CS2106 Operating Systems

Lab 3 - Threads

Member 1 Name: Joshua Lee Kai Sheng	Member 1 Matric No: A0161868R
Member 2 Name: Ng Jun Wei	Member 2 Matric No: A0155156E
Member 3 Name: Ryan Tan Wei Keat	Member 3 Matric No: A0161551M

Question 1 (5 marks)

My completed code is attached below:

```c
#include <stdio.h>
#include <math.h>
#include <time.h>
#include <stdlib.h>
#include <unistd.h>

#define NUMELTS 16384

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];

        //create pipe
        int fd[2];
        pipe(fd);

        //number of prime number from child
        int numPrimeChild;

        //generate random numbers
        srand (time(NULL));

        for(int i=0; i<NUMELTS; i++)
                data[i]=(int) (((double) rand() / (double) RAND_MAX) * 10000);
        if (fork() != 0) { //parent

                //number of prime number from parent
                int numPrimeParent = 0;

                //total number of prime number
                int totalPrime = 0;

                //calculate number of prime from parent
                for(int i=0; i<NUMELTS/2; i++) {
                        if (prime(data[i]))
                                numPrimeParent++;
                }


                close(fd[1]);//parent will read from pipe

                while(1) {
                        //if pipe is not empty
                        if (read(fd[0], &numPrimeChild, sizeof(numPrimeChild)) > 0) {
                                //print the results
                                printf("Number of prime number from parent is %d\n",
numPrimeParent);

                                printf("Number of prime number from child is %d\n",
numPrimeChild);

                                totalPrime = numPrimeParent + numPrimeChild;//update total
prime

                                printf("Total number of prime number is %d\n",totalPrime);
                                break;
                        }
                }

                close(fd[1]);

        } else {//child
                close(fd[0]);

                //calculate number of prime from child
                for(int i=NUMELTS/2; i<NUMELTS; i++) {
                        if(prime(data[i]))
                                numPrimeChild++;
                }

                //write number of prime from child to pipe
                write(fd[1], &numPrimeChild, sizeof(numPrimeChild));

                close(fd[1]);
        }

}
```

Question 2 (2 marks)

The threads print out of order. The reason is because of concurrency or parallelism. The threads are created around the same time in the loop. Hence, during the execution of each thread, it might get suspended to give way to another thread due to scheduling policies for multitasking.

Hence, an earlier thread might print later if it gets suspended by the scheduler before the printf statement and a subsequent/later thread gets to execute the printf statement.

Such scheduling is indeterministic and hence the threads are likely to print out of order.

Question 3 (2 marks)

The threads do share memory. Referring to ctr, I conclude this because the ctr, as a global variable is shared among the threads. This is manifested when the ctr is incremented continuously along the way by the different threads.

Question 4 (3 marks)

The values of ctr as printed by the threads are wrong (not in unique incremental order). The reason is because of race condition.

The printf and ctr++ statements executed in each thread are translated to assembly code during runtime.

Since the threads share the same memory space and execute "concurrently", they can have access to the same variable (even before it is supposed to be updated).

For example, after the first thread prints Ctr=0 and gets suspended by the scheduler, the next thread executed will have access to the same ctr global variable, which is not yet updated to 1. Hence, this next thread will also print Ctr=0, which is not what is expected (Ctr=1). In other words, the ctr values might not have been updated at the point when they are printed.

Another possible scenario is during the ctr++ statement. The ctr value will first be loaded into a register. When a thread is suspended after loading the ctr value into a register and the subsequent thread executed load the same value of ctr into a register, both of these threads will increment on the same ctr value and hence will update the ctr to the same value. This will affect the final ctr value printed (less than 10). However, there is a low possibility where a thread will be suspended right after loading ctr into a register as the surrounding statements are likely to be executed in one cycle (take little time).

Question 5 (2 marks)

The variable "i" must be cast into void * because the argument data type for function *child is void *

In child it does not have to be cast back into int because during printing, "i" is declared (implicitly casted) as an int by using "%d".

Question 6 (3 marks)

My code and explanation:

```
// Maximum number of connections
#define MAX_CONNECTIONS  1000

//void * function for connection thread
void *child(void *i)
{
        writeLog("Connection received.");
        deliverHTTP(i);
}

void startServer(uint16_t portNum)
{
        static int listenfd, connfd;
        static struct sockaddr_in serv_addr;
        pthread_t thread[MAX_CONNECTIONS];//initialise an array of thread type

        listenfd = socket(AF_INET, SOCK_STREAM, 0);

        if(listenfd<0)
        {
                perror("Unable to make socket.");
                exit(-1);
        }

        memset(&serv_addr, 0, sizeof(serv_addr));

        serv_addr.sin_family=AF_INET;
        serv_addr.sin_addr.s_addr = htonl(INADDR_ANY);
        serv_addr.sin_port = htons(portNum);

        if(bind(listenfd, (struct sockaddr *) &serv_addr, sizeof(serv_addr))<0)
        {
                perror("Unable to bind.");
                exit(-1);
        }

        if(listen(listenfd, 10)<0)
        {
                perror("Unable to listen.");
                exit(-1);
        }

        writeLog("Web server started at port number %d", portNum);

        //for the number of maximum connections
        for (int i = 0; i < MAX_CONNECTIONS; i++)
        {
                //create the client socket
                connfd = accept(listenfd, (struct sockaddr *) NULL, NULL);
                //create the thread for connection
                pthread_create(&thread[i], NULL, child, (void*)connfd);
                //detach the thread so resources will be released automatically and
                //we don't have to wait for it to terminate using pthread_join()
                pthread_detach(thread[i]);
        }
}
```

Create a constant, MAX_CONNECTIONS to signify the maximum number of threads that can be created for connection. In start server, create an array of thread type. In a for loop, create a thread for connection after accepting and creating the client socket. The thread will handle writeLog() as well as deliverHTTP().

The pthread_detach() function marks the thread identified by thread as detached. When a detached thread terminates, its resources are automatically released back to the system, without needing the main function that calls pthread_create() to wait for the thread to terminate, like using pthread_join().

<u>Question 7</u> (3 marks)

My code and explanation:

```
int main(int ac, char **av)
{
        logfptr = fopen("webserver.log", "w");
        if(logfptr == NULL)
        {
                fprintf(stderr, "Cannot open log file\n");
                exit(-1);
        }

        fclose(logfptr);

        int *i;

        //create a thread for logger
        pthread_t loggerThread;
        pthread_create(&loggerThread, NULL, log, (void *) i);

        startServer(PORTNUM);
}

//void * function for thread
void *log(void *i)
{
        //an infinite loop
        while(1) {
                //if logReady is set
                if(logReady) {

                        //read from buffer and write to file
                        FILE* logfptr = fopen("webserver.log", "a");
                        fputs(logBuffer, logfptr);
                        fflush(logfptr);
                        fclose(logfptr);

                        //reset logReady
                        logReady = 0;

                }
        }
}
```

Create a new thread for logging in the main method and run an infinite loop to check logReady. If logReady is set, write from the logBuffer to the log file.

TOTAL: ________________/20