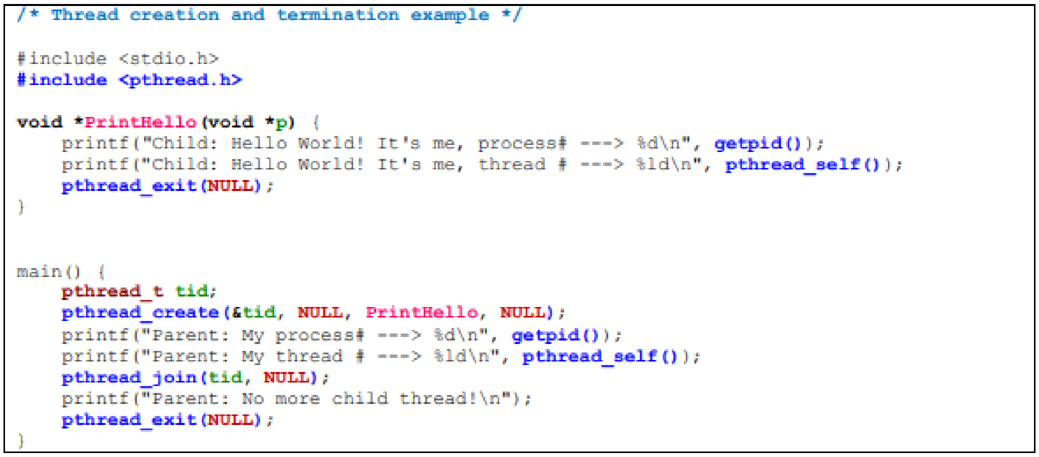
Abdullah Baabdullah 1936975 EE463 Lab6

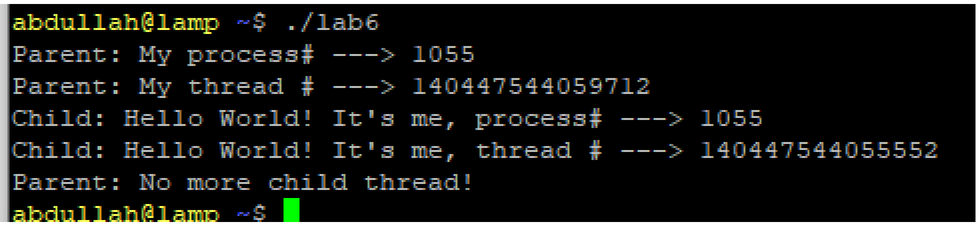
Examples:

1. The following program demonstrates thread creation and termination:



```
/* Thread creation and termination example */

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

void *PrintHello(void *p) {
    printf("Child: Hello World! It's me, process# ---> %d\n", getpid());
    printf("Child: Hello World! It's me, thread # ---> %ld\n", pthread_self());
    pthread_exit(NULL);
}


main() {
    pthread_t tid;
    pthread_create(&tid, NULL, PrintHello, NULL);
    printf("Parent: My process# ---> %d\n", getpid());
    printf("Parent: My thread # ---> %ld\n", pthread_self());
    pthread_join(tid, NULL);
    printf("Parent: No more child thread!\n");
    pthread_exit(NULL);
}
```

1) Run the above program, and observe its output:

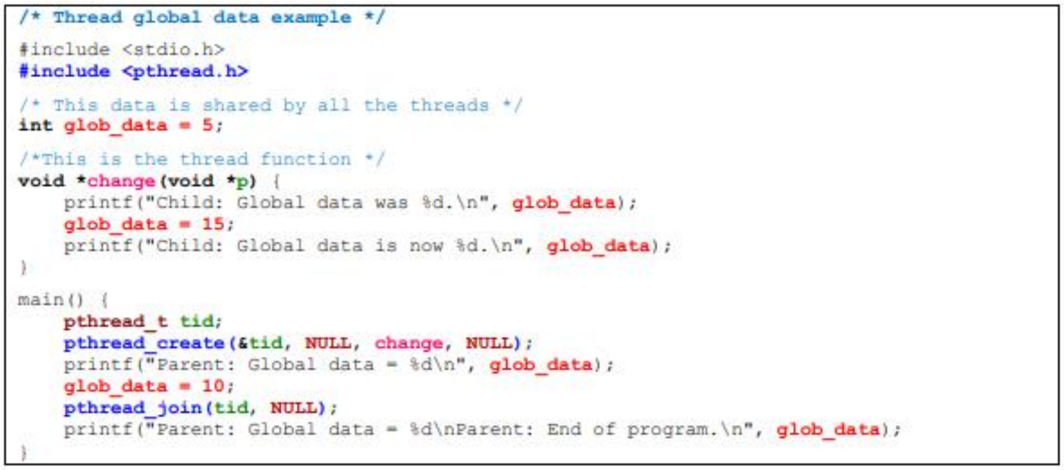


```
abdullah@lamp ~$ ./lab6
Parent: My process# ---> 1055
Parent: My thread # ---> 140447544059712
Child: Hello World! It's me, process# ---> 1055
Child: Hello World! It's me, thread # ---> 140447544055552
Parent: No more child thread!
abdullah@lamp ~$
```

2) Are the process ID numbers of parent and child threads the same or different? Why?

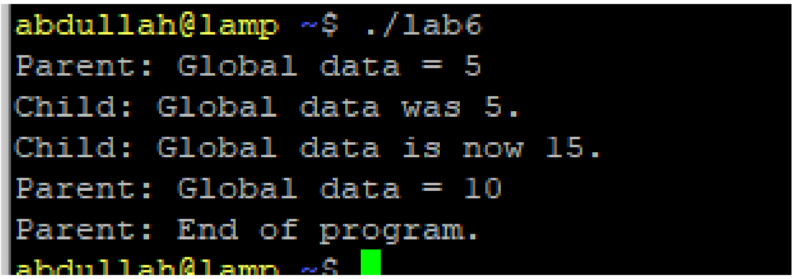
It’s the same because the other threads share their memory with the thread.

2. The following program demonstrates thread global data:



```
/* Thread global data example */
#include <stdio.h>
#include <pthread.h>
/* This data is shared by all the threads */
int glob_data = 5;
/*This is the thread function */
void *change(void *p) {
    printf("Child: Global data was %d.\n", glob_data);
    glob_data = 15;
    printf("Child: Global data is now %d.\n", glob_data);
}
main() {
    pthread_t tid;
    pthread_create(&tid, NULL, change, NULL);
    printf("Parent: Global data = %d\n", glob_data);
    glob_data = 10;
    pthread_join(tid, NULL);
    printf("Parent: Global data = %d\nParent: End of program.\n", glob_data);
}
```

3) Run the above program several times; observe its output every time. A sample output follows:



```
abdullah@lamp ~$ ./lab6
Parent: Global data = 5
Child: Global data was 5.
Child: Global data is now 15.
Parent: Global data = 10
Parent: End of program.
abdullah@lamp ~$
```

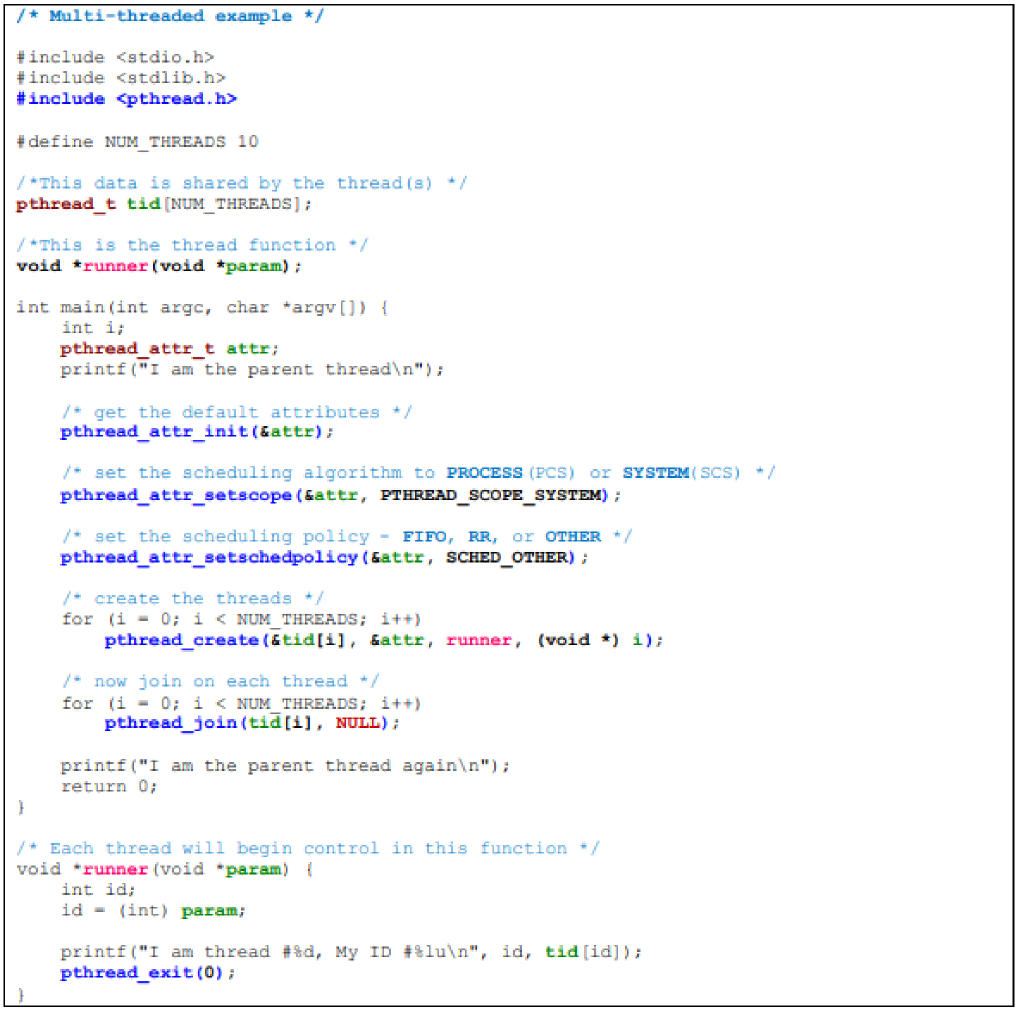
1) Does the program give the same output every time? Why?

No it does not, because the program involves concurrent execution with multiple threads, and the order of execution and timing of operations between threads can vary.

2) Do the threads have separate copies of `glob_data`?

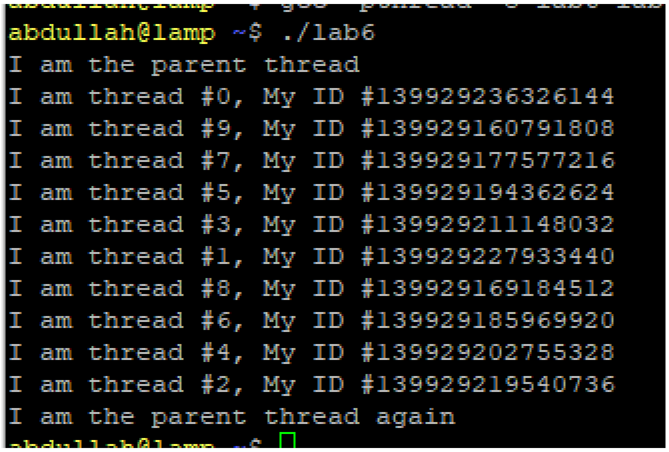
No it does not, because The variable ‘glob_data’ is a global variable defined outside any function, which makes it accessible to all threads in the program. All threads share the same memory space, including global variables.

3. The following example demonstrates a multi-threaded program:



```
/* Multi-threaded example */

#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>

#define NUM_THREADS 10

/*This data is shared by the thread(s) */
pthread_t tid[NUM_THREADS];

/*This is the thread function */
void *runner(void *param);

int main(int argc, char *argv[]) {
    int i;
    pthread_attr_t attr;
    printf("I am the parent thread\n");

    /* get the default attributes */
    pthread_attr_init(&attr);

    /* set the scheduling algorithm to PROCESS(PCS) or SYSTEM(SCS) */
    pthread_attr_setscope(&attr, PTHREAD_SCOPE_SYSTEM);

    /* set the scheduling policy - FIFO, RR, or OTHER */
    pthread_attr_setschedpolicy(&attr, SCHED_OTHER);

    /* create the threads */
    for (i = 0; i < NUM_THREADS; i++)
        pthread_create(&tid[i], &attr, runner, (void *) i);

    /* now join on each thread */
    for (i = 0; i < NUM_THREADS; i++)
        pthread_join(tid[i], NULL);

    printf("I am the parent thread again\n");
    return 0;
}

/* Each thread will begin control in this function */
void *runner(void *param) {
    int id;
    id = (int) param;

    printf("I am thread #%d, My ID #%lu\n", id, tid[id]);
    pthread_exit(0);
}
```

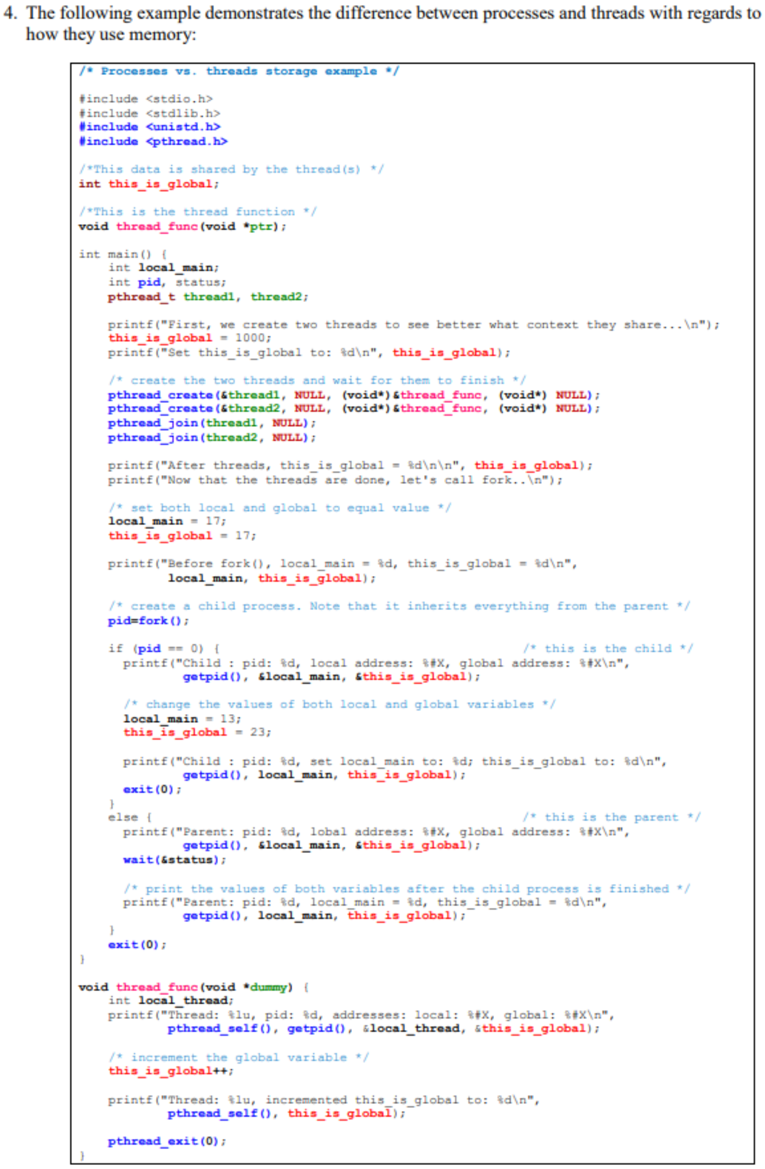
6) Run the above program several times and observe the outputs:



```
abdullah@lamp ~$ gcc -pthread -o lab6 lab
abdullah@lamp ~$ ./lab6
I am the parent thread
I am thread #0, My ID #139929236326144
I am thread #9, My ID #139929160791808
I am thread #7, My ID #139929177577216
I am thread #5, My ID #139929194362624
I am thread #3, My ID #139929211148032
I am thread #1, My ID #139929227933440
I am thread #8, My ID #139929169184512
I am thread #6, My ID #139929185969920
I am thread #4, My ID #139929202755328
I am thread #2, My ID #139929219540736
I am the parent thread again
abdullah@lamp ~$
```

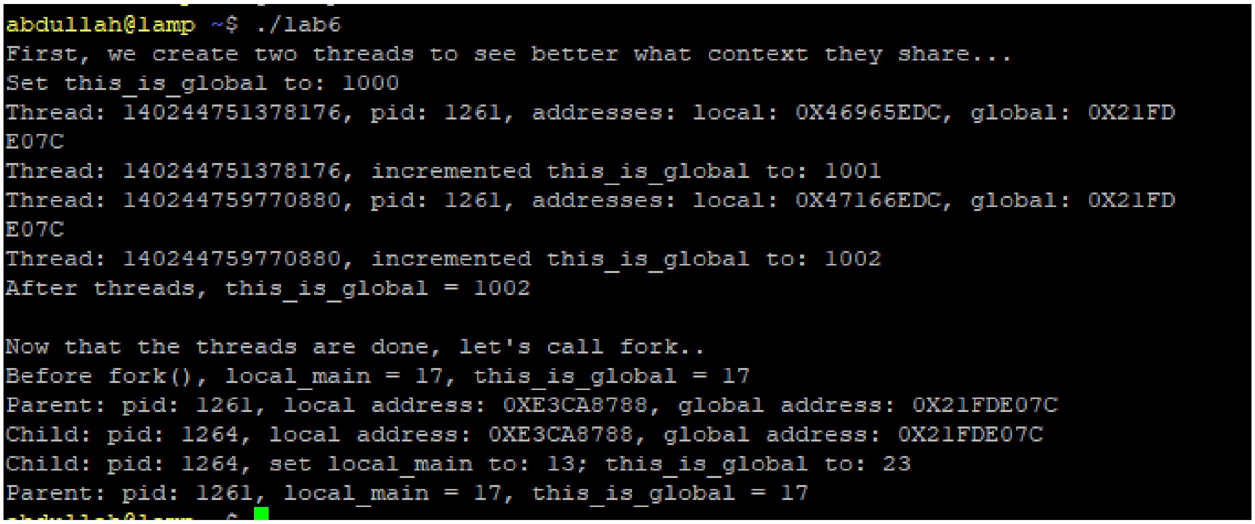
7) Do the output lines come in the same order every time? Why?

No, because the concurrent execution of multiple threads.



4. The following example demonstrates the difference between processes and threads with regards to how they use memory:

```
/* Processes vs. threads storage example */

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <pthread.h>

/*This data is shared by the thread(s) */
int this_is_global;

/*This is the thread function */
void thread_func(void *ptr);

int main() {
    int local_main;
    int pid, status;
    pthread_t thread1, thread2;

    printf("First, we create two threads to see better what context they share...\n");
    this_is_global = 1000;
    printf("Set this_is_global to: %d\n", this_is_global);

    /* create the two threads and wait for them to finish */
    pthread_create(&thread1, NULL, (void*)&thread_func, (void*) NULL);
    pthread_create(&thread2, NULL, (void*)&thread_func, (void*) NULL);
    pthread_join(thread1, NULL);
    pthread_join(thread2, NULL);

    printf("After threads, this_is_global = %d\n\n", this_is_global);
    printf("Now that the threads are done, let's call fork..\n");

    /* set both local and global to equal value */
    local_main = 17;
    this_is_global = 17;

    printf("Before fork(), local_main = %d, this_is_global = %d\n",
            local_main, this_is_global);

    /* create a child process. Note that it inherits everything from the parent */
    pid=fork();

    if (pid == 0) {                                        /* this is the child */
      printf("Child : pid: %d, local address: %#X, global address: %#X\n",
              getpid(), &local_main, &this_is_global);

      /* change the values of both local and global variables */
      local_main = 13;
      this_is_global = 23;

      printf("Child : pid: %d, set local_main to: %d; this_is_global to: %d\n",
              getpid(), local_main, this_is_global);
      exit(0);
    }
    else {                                                 /* this is the parent */
      printf("Parent: pid: %d, lobal address: %#X, global address: %#X\n",
              getpid(), &local_main, &this_is_global);
      wait(&status);

      /* print the values of both variables after the child process is finished */
      printf("Parent: pid: %d, local_main = %d, this_is_global = %d\n",
              getpid(), local_main, this_is_global);
    }
    exit(0);
}

void thread_func(void *dummy) {
    int local_thread;
    printf("Thread: %lu, pid: %d, addresses: local: %#X, global: %#X\n",
            pthread_self(), getpid(), &local_thread, &this_is_global);

    /* increment the global variable */
    this_is_global++;

    printf("Thread: %lu, incremented this_is_global to: %d\n",
            pthread_self(), this_is_global);

    pthread_exit(0);
}
```



```
abdullah@lamp ~$ ./lab6
First, we create two threads to see better what context they share...
Set this_is_global to: 1000
Thread: 140244751378176, pid: 1261, addresses: local: 0X46965EDC, global: 0X21FD
E07C
Thread: 140244751378176, incremented this_is_global to: 1001
Thread: 140244759770880, pid: 1261, addresses: local: 0X47166EDC, global: 0X21FD
E07C
Thread: 140244759770880, incremented this_is_global to: 1002
After threads, this_is_global = 1002

Now that the threads are done, let's call fork..
Before fork(), local_main = 17, this_is_global = 17
Parent: pid: 1261, local address: 0XE3CA8788, global address: 0X21FDE07C
Child: pid: 1264, local address: 0XE3CA8788, global address: 0X21FDE07C
Child: pid: 1264, set local_main to: 13; this_is_global to: 23
Parent: pid: 1261, local_main = 17, this_is_global = 17
```

9) Did **`this_is_global`** change after the threads have finished? Why?

Yes, because It depends on the relative timing and interleaving of the threads' execution.

10) Are the local addresses the same in each thread? What about the global addresses?

the local addresses may vary across threads, while the global addresses should be the same in each thread and in the main program.

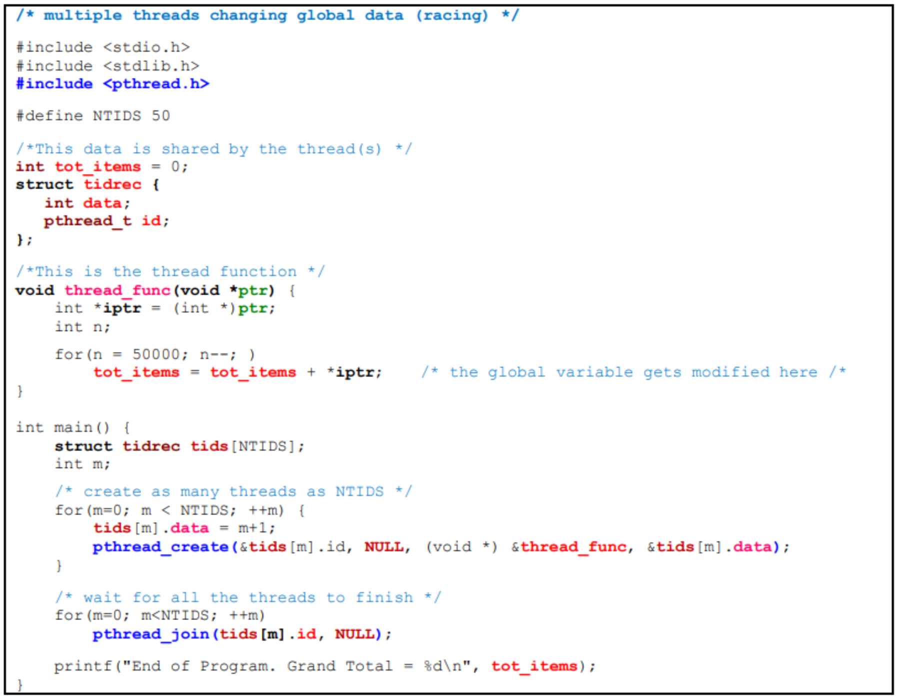
11) Did **`local_main`** and **`this_is_global`** change after the child process has finished? Why?

No, because the values of 'local_main' and 'this_is_global' in the parent process do not change after the child process has finished because the child process operates in a separate memory space.

12) Are the local addresses the same in each process? What about global addresses? What happened?

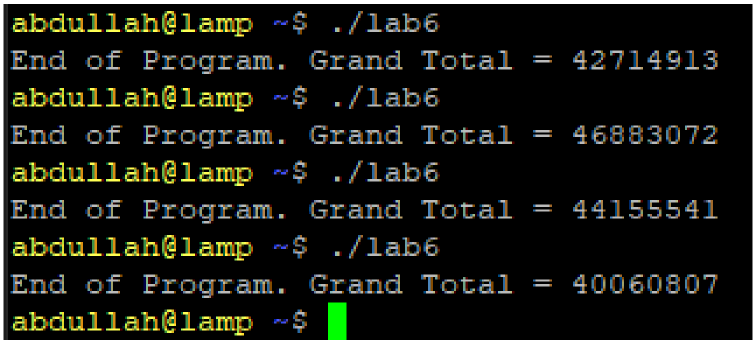
The local addresses may vary across processes, while the global addresses should be the same in both the parent and child processes.

5. The following example demonstrates what happens when multiple threads try to change global data:



```
/* multiple threads changing global data (racing) */

#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>

#define NTIDS 50

/*This data is shared by the thread(s) */
int tot_items = 0;
struct tidrec {
   int data;
   pthread_t id;
};

/*This is the thread function */
void thread_func(void *ptr) {
    int *iptr = (int *)ptr;
    int n;

    for(n = 50000; n--; )
        tot_items = tot_items + *iptr;    /* the global variable gets modified here /*
}

int main() {
    struct tidrec tids[NTIDS];
    int m;

    /* create as many threads as NTIDS */
    for(m=0; m < NTIDS; ++m) {
        tids[m].data = m+1;
        pthread_create(&tids[m].id, NULL, (void *) &thread_func, &tids[m].data);
    }

    /* wait for all the threads to finish */
    for(m=0; m<NTIDS; ++m)
        pthread_join(tids[m].id, NULL);

    printf("End of Program. Grand Total = %d\n", tot_items);
}
```

13) Run the above program several times and observe the outputs, until you get different results.



```
abdullah@lamp ~$ ./lab6
End of Program. Grand Total = 42714913
abdullah@lamp ~$ ./lab6
End of Program. Grand Total = 46883072
abdullah@lamp ~$ ./lab6
End of Program. Grand Total = 44155541
abdullah@lamp ~$ ./lab6
End of Program. Grand Total = 40060807
abdullah@lamp ~$
```

14) How many times the line tot_items = tot_items + *iptr; is executed?

50,000 times for each of the 50 threads which is equal to 2.5 million executions of the line.

15) What values does *iptr have during these executions?

during the executions of the line, the values of *iptr will range from 1 to 50, corresponding to the data values of each thread.

16) What do you expect Grand Total to be?

By using this formula: Sum = (first term + last term) * number of terms / 2
Sum = (1 + 50) * 50,000 / 2
= 51 * 50,000 / 2
= 2,550,000
The grand total will be 2.55 million.

17) Why you are getting different results?

Due to the lack of proper synchronization mechanisms.