**CS2106 Operating Systems**

**Lab 3 - Threads**

|  |  |
| --- | --- |
| Member 1 Name: Quek Yang Sheng | Member 1 Matric No: A0126400Y |
| Member 2 Name: Ang Wei Ming | Member 2 Matric No: A0168721B |
| Member 3 Name: | Member 3 Matric No: |

Question 1 (5 marks)

My completed code is attached below:

|  |
| --- |
| int main() {  int i;     int data[NUMELTS];     // Declare other variables here.     // 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.  int cpid;  int half = NUMELTS/2;  int numPrimes = 0, numPrimesChild = 0;  int status;  int fd[2];  pipe(fd);  char buffer[1024];  if((cpid = fork()) !=0){    //PARENT:  close(fd[1]);     for(i=0; i<half; i++){      numPrimes += prime(data[i]) ? 1 : 0;     }  } else{  // CHILD:  close(fd[0]);      for(i=half; i<NUMELTS; i++){      numPrimes += prime(data[i]) ? 1 : 0;      }  write(fd[1], &numPrimes, sizeof(numPrimes));  close(fd[1]);     exit(0);  }    wait(&status);  **read(fd[0], &numPrimesChild, sizeof(numPrimesChild));**  **close(fd[0]);**  printf("Number of Parent Primes: %d\n", numPrimes);  printf("Number of Child Primes: %d\n", numPrimesChild);  numPrimes += numPrimesChild;    printf("Number of Primes: %d\n", numPrimes);  } |

Question 2 (2 marks)

The threads print out of order. The reason is due to the scheduling policy of the Linux OS.

Question 3 (2 marks)

The threads do share memory. Referring to ctr, I conclude this because if the memory are not shared, then all the output of ctr will be 0 since ctr only increments after the printf statement. Based on observation through multiple runs of the program, there are child that prints the correct total value of ctr regardless of the running order.

Question 4 (3 marks)

The values of ctr as printed by the threads are correct. The reason is the value of ctr read by the child at the point in time is correct before it is being scheduled away by the OS.

Question 5 (2 marks)

The variable "i" must be cast into void \* because pthread\_create only accepts a void \* argument type and casting “i” into a void pointer ***(void \*)*** merely changes “i” type to a void pointer with value of “i”. The value of “i” remains unchanged.

In child it does not have to be cast back into int because void \* already holds the original value of “i”, so we are just conveniently using the pointer value as the original int value.

Question 6 (3 marks)

My code and explanation:

|  |
| --- |
| void startServer(uint16\_t portNum)  {  ...        ...        ...  writeLog("Web server started at port number %d", portNum);  while(1)  {  connfd = accept(listenfd, (struct sockaddr \*) NULL, NULL);  writeLog("Connection received.");  pthread\_t newThread;  pthread\_create(&newThread, NULL, deliverHTTP, (void \*) connfd);  pthread\_detach(newThread);  //deliverHTTP(connfd);  }  }  Explanation: 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 the need for another thread to join with the terminated thread. |

Question 7 (3 marks)

My code and explanation:

|  |
| --- |
| void \*logger(){  while(1){  if(logReady){  fwrite(logBuffer, strlen(logBuffer), 1, logfptr);  fflush(logfptr);  logReady = 0;  }  }  }  Explanation:  Since we have the luxury of having a shared memory region between threads, we can just rely on a shared buffer for the writeLog function to save the logs and the logger function to read from in order to write to a log file. The logReady variable was also shared between the writeLog and logger functions to coordinate their logging operations. |

TOTAL: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/20