**CS2106 Operating Systems**

**Assignment 2 – Processes and Threads**

**Answer Book**

|  |  |
| --- | --- |
| Member 1 Name:  Zheng Weihan | Member 1 Matric No:  A0097582N |
| Member 2 Name:  Tse Hong Ming | Member 2 Matric No:  A0148808U |

Question 1

This is what I see on the screen:

Parent sent message: Hello child! and 128

My single statement description is:

The program forks a process, creates a pipe from the parent to child, and passes messages from the parent to child.

Question 2

The sizeof function returns …

Size, in bytes, of an object(struct) or type that is passed as argument to the function.

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];

int fd[2];

pipe(fd);

// Declare other variables here.

// Create the random number list.

srand(time(NULL));

int i;

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

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

// 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.

close(fd[1]);

int count = 0;

int i;

int childCount;

for(i = 0; i<8192; i++){

if(prime(data[i])){

count++;

}

}

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

printf("%d\n", childCount+count);

// count = 0;

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

// if(prime(data[i])){

// count++;

// }

// }

// printf("actual count: %d\n", count);

}else{

close(fd[0]);

int count = 0;

int i;

for(i =8192; i < NUMELTS; i++){

if(prime(data[i])){

count++;

}

}

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

}

// CHILD:

// Check the 8192 to 16383 sub-list.

// Send # of primes found to the parent.

}

Question 4

The threads print out of order. The reason is it is up to the scheduler to decide which thread to run and therefore no guarantee on order.

Question 5

The threads do share memory. Referring to ctr, I conclude this because ctr’s number are incremented beyond 0, which would be the case if no memory is shared. Since ctr is initialized to 0, if threads do not share memory, then each ctr value in each thread would have ctr value of 0. But since we have some threads which has ctr value of more than 0, it means this thread’s ctr value is incremented by other threads, indicating shared memory.

Question 6

The values of ctr as printed by the threads are wrong. The reason is because of preemption of older threads which has not finished its execution. New threads then are run, and read in the older value, leading to wrong value of ctr.

Question 7

The variable "i" must be cast into void \* because POSIX standard specified it so. The last argument is specified to be a pointer to an array of arguments. The arguments can be of any type, therefore, it is cast to void.

In child it does not have to be cast back into int because t is a “void pointer”, which means t can point to any type of variable. The compiler is also smart enough to auto-cast it to int for us, when we use it with the `%d` flag.

Question 8

The changes I made are adding `pthread\_join(thread[i], NULL); ` after `pthread\_create` within the loop. This ensures one thread finishes its execution before the next thread is created.

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 replace the `child call` in the ‘for loop’ to

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

Question 11

The value printed is incorrect. This is because of race condition.`glob++` is not an atomic operation and can be preempted. In some cases, the preemption causes glob to read the wrong/write the wrong value.

Question 12

The threads now update glob correctly. This is because the mutex ensures that only 1 thread has access to the glob variable at any point in time, thus ensuring that when the thread is in its critical section, no other threads can run. Thus, the glob variable is updated correctly.

Question 13

The changes we made were adding another loop iterating over `pthread\_join(thread[i], NULL);` for all the threads before the printing the final blob value.

Our program is attached below:

#include <stdio.h>

#include <pthread.h>

int glob;

pthread\_t thread[10];

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

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;

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

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

}

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;

}