**CS2106 Operating Systems**

**Assignment 2 – Processes and Threads**

**Answer Book**

|  |  |
| --- | --- |
| Member 1 Name: Wang Riwu | Member 1 Matric No: A0135766W |
| Member 2 Name: | Member 2 Matric No: |

Question 1

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

My single statement description is: Creates a child process and send 2 values to the child using pipes for the child to print it to stdout.

Question 2

The sizeof function returns the size of a variable or datatype in bytes.

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.

// Implements 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];

// Declare other variables here.

int fd[2];

pipe(fd);

int i, count=0;

// Create the random number list.

srand(time(NULL));

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

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

// Now create a parent and child process.

if(fork())

{

//PARENT:

// Check the 0 to 8191 sub-list

for (i=0; i<NUMELTS/2; i++)

if (prime(data[i])) count++;

// Then wait for the prime number count from the child.

close(fd[1]);

int child\_count;

read(fd[0], &child\_count, sizeof(child\_count));

// Parent should then print out the number of primes

// found by it, number of primes found by the child,

// And the total number of primes found.

printf("Number of primes found by parent: %d\n", count);

printf("Number of primes found by child: %d\n", child\_count);

printf("Number of primes found by both: %d\n", count+child\_count);

}

else

{

// CHILD:

// Check the 8192 to 16383 sub-list.

for (i=NUMELTS/2; i<NUMELTS; i++)

if (prime(data[i])) count++;

// Send # of primes found to the parent.

close(fd[0]);

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

}

}

Question 4

The threads print out of order. The reason is threads may get pre-empted by other threads before they finish executing.

Question 5

The threads do share memory. Referring to ctr, I conclude this because value of ctr can increment to up to 9, which would not be possible if the threads each have their own portions of memory. Hence they must be sharing memory.

Question 6

The values of ctr as printed by the threads are wrong. The reason is that the threads might be pre-empted by other threads before the increment of ctr is executed. Furthermore the value ctr could be subjected to race condition, resulting in some threads not correctly incrementing the value.

Question 7

The variable "i" must be cast into void \* because that’s the data type of the parameter of pthread\_create, hence it must be explicitly cast to the data type that matches the parameter type of the function.

In child it does not have to be cast back into int because it is not holding the address of the integer, but the integer itself. This is because the argument is ‘(void \*) i’, rather than &i.

The printf function with “%d” format specifier then cast the four byte data back into integer.

Question 8

The changes I made are calling pthread\_join after pthread\_create to wait for it to finish executing.

My code is attached here:

#include <stdio.h>

#include <pthread.h>

// Global variable.

int ctr=0;

pthread\_t thread[10];

void \*child(void \*t)

{

// Print out the parameter passed in, and the current value of ctr.

printf("I am child %d. Ctr=%d\n", t, ctr);

// Then increment ctr

ctr++;

pthread\_exit(NULL);

}

int main()

{

int i;

// Initialize ctr

ctr=0;

// Create the threads

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

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

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

}

// And print out ctr

printf("Value of ctr=%d\n", ctr);

return 0;

}

Question 9

The value of glob printed by main is 20.

Question 10

The changes we made are

1. declare a variable pthread\_t thread[10]
2. change child((void \*) i); to pthread\_create(&thread[i], NULL, child, (void \*) i);

Question 11

The value printed is incorrect. This is because the printf statement in main sometimes executes before all the threads finish executing. Also the variable glob can be subjected to race condition as the threads might attempt to modify it concurrently.

Question 12

The threads now update glob correctly. This is because before the threads goes into modifying the glob variable, they are blocked until they successfully obtains the mutex lock, at which point they will prevent other threads from successfully obtaining the same mutex lock until they finish modifying the glob variable and frees the mutex lock, hence preventing race conditions.

However the value of glob printed by main is still incorrect as printf in main might still execute before all the threads finish executing.

Question 13

The changes we made were adding

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

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

before the printing of glob in main to wait for all the threads to finish executing.

Our program is attached below:

#include <stdio.h>

#include <pthread.h>

pthread\_mutex\_t mutex=PTHREAD\_MUTEX\_INITIALIZER;

int glob;

void \*child(void \*t)

{

// Increment glob by 1, wait for 1 second, then increment by 1 again.

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

pthread\_mutex\_lock(&mutex);

glob++;

sleep(1);

glob++;

pthread\_mutex\_unlock(&mutex);

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

}

int main()

{

int i;

glob=0;

pthread\_t thread[10];

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

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

//wait for all threads to finish executing

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;

}