**CG2271 Real Time Operating Systems**

**Lab 7 – Threads**

**Answer Book**

|  |  |
| --- | --- |
| Member 1 Name: Vu Ngoc Quang | Matric No: A0099878W |
| Member 2 Name: Prachi Singhal | Matric No: A0103527N |

**Question 1**

The threads print out of order. It is multi-threading so all the ten threads are being executed at the same time. So it’s uncertain which thread finishes first. Hence, they the threads do not finish in order and the values printed are out of order.

**Question 2**

The threads share memory. Initially the value of ctr is 0. For example, if two threads are being executed, thread one takes this value and stores it locally on its own stack. When thread two pre-empts thread one, it takes the value on its own stack, then increments the value by one. Then thread one increments the value by one and hence, both thread one and thread two print out 1 as the ctr value. We observe that the values of ctr are printed arbitrarily. This shows that the threads share common memory as the value is changed randomly.

**Question 3**

The values of ctr as printed by the threads are wrong. The reason is they share the same memory space as ctr is declared as a global variable. Based on the answer of question one, it is uncertain which thread finishes first.

**Question 4**

The variable "i" must be cast into void \* because the function child only accept a parameter of type (void\*), not (int).

In child it does not have to be cast back into int because the variable i now becomes the address value (not the value the pointer points to) of type integer. Therefore, printf will now print the address of type int from 0 to 9.

**Question 5**

My code is attached here:

for(i=0; i<10; i++){

pthread\_create(&thread[i], NULL, child, (void \*) i);

// wait for the i-th thread

pthread\_join(thread[i], NULL);

}

The changes I made are: include the pthread\_join inside the for loop. This is so that after each thread is created, the program has to wait for the function in that thread to finish executing before the next thread is created. Therefore, all threads will now execute in order.

**Question 6**

The value of glob printed by main is 20.

**Question 7**

The changes we made are:

pthread\_t thread[10]; //declare an array of 10 threads to be created

…

/\* in function child

Add this line at the end

pthread\_exit(NULL);

\*/

printf("Child %d exiting. Glob is currently %d\n", t, glob);

pthread\_exit(NULL);

\*/

/\* in function main

replace the line

child((void \*) i);

by

pthread\_create(&thread[i], NULL, child, (void \*) i);

\*/

for(i=0; i<10; i++)

pthread\_create(&thread[i], NULL, child, (void \*) i);

**Question 8**

The value printed is incorrect. This is because the final value of glob is 9 instead of 20. Furthermore, glob is supposed to increase by 2 after each thread. However, during execution, at some instances, glob doesn’t change value or only increases by 1. This is because, there is a race condition of all the threads on the global value of glob, resulting in conflict.

**Question 9**

The threads still update glob incorrectly. This is because after spawning all 10 threads, the main thread may destroy the mutex and end execution before all the 10 threads finish execution, resulting in wrong value of glob.

**Question 10**

The change we add in is to make the main thread to wait for all the 10 threads to finish executing before destroying the mutex and printing out the final value of glob.

The codes in the main are changed as followed:

int main()

{

int i;

pthread\_t thread[10];

glob=0;

for(i=0; i<10; i++){

pthread\_create(&thread[i], NULL, child, (void \*) i);

}

//wait for all threads to finish

for(i=0; i<10; i++) {

pthread\_join(thread[i], NULL);

}

printf("Final value of glob is %d\n", glob);

pthread\_mutex\_destroy(&mutex);

return 0;

}