**CS2106 Operating Systems**

**Lab 4**

**Introduction to Mutexes (ANSWER BOOK)**

|  |  |
| --- | --- |
| A0168721B | Ang Wei Ming |
| A0126400Y | Quek Yang Sheng |

**Question 1** (1 mark)

The value of glob at the end of main is: 20

**Question 2** (3 marks)

My changes are:

|  |
| --- |
| int main()  {  **int i;**  glob = 0;  **for(i=0; i<10; i++)**  **pthread\_create(&thread[i], NULL, child, (void \*) i);**  ..  ..  } |

**Question 3** (3 marks)

The value of glob is not correct. This is why: The value of glob is not printing correctly in order because multiple child threads are entering the critical section and incrementing the value at the same time. Therefore the value of glob when the thread enters and the value of glob when the thread exits can differ more than 2.

**Question 4** (3 marks)

The value of glob is still not correct. This is why: The mutex is not locked before it prints the value, but is locked only after the value is printed. This allows other child threads that has acquired the mutex to update the value. The mutex should also be unlocked after it has printed the value when exiting for the same reason that other child threads can update this value before it prints.

**Question 5** (4 marks)

Here are my modifications:

|  |
| --- |
| int main()  {  ..  . .  **for(i=0; i<10; i++)**  **pthread\_join(thread[i], NULL);**      ..  ..  return 0;  } |

**Question 6** (2 marks)

I see each sentences being printed on a new line and all lines are printed without any errors or corruptions.

**Question 7** (2 marks)

I see lines printed with errors and corruptions, sometimes multiple sentences on a single line.

**Question 8** (4 marks)

lab4p3.c prints incorrectly because multiple child threads are allowed to write to the buffer at the same time (i.e. no synchronization), therefore overwriting or combining with each other’s input.

The mutexes fix the problem because before each thread wants to write to the buffer, it has to lock the mutex first before it can write to it. This allows only a single thread to write to the buffer at any point in time. The mutex is only unlocked when the thread that is responsible for printing the buffer finishes consuming the buffer and printing it. At any point in time, **threadCode** threads are asleep and waiting to be woken up to enter the critical zone (i.e. accessing the buffer).   
  
**Question 9** (3 marks)

My modifications are:

|  |
| --- |
| void writeLog(const char \*format, ...)  {  char myBuffer[LOG\_BUFFER\_LEN];  va\_list args;    va\_start(args, format);  vsprintf(myBuffer, format, args);  va\_end(args);  **pthread\_mutex\_lock(&mutex);**  sprintf(logBuffer, "%s: %s\n", getCurrentTime(), myBuffer);  logReady=1;  }  void \*logger(){  while(1){  if(logReady){  fwrite(logBuffer, strlen(logBuffer), 1, logfptr);  fflush(logfptr);  logReady = 0;  **pthread\_mutex\_unlock(&mutex);**  }  }  } |

They work because:

because before each thread wants to write to the buffer, it has to lock the mutex first before it can write to it. This allows only a single thread to write to the buffer at any point in time. The mutex is only unlocked when the thread that is responsible for printing the buffer finishes consuming the buffer and printing it. At any point in time, **threadCode** threads are asleep and waiting to be woken up to enter the critical zone (i.e. accessing the buffer).

**TOTAL: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ / 25**