Operating Systems (Code: COMP2006)

CURTIN UNIVERSITY
School of Electrical Engineering, Computing and Mathematical Science
Discipline of Computing

Assignment
Simple Sorting Simulation (sss)

Due Date: **5PM (local time), Monday, 5 May 2025**

The objective of this programming assignment is to give you some experiences in using multiple threads and inter-thread communications. You will learn how to create threads and solve the critical section problems.

A. Description

1) Your assignment is to **simulate** a simple algorithm to sort *n* integers using two threads: T1 and T2. **Note that you don't need to know any other sorting algorithm to complete this assignment. More specifically, you just need to apply the steps described below.**

2) Let the integers be stored in an array *A* of size *n*. For example, consider an array *A* = (5, 3, 11, 2, 1, 4, 5, 1, 10, 11, 21, 17, 25, 16, 6), i.e., *n* = 15. Assume array *A* starts at location 0 and thus it ends at location *n* – 1, i.e., *A*[0] = 5, *A*[1] = 3, …, and *A*[14] = 6.

3) Thread T1 repeatedly checks each pair of integers at locations *A*[2*i*] and *A*[2*i*+1], for *i* = 0, 1, …, $\lfloor (n\text{-}1)/2 \rfloor$. For each pair of integers, T1 will swap them if they are not in order and updates the total number of swaps in a **shared** variable named **swap**.

   Note: $\lfloor x/y \rfloor$ is a floor function that returns the closest integer value lower than *x*/*y*. For example, $\lfloor 15/2 \rfloor = \lfloor 7.5 \rfloor = 7$, $\lfloor 16/2 \rfloor = \lfloor 8.0 \rfloor = 8$, and $\lfloor 17/3 \rfloor = \lfloor 5.667 \rfloor = 5$.

   For example, for *n* = 15, T1 will sort the integers at *A*[0] and *A*[1], *A*[2] and *A*[3], *A*[4] and *A*[5], *A*[6] and *A*[7], *A*[8] and *A*[9], *A*[10] and *A*[11], *A*[12] and *A*[13], *A*[14] and *A*[15]. Note that there is NO *A*[15] in the example. Thus, T1 will keep the content of *A*[14].

4) Thread T2 repeatedly checks each pair of integers at locations *A*[2*i* – 1] and *A*[2*i*], for *i* = 1, …, $\lfloor (n\text{-}1)/2 \rfloor$. For each pair of elements, T2 will swap them if they are not in order and updates the total number of swaps in a **shared** variable named **swap**.

   For example, for *n* = 15, T2 will sort the integers at *A*[1] and *A*[2], *A*[3] and *A*[4], *A*[5] and *A*[6], *A*[7] and *A*[8], *A*[9] and *A*[10], *A*[11] and *A*[12], *A*[13] and *A*[14].

5) Threads T1 and T2 will stop their repetition, i.e., terminate, if

   (i) There is no swap in T1 followed by there is no swap in T2, or

   (ii) There is no swap in T2 followed by there is no swap in T1.

   In other words, **the repetition stops when array *A* is already sorted**.

(iii) Once T1 and T2 terminate, the parent thread prints the contents of the **sorted array** *A*, and the total number of swaps, i.e., the value of the shared variable **swap** to the screen.

The following example shows how the simple sorting simulation works. Notice that in each iteration, T2 will start sorting only after T1 completes its iteration. Furthermore, T1 will start sorting only after T2 completes in the previous iteration.

Iteration 1

T1: For *A* = (5, 3, 11, 2, 1, 4, 5, 1, 10, 11, 21, 17, 25, 16, 6), P1 will sort the array into

A = (**3, 5**, **2, 11**, 1, 4, **1, 5**, 10, 11, **17, 21**, **16, 25,** 6). There are 5 swaps (see the pairs in **bold**). Thus, P1 will update variable **swap** = 0 + 5 = 5.

T2: For *A* = (3, 5, 2, 11, 1, 4, 1, 5, 10, 11, 17, 21, 16, 25, 6), P2 will sort the array into

A = (3, **2, 5**, **1, 11**, 1, 4, **5, 10**, 11, 17, **16, 21**, **6, 25**). There are 5 swaps. Thus, P2 will update **swap** = 5 + 5 = 10.

Iteration 2

T1: For *A* = (3, 2, 5, 1, 11, 1, 4, 5, 10, 11, 17, 16, 21, 6, 25), P1 will sort the array into

A = (**2, 3**, **1, 5**, **1, 11**, 4, 5, 10, 11, **16, 17**, **6, 21**, 25). There are 5 swaps. Thus, P1 will update **swap** = 10 + 5 = 15.

T2: For *A* = (2, 3, 1, 5, 1, 11, 4, 5, 10, 11, 16, 17, 6, 21, 25), P2 will sort the array into

A = (2, **1, 3**, **1, 5**, **4, 11**, 5, 10, 11, 16, **6, 17**, 21, 25). There are 4 swaps. Thus, P2 will update **swap** = 15 + 4 = 19.

Iteration 3

T1: For *A* = (2, 1, 3, 1, 5, 4, 11, 5, 10, 11, 16, 6, 17, 21, 25), P2 will sort the array into

A = (**1, 2**, **1, 3**, **4, 5**, **5, 11**, 10, 11, **6, 16**, 17, 21, 25). There are 5 swaps. Thus, P1 will update **swap** = 19 + 5 = 24.

T2: For *A* = (1, 2, 1, 3, 4, 5, 5, 11, 10, 11, 6, 16, 17, 21, 25), P2 will sort the array into

A = (1, **1, 2**, 3, 4, 5, 5, **10, 11**, **6, 11**, 16, 17, 21, 25). There are 3 swaps. Thus, P2 will update **swap** = 24 + 3 = 27.

Iteration 4

T1: For *A* = (1, 1, 2, 3, 4, 5, 5, 10, 11, 6, 11, 16, 17, 21, 25), P1 will sort the array into

A = (1, 1, 2, 3, 4, 5, 5, 10, **6, 11**, 11, 16, 17, 21, 25). There is 1 swap. Thus, P1 will update **swap** = 27 + 1 = 28.

T2: For *A* = (1, 1, 2, 3, 4, 5, 5, 10, 6, 11, 11, 16, 17, 21, 25), P2 will sort the array into

A = (1, 1, 2, 3, 4, 5, 5, **6, 10**, 11, 11, 16, 17, 21, 25). There is 1 swap. Thus, P2 will update **swap** = 28 + 1 = 29.

Iteration 5

T1: For *A* = (1, 1, 2, 3, 4, 5, 5, 6, 10, 11, 11, 16, 17, 21, 25), P1 will sort the array into

A = (1, 1, 2, 3, 4, 5, 5, 6, 10, 11, 11, 16, 17, 21, 25) → **No change**: 0 swap. Thus, P1 will update **swap** = 29 + 0 = 29.

T2: For *A* = (1, 1, 2, 3, 4, 5, 5, 6, 10, 11, 11, 16, 17, 21, 25), P2 will sort the array into

A = (1, 1, 2, 3, 4, 5, 5, 6, 10, 11, 11, 16, 17, 21, 25) → **No change**: 0 swap. Thus, P2 will update **swap** = 29 + 0 = 29.

There are two consecutive "No Change", meaning the array is already sorted. Thus, **both T1 and T2 terminate.**

Each terminating thread prints its thread ID and the total number of swaps it has completed. For the example, thread T1 will print the following on the screen, where ID1 is the thread ID for thread T1:

Thread ID1: total number of swaps = 16

Similarly, thread T2 will print the following:

Thread ID2: total number of swaps = 13

The main thread will then print the sorted array, and the total number of swaps, i.e.,

A = (1, 1, 2, 3, 4, 5, 5, 6, 10, 11, 11, 16, 17, 21, 25)

Total number of swaps to sort array *A* = 29.

Implementation

1) Write a program in C language that implements a simple sorting simulation using two child threads: T1 and T2. Both threads run the same function called sort(). Use pthread_create() to create a new thread so that the thread runs function sort(), and if the thread is T1 or T2. The integers to be sorted in stored in a file named ToSort. The file contains at maximum of $n = 200$ integers, where each pair of integers is separated by a space. Use the C version that is supported by the computers available in our computing laboratory.

2) Given a set of n integers in file ToSort, the main program / thread does the following:

   (i) Store the n integers in file ToSort into an array A and initialize a variable **swap** to zero.

   (ii) Creates two child threads: T1, and T2.

   (iii) Make T1 sort the integers at each pair $A[2i]$ and $A[2i + 1]$, for $i = 0, 1, \ldots, \lfloor (n\text{-}1)/2 \rfloor$, and T2 sort the integers at each pair $A[2i – 1]$ and $A[2i]$, for $i = 1, \ldots, \lfloor (n\text{-}1)/2 \rfloor$. Notice that T1 and T2 must run *concurrently*.

(iv) While waiting for T1 and T2 to finish their sorting steps, the main thread must block. It will be unblocked by one of the child threads. Use pthread_cond_wait() and pthread_cond_signal() for the block and unblock.

(v) Once T1 and T2 finish their sorting, i.e., they terminate, the main thread prints the sorted integers in array *A* and the total number of swaps in the shared variable **swap** to the screen.

3) For each iteration, T2 must block waiting for T1 to complete its sorting, which will unblock T2. Similarly, T1 in each iteration *k* must block waiting for T2 to complete its sorting in the earlier iteration *k*–1 that will unblock T1. Use pthread_cond_wait() and pthread_cond_signal() for the block and unblock.

4) You are allowed to use any additional variables. For each shared variable, you must address any possible synchronization issue.

5) Let us call the executable for the parent thread as **sss**. The program should be run as:

sss ToSort

where ToSort is the name of a file that contains *n* integers to be sorted.

Instruction for submission and assignment demonstration

1. Assignment submission is **compulsory**. Late submission is allowed. As stated in the unit outline, the penalty for late submission is as follows:

   - *For assessment items submitted within the first 24 hours after the due date/time, students will be penalised by a deduction of 5% of the total marks allocated for the assessment task;*
   - *For each additional 24-hour period commenced an additional penalty of 10% of the total marks allocated for the assessment item will be deducted; and*
   - *Assessment items submitted more than 168 hours late (7 calendar days) will receive a mark of zero.*

   Due dates and other arrangements may only be altered with the consent of most of the students enrolled in the unit and with the consent of the lecturer.

2. You must
   - submit your assignment (**in one zip file)** to the unit Blackboard, i.e., **FirstName_FamilyName_ID.zip**. Your submitted assignment includes: (i) **all source code** for the program with proper in-line and header documentation for each function. Use proper indentation so that your code can be easily read. Make sure that you use *meaningful* variable names and delete all *unnecessary* / commented code that you created while debugging your program; and (ii) **readme file** that, among others, explains how to compile your program and how to run the program, and (iii) a report (in pdf), described in the following instruction 3.
   - keep your assignment program / code in your Curtin account. During program demonstration, you need to run your program from the account.

   Make sure you keep a backup copy of your program in case you experience any system failure or lost your program.

3. Your report must include:

   - A signed cover page (i.e., declaration of originality). The declaration form is available from the unit Blackboard. By signing the form, among others, you agree on the following two statements:
     1. *The work I am submitting is entirely my own, except where clearly indicated otherwise and correctly referenced.*
     2. *Even with correct referencing, my submission will only be marked according to what I have done myself, specifically for this assessment. I cannot re-use the work of others, or my own previously submitted work, in order to fulfil the assessment requirements.*

   - Detailed discussion on all shared data structures used in your program, and how any mutual exclusion / thread synchronization is achieved on shared resources, e.g., what threads access the shared resources.

   - Description of any cases for which your program is not working correctly or how you

test your program that make you believe it works perfectly.

- Sample inputs and outputs from running your programs. For each sample output, you MUST explain if the output is correct or incorrect.

- List of your program / code. This code should be the same as the source code that you submitted to Blackboard per instruction 2(i). This copy is needed by your grader when they check your source code.

Your report will be assessed (worth 20% of the overall mark for the Assignment)

4. Assignment demonstration
   - You are required to demonstrate your program, subject to available computers and availability of graders. The time schedule for the demonstration will be announced later.
   - For the program demo, you MUST keep the source code of your programs in your Curtin account (per instruction 2), and the source code MUST be that you have submitted.
   - **The programs must run on any computer in Computing labs.**

Failure to meet these requirements may result in the assignment not being marked.