kolom pada matriks tersebut selesai dihitung. Logikanya apabila seperti itu maka akan lama, karena tidak bisa menghitung secara bersamaan.

6. Multi Thread

a. Source Code

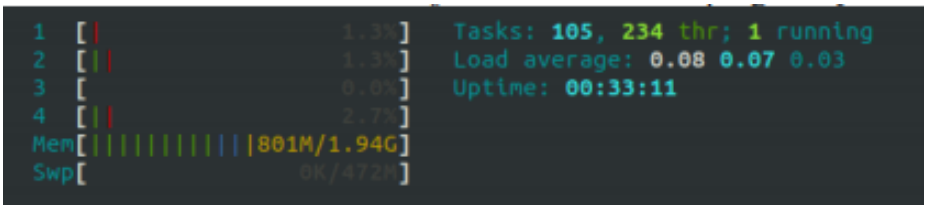
```
1)  // CPP Program to multiply two matrix using pthreads
2)  #include <bits/stdc++.h>
3)  #include <chrono>
4)  using namespace std;
5)  using namespace chrono;
6)  // maximum size of matrix
7)  #define MAX 1000
8)
9)  // maximum number of threads
10) #define MAX_THREAD 4096
11)
12) int matA[MAX][MAX];
13) int matB[MAX][MAX];
14) int matC[MAX][MAX];
```

```
15) int step_i = 0;
16) int mymax;
17)
18) void* multi(void* arg)
19) {
20)     int core = step_i++;
21)
22)     // Each thread computes 1/MAX_THREAD th of matrix multiplication
23)     for (int i = core * MAX / mymax; i < (core + 1) * MAX / mymax; i++)
24)         for (int j = 0; j < MAX; j++)
25)             for (int k = 0; k < MAX; k++)
26)                 matC[i][j] += matA[i][k] * matB[k][j];
27) }
28)
29) // Driver Code
30) int main()
31) {
32)
33)     // Generating random values in matA and matB
34)     for (int i = 0; i < MAX; i++) {
35)         for (int j = 0; j < MAX; j++) {
36)             matA[i][j] = rand() % 10;
37)             matB[i][j] = rand() % 10;
38)         }
39)     }
40)
41)     printf("how many threads: ");
42)     scanf("%d", &mymax);
43)     fflush(stdin);
44)     auto a = high_resolution_clock::now();
45)     // declaring four threads
46)     pthread_t threads[MAX_THREAD];
47)
48)     // Creating four threads, each evaluating its own part
49)     for (int i = 0; i < mymax; i++) {
50)         int* p;
51)         pthread_create(&threads[i], NULL, multi, (void*)(p));
52)     }
53)
54)     // joining and waiting for all threads to complete
55)     for (int i = 0; i < mymax; i++)
56)         pthread_join(threads[i], NULL);
57)
58)     auto b = high_resolution_clock::now();
59)     cout << "Using " << mymax << " threads" << endl;
60)     cout << "Took " << duration_cast<milliseconds>(b -
    a).count() << " milliseconds" <<  endl;
61)     return 0;
62) }
```

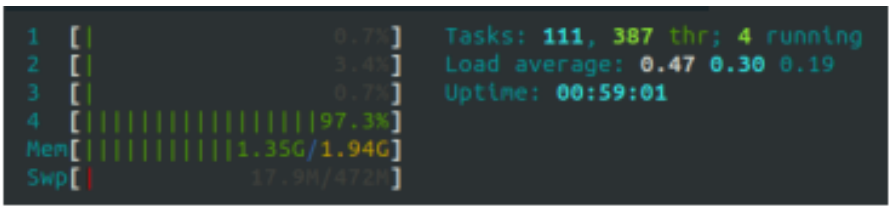
ANALISA OUTPUT:

Source code diatas masih sama dengan source code sebelumnya, dimana kita akan mencoba lagi menjalankan program perkalian dua matriks. Tetapi kali ini kita akan menentukan sendiri berapa jumlah thread yang ingin kita jalankan. Setelah itu kita buat perbandingan waktunya dengan menggunakan grafik.

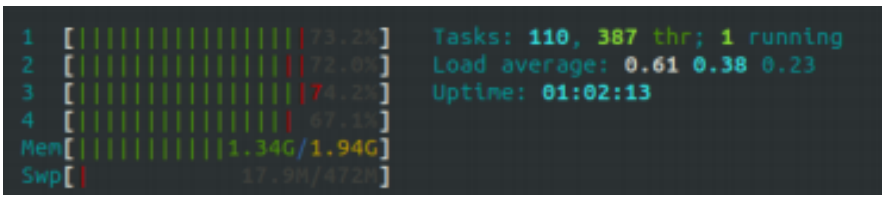
Maka kondisi sebelum kita menjalankan thread dan programnya adalah seperti berikut:



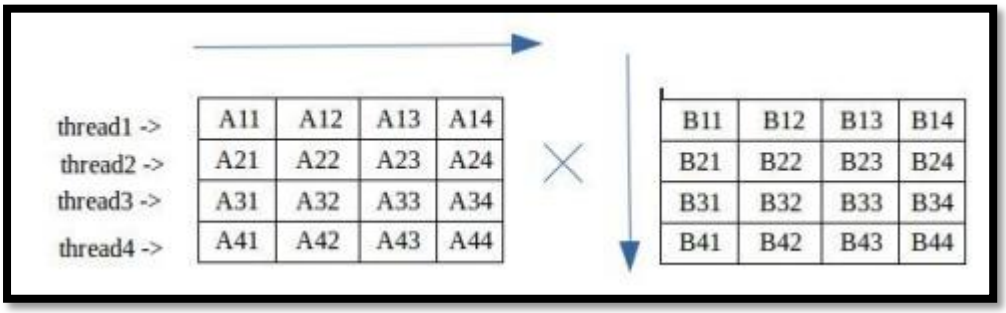
Dan lalu kita jalankan program matriks multi-thread nya, kita coba memasukkan hanya satu thread saja seperti pada percobaan sebelumnya:



Lalu kita bandingkan dengan thread lebih dari satu, misalnya empat threads. Maka yang terjadi adalah seperti berikut:

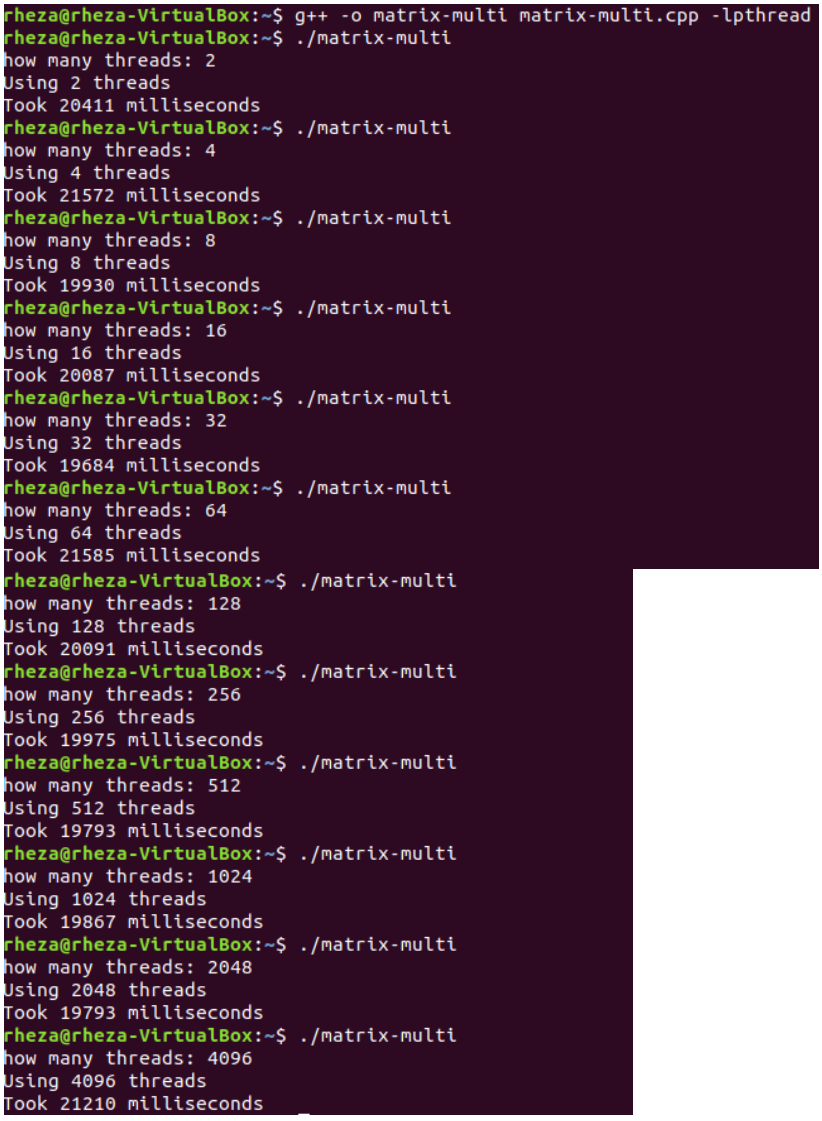


Kita bisa mengetahui bahwa dengan multi-thread, sistem dapat dengan mudah untuk menjalankan banyak proses sekali waktu (secara bersamaan). Dengan multi-thread maka proses yang bercabang, akan dibagi menjadi beberapa thread yang gunanya adalah untuk tetap menjaga supaya tidak membuang-buang waktu. Misal terdapat suatu proses yang membutuhkan banyak proses-proses kecil lainnya, sementara hanya ada satu thread yang bisa melayani, maka proses-proses kecil yang banyak tersebut akan dikerjakan secara bergantian dan bukan bersamaan.

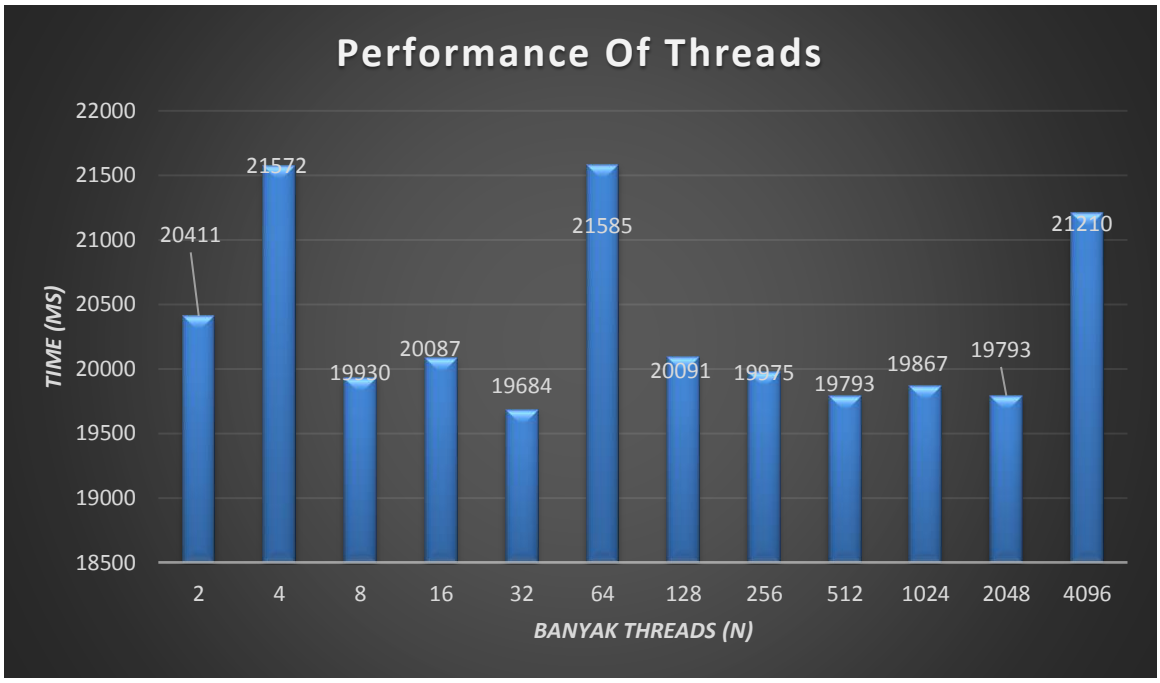


Pada gambar diatas terdapat misal 4 thread. Jika pada single thread dapat berjalan secara bergantian, maka dengan multithread proses perhitungan dapat dilakukan bersamaan. Hal ini dikarenakan ibarat 4 thread tersebut dibagi tugas sama rata untuk mengerjakan perkalian tersebut. Maka thread 1 bisa mengerjakan perhitungan baris satu dengan kolom-kolom yang lain. lalu thread 2 bisa mengerjakan perhitungan baris dua dengan kolom-kolom yang lain. dan akan begitu sampai thread 4. Dan akhirnya proses perhitungan dapat berjalan secara efisien.

Sekarang kita akan analisa multi-thread dari segi waktunya. Kita akan coba dengan jumlah thread sebesar 2^1 sampai 2^{12}



Analisa Output Dalam Bentuk Grafik



Dari thread n=0 sampai n=4 kita lihat terjadi penurunan waktu komputasinya. Dugaan awal kenapa terjadi penurunan waktu komputasi dikarenakan semakin banyak thread, maka proses yang dilakukan akan semakin cepat. Kita lihat pada n=0, didapat waktu komputasinya adalah 6358 millisecond, menandakan bahwa satu proses ini besar dan lama jika hanya ditanggung oleh satu thread saja. Lalu kita lihat pada n=1 waktu komputasinya pun turun drastis hingga 2877 hampir setengahnya dari n=0. Karena memang satu proses yang sama di tanggung oleh dua thread, ibaratnya satu pekerjaan ditanggung oleh 2 orang. Dan begitu terus sampai pada n=4.

Tetapi semuanya dipatahkan ketika percobaan n=5, ternyata waktu komputasi tersebut naik walaupun hanya sedikit sekali. Dan lebih naik lagi pada n=6. Dan jika kita teruskan dari n=7 sampai n=9, akan turun lagi. Dan ketika dari n=10 sampai n=12, terjadi kenaikan waktu lagi. Ternyata ide “banyak proses jika dikerjakan oleh banyak thread, maka proses tersebut akan berlangsung lebih cepat” adalah tidak selalu benar.

Hal tersebut juga dijelaskan pada Hukum Amdahl:

$$speedup \leq \frac{1}{S + \frac{(1-S)}{N}}$$

Dalam kasus thread ini, kita bisa memakai konsep komputasi paralel dari hukum ini. Pada rumus tersebut dijelaskan bahwa ketika kita ingin menambah kuantitas bagian komputer dengan tujuan mempercepat komputasi (misal menambah prosesor) itu tidak selalu akan memberikan hasil yang benar. Maka pada hukum tersebut, komputasi parallel juga penting untuk memroses atau melakukan komputasi yang efisien. Dan mengapa thread dengan jumlah yang banyak itu tidak menurunkan waktu komputasi, itu dikarenakan dikerjakan secara serial dan bukan paralel. Pada hukum Amdahl dijelaskan bahwa jika kita setidaknya membuat suatu proses dikerjakan 75% paralel

dan 25% seri, maka akan menaikkan performa naik menjadi 1.6 kali lebih besar. Maka konsep tersebut bias digunakan dalam multi-threading.

7. Deadlock

a. Source Code

```
1.  #include <stdio.h>
2.  #include <stdlib.h>
3.  #include <unistd.h>
4.  #include <pthread.h>
5.
6.  void *function1();
7.  void *function2();
8.  pthread_mutex_t mutex1 = PTHREAD_MUTEX_INITIALIZER;
9.  pthread_mutex_t mutex2 = PTHREAD_MUTEX_INITIALIZER;
10. int  counter = 0;
11.
12. main()
13. {
14.    int rc1, rc2;
15.    pthread_t thread1, thread2;
16.
17.    /* Create independent threads each of which will execute functionC */
18.
19.    /*
20.    if( (rc1=pthread_create( &thread1, NULL, &function1, NULL)) )
21.    {
22.       printf("Thread creation failed: %d\n", rc1);
23.    }
24.    if( (rc2=pthread_create( &thread2, NULL, &function2, NULL)) )
25.    {
26.       printf("Thread creation failed: %d\n", rc2);
27.    } */
28.
29.    pthread_create( &thread1, NULL, &function1, NULL);
30.    pthread_create( &thread2, NULL, &function2, NULL);
31.
32.    /* Wait till threads are complete before main continues. Unless we  */
33.    /* wait we run the risk of executing an exit which will terminate   */
34.    /* the process and all threads before the threads have completed.   */
35.
36.    pthread_join( thread1, NULL);
37.    pthread_join( thread2, NULL);
38.
39.    exit(0);
40. }
41.
42. void *function1()
43. {
44.    pthread_mutex_lock( &mutex1 );
45.    sleep(5);
46.    pthread_mutex_lock( &mutex2 );
47.    counter++;
48.    printf("Counter value in function1: %d\n",counter);
49.    pthread_mutex_unlock( &mutex2 );
50.    pthread_mutex_unlock( &mutex1 );
51. }
52.
53. void *function2()
54. {
55.    pthread_mutex_lock( &mutex2 );
56.    sleep(5);
```

```
57.    pthread_mutex_lock( &mutex1 );
58.    counter++;
59.    printf("Counter value in function2: %d\n",counter);
60.    pthread_mutex_unlock( &mutex1 );
61.    pthread_mutex_unlock( &mutex2 );
62. }
```

b. Output

```
rheza@rheza-VirtualBox:~$ gcc -o deadlock deadlock.c -lpthread
rheza@rheza-VirtualBox:~$ ./deadlock
```

c. Analisa (list angka menunjukkan baris yang dimaksud pada Source Code)

1) #include<stdio.h> merupakan file header yang berisi deklarasi fungsi-fungsi dasar untuk membuat program C
2) #include <stdlib.h> Merupakan file header yang berfungsi untuk operasi pembanding dan operasi konversi.
3) #include <unistd.h> merupakan file header dari standart library bahasa pemrogaman C yang fungsinya untuk mendeklarasikan variable dan fungsi dari user yang terdiri dari API sitem operasi POSIX, seperti size_t dan fork().
4) #include <pthread.h> digunakan agar kita dapat menggunakan fungsi - fungsi dari library pthread. Dan untuk mengeksekusi file tersebut, maka kita harus menggunakan -pthread atau -lpthread dalam command line
5) –
6) deklarasi fungsi *function1();
7) deklarasi fungsi *function2();
8) deklarasi mutex1 dengan tipe pthread_mutex_t yang berisi konstanta PTHREAD_MUTEX_INITIALIZER.
9) deklarasi mutex2 dengan tipe pthread_mutex_t yang berisi konstanta PTHREAD_MUTEX_INITIALIZER.
10) Mendeklarasikan variabel counter bernilai 0 dengan tipe data int.
11) –
12) main() adalah fungsi utama dalam program, dimana fungsi ini akan dieksekusi pertama kali saat program dijalankan.
13) Syntax untuk membuka deklarasi fungsi main().
14) Mendeklarasikan variable rc1 dan rc2 dengan tipe data int.
15) Mendeklarasikan variable thread1 dan thread2 dengan tipe data pthread_t. pthread_t adalah sebuah tipe data yang digunakan untuk mengidetifikasi sebuah thread
16) –
17) /* tanda pembuka komentar untuk : Create independent threads each of which will execute functionC. */ tanda penutup komentar
18) –
19) /* tanda pembuka komentar
20) if((rc1=pthread_create(&thread1, NULL, &function1, NULL)))
21) {

22) printf("Thread creation failed: %d\n", rc1);
23) }
24) if((rc2=pthread_create(&thread2, NULL, &function2, NULL)))
25) {
26) printf("Thread creation failed: %d\n", rc2);
27) }. */ tanda penutup komentar
28) –
29) pthread_create adalah sebuah fungsi yang digunakan untuk membuat sebuah thread baru. fungsi ini memiliki 4 parameter. Parameter yang pertama berfungsi sebagai nilai kembalian berupa thread id dari thread1 yang berhasil dibuat. parameter kedua berisi NULL karena menggunakan variabel default. Parameter yang ketiga digunakan untuk menunjuk fungsi yang akan dikerjakan oleh thread1. Pada parameter keempat, digunakan untuk mengarahkan ke fungsi yang akan di threaded. Karena berisi NULL maka tidak akan mengarah.
30) pthread_create adalah sebuah fungsi yang digunakan untuk membuat sebuah thread baru. fungsi ini memiliki 4 parameter. Parameter yang pertama berfungsi sebagai nilai kembalian berupa thread id dari thread2 yang berhasil dibuat. parameter kedua berisi NULL karena menggunakan variabel default. Parameter yang ketiga digunakan untuk menunjuk fungsi yang akan dikerjakan oleh thread2. Pada parameter keempat, digunakan untuk mengarahkan ke fungsi yang akan di threaded. Karena berisi NULL maka tidak akan mengarah.
31) –
32) /* tanda pembuka komentar untuk : Wait till threads are complete before main continues. Unless we. */ tanda penutup komentar
33) /* tanda pembuka komentar untuk : wait we run the risk of executing an exit which will terminate. */ tanda penutup komentar
34) /* tanda pembuka komentar untuk : the process and all threads before the threads have completed.. */ tanda penutup komentar
35) –
36) pthread_join digunakan untuk melakukan wait sebuah thread untuk dihentikan, kemudian dua parameter tersebut adalah, parameter pertama adalah thread id dari thread yang sedang melakukan waiting. Kemudian parameter kedua adalah NULL.
37) pthread_join digunakan untuk melakukan wait sebuah thread untuk dihentikan, kemudian dua parameter tersebut adalah, parameter pertama adalah thread id dari thread yang sedang melakukan waiting. Kemudian parameter kedua adalah NULL.
38) –
39) exit digunakan untuk keluar dari shell dimana ia sedang berjalan. Pada baris ini, proses akan keluar dan mengembalikan nilai 0.
40) Syntax untuk menutup deklarasi fungsi main().
41) –
42) Pendefinisian fungsi void bernama fuction1().
43) Syntax untuk membuka deklarasi fungsi void *functio1().
44) pthread_mutex_lock digunakan untuk mengunci object mutex yang direferensikan dengan mutex. Dan dalam kasus ini adalah variabel mutex1. Jika mutex sudah

dikunci, maka pthread_mutex_lock akan menghalanginya hingga mutex tersedia kembali. Operasi ini kembali dengan object mutex yang direferensikan oleh mutex yang ada dalam kondisi terkunci dengan calling thread sebagai pemiliknya.

45) Terdapat pemanggilan sistem call sleep() dengan parameter berupa angka dalam satuan second (waktu). Dalam hal ini, parameternya 5 detik, tujuannya adalah untuk menghentikan program secara sementara selama 5 detik.
46) pthread_mutex_lock digunakan untuk mengunci object mutex yang direferensikan dengan mutex. Dan dalam kasus ini adalah variabel mutex2. Jika mutex sudah dikunci, maka pthread_mutex_lock akan menghalanginya hingga mutex tersedia kembali. Operasi ini kembali dengan object mutex yang direferensikan oleh mutex yang ada dalam kondisi terkunci dengan calling thread sebagai pemiliknya.
47) Nilai variable counter ditambahkan dengan 1.
48) Printf akan menampilkan “Counter value in function1: %d\n“ dengan %d adalah return value dari counter yang bertipe data int.
49) pthread_mutex_unlock digunakan untuk melepaskan object mutex yang direferensikan oleh mutex. Dan membuat mutex kembali tersedia. Dalam baris ini adalah variabel mutex2.
50) pthread_mutex_unlock digunakan untuk melepaskan object mutex yang direferensikan oleh mutex. Dan membuat mutex kembali tersedia. Dalam baris ini adalah variabel mutex1.
51) Syntax untuk menutup deklarasi fungsi void *function1().
52) –
53) Pendefinisian fungsi void bernama fuction2().
54) Syntax untuk membuka deklarasi fungsi void *functio2().
55) pthread_mutex_lock digunakan untuk mengunci object mutex yang direferensikan dengan mutex. Dan dalam kasus ini adalah variabel mutex2. Jika mutex sudah dikunci, maka pthread_mutex_lock akan menghalanginya hingga mutex tersedia kembali. Operasi ini kembali dengan object mutex yang direferensikan oleh mutex yang ada dalam kondisi terkunci dengan calling thread sebagai pemiliknya.
56) Terdapat pemanggilan sistem call sleep() dengan parameter berupa angka dalam satuan second (waktu). Dalam hal ini, parameternya 5 detik, tujuannya adalah untuk menghentikan program secara sementara selama 5 detik.
57) pthread_mutex_lock digunakan untuk mengunci object mutex yang direferensikan dengan mutex. Dan dalam kasus ini adalah variabel mutex1. Jika mutex sudah dikunci, maka pthread_mutex_lock akan menghalanginya hingga mutex tersedia kembali. Operasi ini kembali dengan object mutex yang direferensikan oleh mutex yang ada dalam kondisi terkunci dengan calling thread sebagai pemiliknya.
58) Nilai variable counter ditambahkan dengan 1.
59) Printf akan menampilkan “Counter value in function1: %d\n“ dengan %d adalah return value dari counter yang bertipe data int.
60) pthread_mutex_unlock digunakan untuk melepaskan object mutex yang direferensikan oleh mutex. Dan membuat mutex kembali tersedia. Dalam baris ini adalah variabel mutex1.

61) pthread_mutex_unlock digunakan untuk melepaskan object mutex yang direferensikan oleh mutex. Dan membuat mutex kembali tersedia. Dalam baris ini adalah variabel mutex2.
62) Syntax untuk menutup deklarasi fungsi void *function2().

8. Deadlock Prevention

a. Source Code

```
1.  #include <stdio.h>
2.  #include <stdlib.h>
3.  #include <pthread.h>
4.  #include <unistd.h>
5.
6.  void *function1();
7.  void *function2();
8.  pthread_mutex_t mutex1 = PTHREAD_MUTEX_INITIALIZER;
9.  pthread_mutex_t mutex2 = PTHREAD_MUTEX_INITIALIZER;
10. int  counter = 0;
11.
12. int main()
13. {
14.    int rc1, rc2;
15.    pthread_t thread1, thread2;
16.
17.    /* Create independent threads each of which will execute functionC */
18.
19.    if( (rc1=pthread_create( &thread1, NULL, &function1, NULL)) )
20.    {
21.       printf("Thread creation failed: %d\n", rc1);
22.    }
23.
24.    if( (rc2=pthread_create( &thread2, NULL, &function2, NULL)) )
25.    {
26.       printf("Thread creation failed: %d\n", rc2);
27.    }
28.
29.    /* Wait till threads are complete before main continues. Unless we  */
30.    /* wait we run the risk of executing an exit which will terminate   */
31.    /* the process and all threads before the threads have completed.   */
32.
33.    pthread_join( thread1, NULL);
34.    pthread_join( thread2, NULL);
35.
36.    exit(0);
37. }
38.
39. void *function1()
40. {
41.    pthread_mutex_lock( &mutex1 );
42.     sleep(3);
43.     while ( pthread_mutex_trylock( &mutex2 ) )
44.     {
45.         pthread_mutex_unlock(&mutex1);
46.         printf("Deadlock terhindari\n");
47.         pthread_mutex_lock(&mutex1);
48.     }
49.    counter++;
50.    printf("Counter value: %d\n",counter);
51.     pthread_mutex_unlock( &mutex2 );
52.    pthread_mutex_unlock( &mutex1 );
```

```
53. }
54. 
55. void *function2()
56. {
57.     pthread_mutex_lock(&mutex2);
58.     sleep(3);
59.     while ( pthread_mutex_trylock( &mutex1 ) )
60.     {
61.         pthread_mutex_unlock(&mutex2);
62.         printf("Deadlock terhindari\n");
63.         pthread_mutex_lock(&mutex2);
64.     }
65.     counter++;
66.     printf("Counter value: %d\n", counter);
67.     pthread_mutex_unlock(&mutex1);
68.     pthread_mutex_unlock(&mutex2);
69. }
```

b. Analisis

1) #include<stdio.h> merupakan file header yang berisi deklarasi fungsi-fungsi dasar untuk membuat program C
2) #include <stdlib.h> Merupakan file header yang berfungsi untuk operasi pembanding dan operasi konversi.
3) #include <pthread.h> digunakan agar kita dapat menggunakan fungsi - fungsi dari library pthread. Dan untuk mengeksekusi file tersebut, maka kita harus menggunakan -pthread atau -lpthread dalam command line
4) Unistd.h adalah sebuah library yang menyediakan fungsi sleep()
5) -
6) deklarasi fungsi *function1();
7) deklarasi fungsi *function1();
8) deklarasi mutex1 dengan tipe pthread_mutex_t yang berisi konstanta PTHREAD_MUTEX_INITIALIZER.
9) deklarasi mutex2 dengan tipe pthread_mutex_t yang berisi konstanta PTHREAD_MUTEX_INITIALIZER.
10) Mendeklarasikan variabel counter bernilai 0 dengan tipe data int.
11) -
12) main() adalah fungsi utama dalam program, dimana fungsi ini akan dieksekusi pertama kali saat program dijalankan.
13) Syntax untuk membuka deklarasi fungsi main().
14) Mendeklarasikan variable rc1 dan rc2 dengan tipe data int.
15) Mendeklarasikan variable thread1 dan thread2 dengan tipe data pthread_t. pthread_t adalah sebuah tipe data yang digunakan untuk mengidetifikasi sebuah thread
16) -
17) Memberikan comment, membuat independent threads yang masing-masing akan menjalankan fungsi.
18) -
19) pthread_create adalah sebuah fungsi yang digunakan untuk membuat sebuah thread baru. fungsi ini memiliki 4 parameter. Parameter yang pertama berfungsi sebagai nilai kembalian berupa thread id dari thread yang

berhasil dibuat. parameter kedua berisi NULL karena menggunakan variabel default. Parameter yang ketiga digunakan untuk menunjuk fungsi yang akan dikerjakan oleh thread. Pada parameter keempat, digunakan untuk mengarahkan ke fungsi yang akan di threaded. Karena berisi NULL maka tidak akan mengarah.

20) Syntax untuk membuka kondisi pertama.
21) Printf akan menampilkan "Thread creation failed: %d\n" dengan %d adalah return value dari rc1 yang bertipe data int.
22) Syntax untuk menutup kondisi pertama.
23) -
24) pthread_create adalah sebuah fungsi yang digunakan untuk membuat sebuah thread baru. fungsi ini memiliki 4 parameter. Parameter yang pertama berfungsi sebagai nilai kembalian berupa thread id dari thread yang berhasil dibuat. parameter kedua berisi NULL karena menggunakan variabel default. Parameter yang ketiga digunakan untuk menunjuk fungsi yang akan dikerjakan oleh thread. Pada parameter keempat, digunakan untuk mengarahkan ke fungsi yang akan di threaded. Karena berisi NULL maka tidak akan mengarah.
25) Syntax untuk membuka selain kondisi pertama.
26) Printf akan menampilkan "Thread creation failed: %d\n" dengan %d adalah return value dari rc2 yang bertipe data int.
27) Syntax untuk menutup selain kondisi pertama.
28) -
29) Memberikan comment, menunggu threads selesai, sebelum melanjutkan proses utama. Kecuali jika kita tidak,
30) Memberikan comment, menunggu kita beresiko keluar dari proses yang dieksekusi yang akan,
31) Memberikan comment, menghentikan proses dan semua threads sebelum threads selesai.
32) -
33) pthread_join digunakan untuk melakukan wait sebuah thread untuk dihentikan, kemudian dua parameter tersebut adalah, parameter pertama adalah thread id dari thread yang sedang melakukan waiting. Kemudian parameter kedua adalah exit status dari thread yang melakukan waiting.
34) pthread_join digunakan untuk melakukan wait sebuah thread untuk dihentikan, kemudian dua parameter tersebut adalah, parameter pertama adalah thread id dari thread yang sedang melakukan waiting. Kemudian parameter kedua adalah exit status dari thread yang melakukan waiting.
35) -
36) exit digunakan untuk keluar dari shell dimana ia sedang berjalan. Pada baris ini, proses akan keluar dan mengembalikan nilai 0.
37) Syntax untuk menutup deklarasi fungsi main().
38) -
39) Pendefinisian fungsi void bernama function1().
40) Syntax untuk membuka deklarasi fungsi void *functio1n().
41) pthread_mutex_lock digunakan untuk mengunci object mutex yang direferensikan dengan mutex. Dan dalam kasus ini adalah variabel mutex1.

Jika mutex sudah dikunci, maka pthread_mutex_lock akan menghalanginya hingga mutex tersedia kembali. Operasi ini kembali dengan object mutex yang direferensikan oleh mutex yang ada dalam kondisi terkunci dengan calling thread sebagai pemiliknya.

42) Sleep() adalah fungsi untuk melakukan pemberhentian program secara sementara dengan argument berupa int (bisa berupa angka) sebagai waktu lamanya dalam satuan detik
43) Pengecekan kondisi dengan while. Yaitu, mutex2 di cek apakah sudah di kunci dengan fungsi **pthread_mutex_trylock**. Jika iya kerjakan baris 44-46
44) Pembuka statement eksekusi while
45) Mutex 1 di unlock untuk membebaskan *resource* agar menghindari deadlock dengan fungsi **pthread_mutex_unlock**
46) Mencetak string sebagai penanda apabila deadlock berhasil di hindari
47) Mengunci mutex1 kembali dengan fungsi **pthread_mutex_lock**
48) Penutup statement eksekusi while
49) Nilai variable counter ditambahkan dengan 1.
50) Printf akan menampilkan "Counter value: %d\n" dengan %d adalah return value dari counter yang bertipe data int.
51) pthread_mutex_unlock digunakan untuk melepaskan object mutex2 yang direferensikan oleh mutex. Dan membuat mutex kembali tersedia.
52) pthread_mutex_unlock digunakan untuk melepaskan object mutex1 yang direferensikan oleh mutex. Dan membuat mutex kembali tersedia.
53) Syntax untuk menutup deklarasi fungsi void *function1().
54) -
55) Pendefinisian fungsi void bernama function2().
56) Syntax untuk membuka deklarasi fungsi void *function2().
57) pthread_mutex_lock digunakan untuk mengunci object mutex yang direferensikan dengan mutex. Dan dalam kasus ini adalah variabel mutex2. Jika mutex sudah dikunci, maka pthread_mutex_lock akan menghalanginya hingga mutex tersedia kembali. Operasi ini kembali dengan object mutex yang direferensikan oleh mutex yang ada dalam kondisi terkunci dengan calling thread sebagai pemiliknya.
58) Sleep() adalah fungsi untuk melakukan pemberhentian program secara sementara dengan argument berupa int (bisa berupa angka) sebagai waktu lamanya dalam satuan detik
59) Pengecekan kondisi dengan while. Yaitu, mutex1 di cek apakah sudah di kunci dengan fungsi pthread_mutex_trylock. Jika iya kerjakan baris 60-62
60) Pembuka statement eksekusi while
61) Mutex 2 di unlock untuk membebaskan resource agar menghindari deadlock dengan fungsi pthread_mutex_unlock
62) Mencetak string sebagai penanda apabila deadlock berhasil di hindari
63) Mengunci mutex2 kembali dengan fungsi pthread_mutex_lock
64) Penutup statement eksekusi while
65) Nilai variable counter ditambahkan dengan 1.
66) Printf akan menampilkan "Counter value: %d\n" dengan %d adalah return value dari counter yang bertipe data int.

67) pthread_mutex_unlock digunakan untuk melepaskan object mutex1 yang direferensikan oleh mutex. Dan membuat mutex kembali tersedia.
68) pthread_mutex_unlock digunakan untuk melepaskan object mutex2 yang direferensikan oleh mutex. Dan membuat mutex kembali tersedia.
69) Syntax untuk menutup deklarasi fungsi void *function2().

Analisis Program Prevent Deadlock

Sederhananya, program ini mencoba menghindari deadlock dengan mengecek dlu setelah proses sleep apakah mutex yang ingin dikunci sudah terkunci atau belum, jika sudah maka kita perlu *unlock* mutex yang sudah dikunci sebelumnya untuk membbebaskan *resource* untuk menghindari deadlock.