**CS2106 Operating Systems**

**Assignment 2 – Processes and Threads**

**Answer Book**

|  |  |
| --- | --- |
| Member 1 Name: Ram Janarthan | Member 1 Matric No: A0147620L |
| Member 2 Name: Advay Pal | Member 2 Matric No: A0144939R |

Question 1

This is what I see on the screen: Parent sent message: Hello child! and 128

My single statement description is: This program allows a parent process to send a message to child process.

Question 2

The sizeof function returns the size (in bytes) of the argument passed to it

Question 3

My completed code is attached below:

#include <stdio.h>

#include <math.h>

#include <time.h>

#include <stdlib.h>

#define NUMELTS 16384

// IMPORTANT: Compile using "gcc assg2p2.c -lm -o assg2p2".

// The "-lm" is important as it brings in the Math library.

// Implement the naive primality test.

// Returns TRUE if n is a prime number

int prime(int n) {

int ret = 1, i;

for (i = 2; i <= (int) sqrt(n) && ret; i++)

ret = (n % i);

return ret;

}

int main() {

int data[NUMELTS];

int fd[2];

pipe(fd);

// Declare other variables here.

// Create the random number list.

srand(time(NULL));

for(int i=0; i<NUMELTS; i++)

data[i] = (int) (((double) rand () / (double) RAND\_MAX) \* 10000);

if(fork()) {

int count = 0;

int input;

for(int j = 0; j < 8192; j ++) {

if(prime(data[j]))

count ++;

}

close(fd[1]);

read(fd[0], &input, sizeof(input));

printf("Number of primes: %d", input + count);

} else {

int count\_child = 0;

for(int j = 8192; j < 16384; j++) {

if(prime(data[j]))

count\_child ++;

}

close(fd[0]);

write(fd[1], &count\_child, sizeof(count\_child));

}

return 0;

}

Question 4

The threads print out of order. The reason is that the creation of a thread does not guarantee that it is executed immediately after that, so the threads can get pre-empted in a random order.

Question 5

The threads do share memory. Referring to ctr, I conclude this because the value of ctr in different cases is not just 0/1, but also 2, 3...8, 9. This means that the same ctr is incremented multiple times, and shows that the threads share memory.

Question 6

The values of ctr as printed by the threads are wrong. The reason is sometimes the threads are pre-empted before they can increment ctr.

Question 7

The variable "i" must be cast into void \* because the argument type for the start routine can be anything, and this is supported by simply passing a pointer holding the address of the argument, as opposed to passing the argument itself.

In child it does not have to be cast back into int because the ‘%d’ in the printf statement casts it to an int. It is also known that the parent never passes anything other than an int.

Question 8

The changes I made are to include a ‘pthread\_join’ call to the ‘i’th thread immediately after it has been created. This ensures that no thread ‘i’ gets created before the ‘i-1’th thread has been executed.

My code is attached here: (On line 25)

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

Question 9

The value of glob printed at the end of main is 20

Question 10

The changes we made are as follows:  
Change child((void \*) i); to   
pthread\_create(&thread[i], NULL, child, (void \*) i);

Question 11

The values of glob are incorrect because all threads share the same glob variable and threads are pre-empted at random points, so we cannot guarantee that one thread manages to increment glob before it is pre-empted. Furthermore, we see that no process manages to reach its exit print statement, which we attribute to the fact that it sleeps for 1s, which means it is still asleep by the time the main function is done.

Question 12

The threads still update glob incorrectly; with the only difference that glob is either 0 or 1. This is because a thread may get pre-empted before it locks the mutex, or after it locks the mutex and increments glob. If it gets pre empted after locking the mutex, but not before unlocking, no other thread can increment glob. Again, we don’t see any thread unlocking the mutex, which we attribute to 1s being too long a duration.

Question 13

The changes we made were as follows:  
1. Delete all lines dealing with the mutex  
2. Add a pthread\_join statement immediately after the pthread\_create statement

Our program is attached below:

At Line 21, we add:

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