**CS2106 Operating Systems**

**Assignment 2 – Processes and Threads**

**Answer Book**

|  |  |
| --- | --- |
| Member 1 Name:  Yeo Xiang Guang Brandon | Member 1 Matric No:  A0135782Y |
| 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: The program forks out into 2 processes. The parent writes to the child the message of “Hello Child” and number 128. The child reads the message and prints it out.

Question 2

The sizeof function returns the number of bytes requred to meausre the datatype of that variable.

Question 3

My completed code is attached below:

#include <stdio.h>

#include <math.h>

#include <time.h>

#include <stdlib.h>

#include <unistd.h>

#define NUMELTS 16384

#define IN 0

#define OUT 1

// 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;

}

/\*\*

\* Counts the number of prime numbers in an array

\*

\* @param startIdx starting index of the array.

\* @param lenToRead sizeof the array to look at.

\* @return count of prime numbers in the list.

\*/

int primeCounter(int \*numbers, int startIdx, int lenToRead) {

int count = 0, i;

for (i = startIdx; i < lenToRead; i++) {

if (prime(numbers[i])) {

count += 1;

}

}

return count;

}

/\*\*

\* Parent will print out the total prime count including the child's count.

\*

\* @param fd controls the pipe for parent to read child number.

\* @param data contains data to be analysed for prime.

\*/

void parentPrintOutTotalPrimeCount(int \*fd, int \*data) {

int cnum = 0;

close(fd[OUT]);

int primeCounts = primeCounter(data, 0, (NUMELTS/2));

read(fd[IN], &cnum, sizeof(cnum));

close(fd[IN]);

primeCounts += cnum;

printf("Total prime count = %d\n", primeCounts);

}

/\*\*

\* Child will send the parent its prime count.

\*

\* @param fd controls the pipe for child to send to parent.

\* @param data contains data to be analysed for prime.

\*/

void childSendParentPrimeCount(int \*fd, int \*data) {

close(fd[IN]);

int primeCounts = primeCounter(data, NUMELTS/2, NUMELTS);

write(fd[OUT], &primeCounts, sizeof(primeCounts));

close(fd[OUT]);

}

int main() {

int data[NUMELTS], i;

srand(time(NULL));

pid\_t pid = -1;

// Declare other variables here.

int fd[2];

if (pipe(fd) == -1) {

perror("Unable to pipe properly\n");

exit(EXIT\_FAILURE);

}

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

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

}

pid = fork();

if (pid > 0) {

parentPrintOutTotalPrimeCount(fd, data);

} else if (pid == 0) {

childSendParentPrimeCount(fd, data);

} else {

perror("Unable to fork");

exit(EXIT\_FAILURE);

}

exit(EXIT\_SUCCESS);

}

Question 4

The threads print out of order. The reason is that the threads finishes running at different times. The threads are not force to wait for each other.

Question 5

The threads do share memory. Referring to ctr, I conclude this because the ctr was added to about the value of 8 or 9. If they do not share memory the ctr should be the value 1.

Question 6

The values of ctr as printed by the threads are incorrect. The reason is that the threads are not running sequentially, some threads run before the others which causes the ctr to be incorrectly printed by child threads.

Question 7

The variable "i" must be cast into void \* because the argument for pthread\_create requires void \* pointer.

In child it does not have to be cast back into int because void \* pointer points to raw data. The printf function uses that raw data and displays it in the integer format based on the format specified.

Question 8

The changes I made are adding pthread\_join(thread[i],NULL) so that thread 0 executes first and then thread 1 and so on.

My code is attached he#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

pthread\_join(thread[9], NULL);

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 to add pthread\_create(&thread[i], NULL, child, (void \*) i) in the for loop to create 10 threads.

Question 11

The value printed is incorrect. This is because the threads are entering into each others’ critical section thus causing a race condition. The sleep function also affects the thread from executing the rest of the code as the wake up signal maybe lost.

Question 12

The threads now update glob incorrectly. When the first thread runs, it locks the mutex and increments the glob value. It then goes to sleep. If the wake up signal is lost than the sleep function causes that thread from not executing the rest of its code thus not releasing the mutex, the other thread cannot increment glob as a result.

Question 13

The changes we made were to add pthread\_barrier\_t barrier.

Initialising it with pthread\_barrier\_init(&barrier, NULL, 11), for 10 thread and 1 more main thread to synchronise statement “printf("Final value of glob=%d\n", glob);" to be printed last.

pthread\_barrier\_wait(&barrier) is also used in the child func and also in the main func.

Our program is attached below:

#include <stdio.h>

#include <pthread.h>

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

int glob;

pthread\_t thread[10];

pthread\_barrier\_t barrier;

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);

pthread\_barrier\_wait(&barrier);

}

int main()

{

int i;

glob=0;

pthread\_barrier\_init(&barrier, NULL, 11);

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

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

}

pthread\_barrier\_wait(&barrier);

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

pthread\_mutex\_destroy(&mutex);

pthread\_barrier\_destroy(&barrier);

return 0;

}