King Abdulaziz University Faculty of Engineering

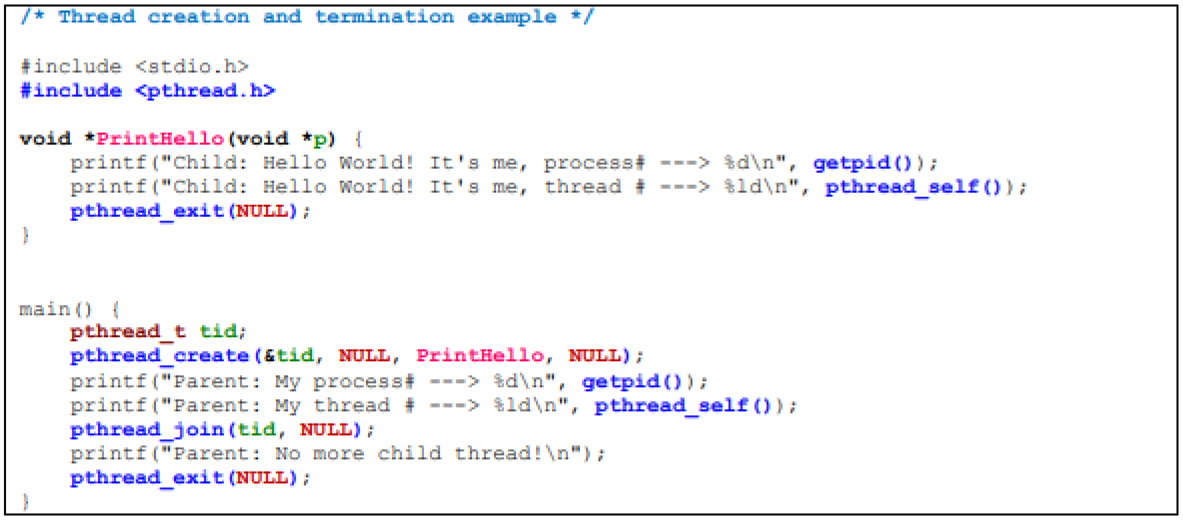
Lab (6)
Multi-Threading

Operating Systems– EE463

Lecturer(s): Dr. Abdulghani M. Al-Qasimi
& Eng. Turki Abdul Hafeez
3rd Semester Spring 2023
Section: C3 (Thursday 13 to 14:40)

#	Name	ID
1	Moad Abdulaali Alshikh	1937343

Exercises:



```c
/* 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:

Parent: My process# ---> 1090

Parent: My thread # ---> 140586219689792

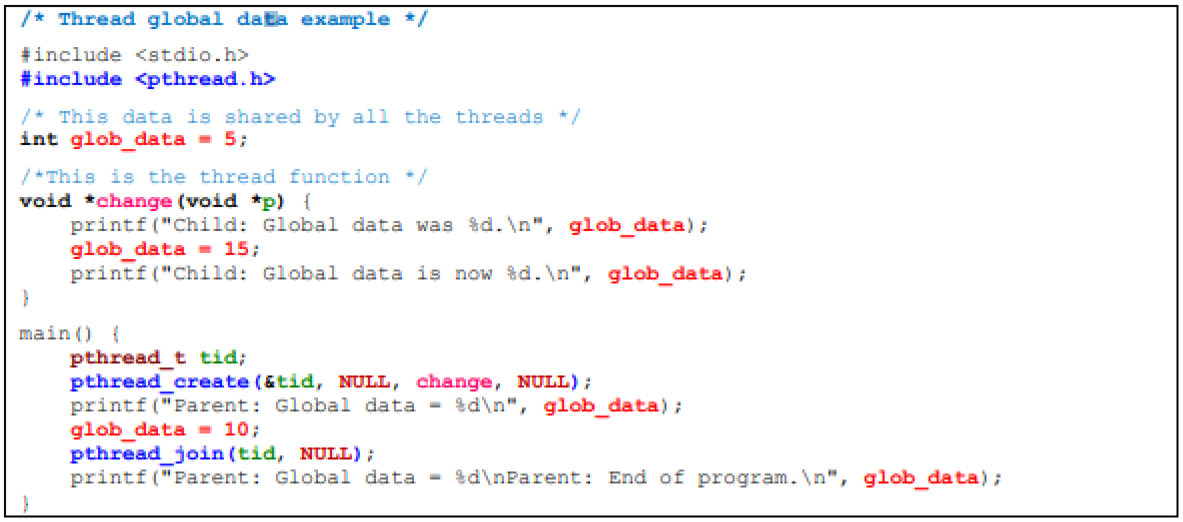
Child: Hello World! It's me, process# ---> 1090

Child: Hello World! It's me, thread # ---> 140586219685632

Parent: No more child thread!

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

The process ID numbers of the parent and child threads are the same because they belong to the same process. Threads share the same process address space, so they have the same process ID.



```c
/* 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:**

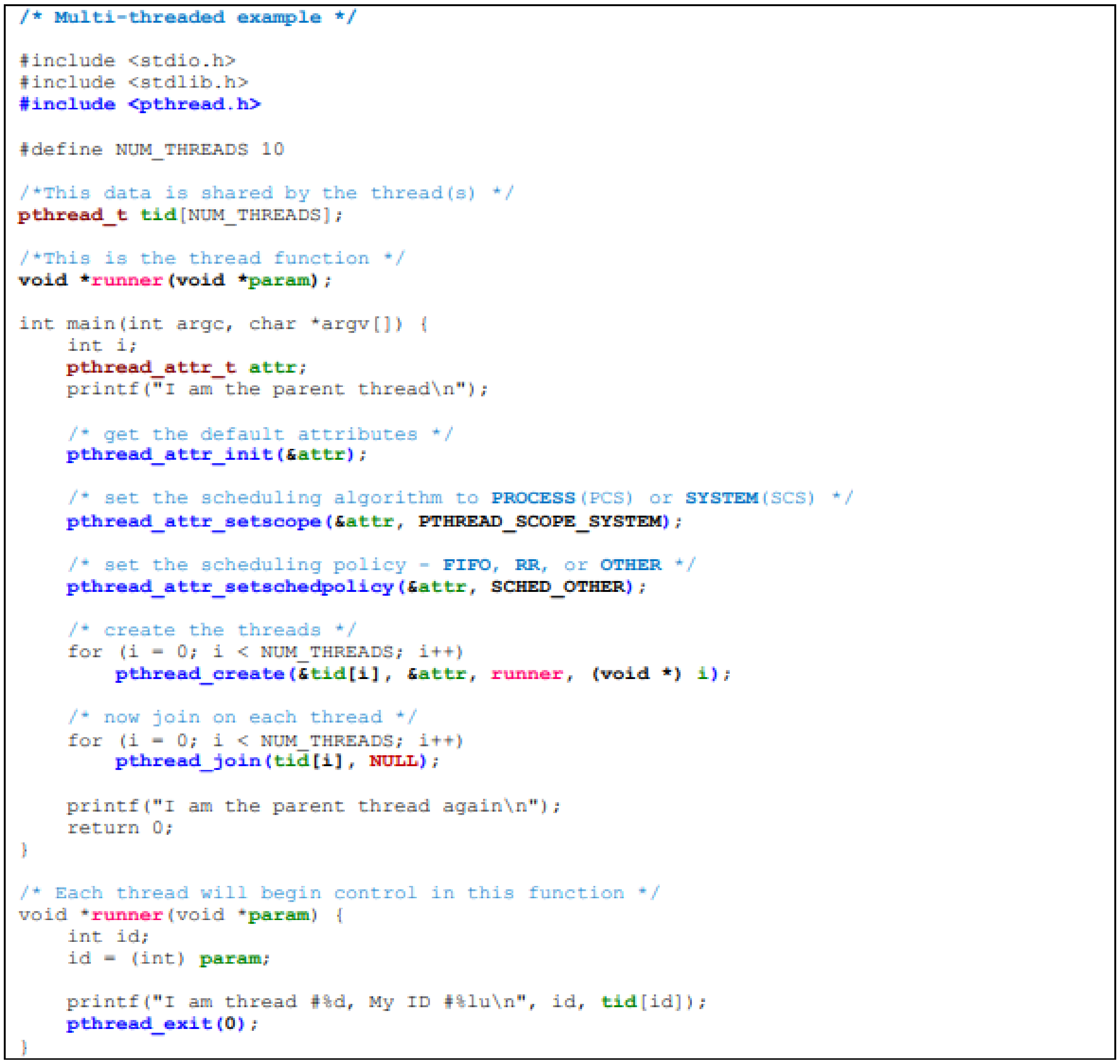
```
Parent: Global data = 5
Child: Global data was 10.
Child: Global data is now 15.
Parent: Global data = 15
Parent: End of program.
```

4- **Does the program give the same output every time? Why?**

The program doesn't give the same output every time because the order in which the threads are executed is not deterministic. This depends on the operating system's thread scheduler.

5- **Do the threads have separate copies of glob_data?**

The threads do not have separate copies of glob_data. It is a global variable, so all threads share the same instance of it.



```c
/* 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:

```
I am the parent thread
I am thread #3, My ID #139919837026048
I am thread #1, My ID #139919853811456
I am thread #0, My ID #139919862204160
I am thread #2, My ID #139919845418752
I am thread #4, My ID #139919828633344
I am thread #5, My ID #139919820240640
I am thread #6, My ID #139919811741440
I am thread #7, My ID #139919803348736
I am thread #9, My ID #139919786563328
I am thread #8, My ID #139919794956032
I am the parent thread again
```

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

The output lines do not come in the same order every time because the order in which threads are executed depends on the operating system's thread scheduler, which is not deterministic.

```c
/* 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);
}
```

8- Run the above program and observe its output. Following is a sample output:

```
First, we create two threads to see better what context they share...
Set this_is_global to: 1000
Thread: 140702550734592, pid: 1168, addresses: local: 0XDD8E2EDC, global:
0XC1DC807C
Thread: 140702550734592, incremented this_is_global to: 1001
Thread: 140702559127296, pid: 1168, addresses: local: 0XDE0E3EDC, global:
0XC1DC807C
Thread: 140702559127296, 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: 1168, lobal address: 0XA6BB0A38, global address: 0XC1DC807C
Child : pid: 1171, local address: 0XA6BB0A38, global address: 0XC1DC807C
Child : pid: 1171, set local_main to: 13; this_is_global to: 23
Parent: pid: 1168, local_main = 17, this_is_global = 17
```

9- Did this_is_global change after the threads have finished? Why?

this_is_global did change after the threads have finished, because both threads incremented it.

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

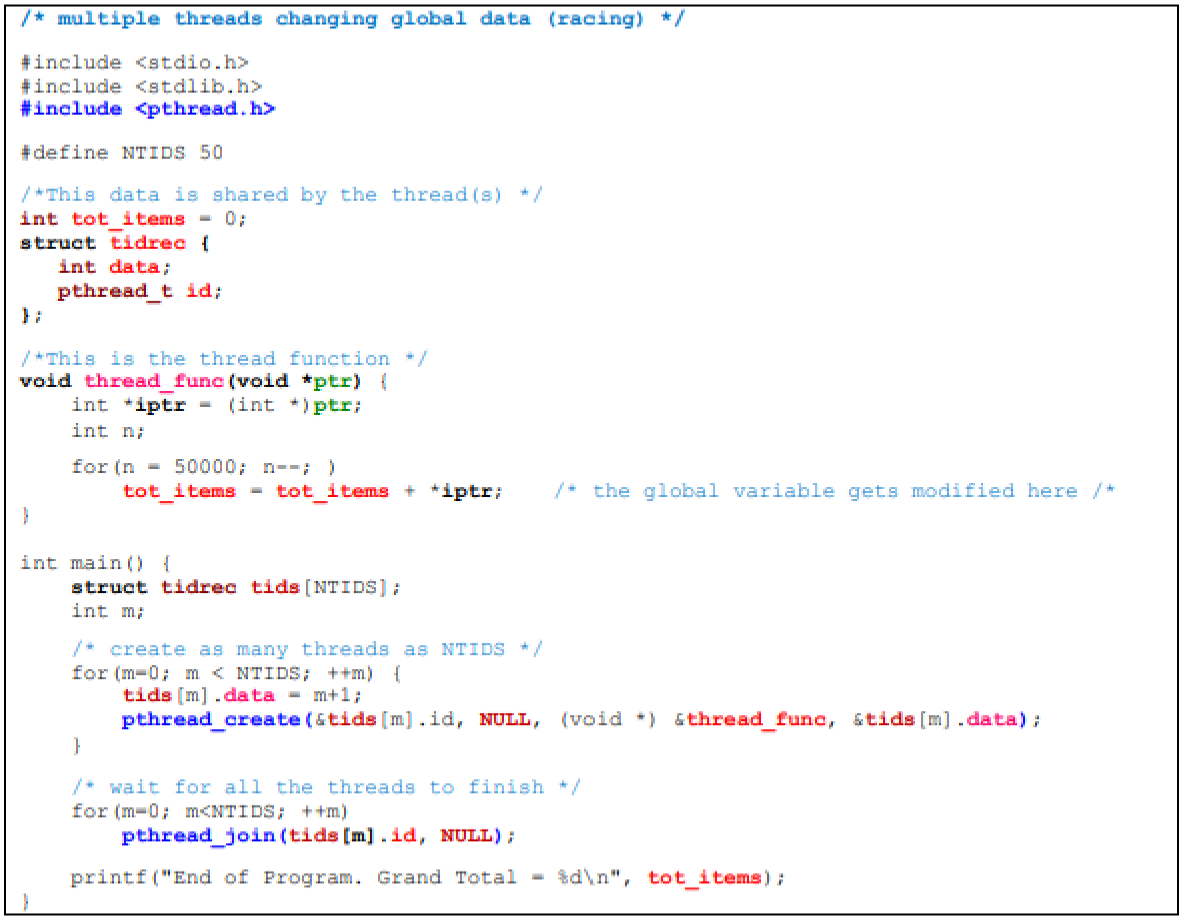
The local addresses are different in each thread, as each thread has its own stack. The global addresses are the same because all threads share the same process address space.

11- Did local_main and this_is_global change after the child process has finished? Why?

local_main and this_is_global did not change after the child process has finished because child processes have their own copy of the parent's memory space, and changes made in the child process do not affect the parent.

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

The local addresses are the same in each process, because they are copied from the parent process to the child process. The global addresses are also the same, because they are also copied from the parent process to the child process. However, changes made in the child process do not affect the parent process, as they have separate memory spaces.



```c
/* 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.

```
End of Program. Grand Total = 63750000
End of Program. Grand Total = 63750000
End of Program. Grand Total = 63750000
End of Program. Grand Total = 63750000
End of Program. Grand Total = 63749998
```

The output of the fifth program is changed due to the race condition caused by multiple threads modifying the global variable tot_items.

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

The line tot_items = tot_items + *iptr; is executed 50 (NTIDS) * 50,000 = 2,500,000 times.

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

*iptr takes on values from 1 to 50 (inclusive) during these executions.

16- What do you expect Grand Total to be?

We might expect the Grand Total to be the sum of 1 to 50 (inclusive) multiplied by 50,000, which is 1,275,000,000, but due to the race condition, the actual result may be different.

17- Why you are getting different results?

We are getting different results because there is a race condition where multiple threads are modifying the global variable tot_items without any synchronization, leading to unpredictable results.