COMPSCI 3SH3 Winter, 2021
February 18, 2021

Lab3 Practice Time (Feb.22-26) - Threads and Concurrency

During Lab3 practice time you will learn how to write multithread program.

Practice Questions

1. Compile and run sorting program given in Listing 1.
2. Modify the code to sort the list
   `int list[SIZE] = {7,12,19,17,23,3,18,4,2,6,15,1,8};`

Listing 1: Sorting

```
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>

#define SIZE                    10
#define NUMBER_OF_THREADS        3

void *sorter(void *params);     /* thread that performs basic sorting algorithm
   */
void *merger(void *params);     /* thread that performs merging of results */

int list[SIZE] = {7,12,19,3,18,4,2,6,15,8};

int result[SIZE];

typedef struct
{
        int from_index;
        int to_index;
} parameters;

int main (int argc, const char * argv[])
{
        int i;

        pthread_t workers[NUMBER_OF_THREADS];

        /* establish the first sorting thread */
        parameters *data = (parameters *) malloc (sizeof(parameters));
        data->from_index = 0;
        data->to_index = (SIZE/2) - 1;
        pthread_create(&workers[0], 0, sorter, data);
        /* establish the second sorting thread */
        data = (parameters *) malloc (sizeof(parameters));
        data->from_index = (SIZE/2);
        data->to_index = SIZE - 1;
        pthread_create(&workers[1], 0, sorter, data);

        /* now wait for the 2 sorting threads to finish */
        for (i = 0; i < NUMBER_OF_THREADS - 1; i++)
                pthread_join(workers[i], NULL);

        /* establish the merge thread */
        data = (parameters *) malloc(sizeof(parameters));
        data->from_index = 0;
        data->to_index = (SIZE/2);
        pthread_create(&workers[2], 0, merger, data);

        /* wait for the merge thread to finish */
        pthread_join(workers[2], NULL);

        /* output the sorted array */
        for (i = 0; i < SIZE; i++)
                printf("%d  ",result[i]);
        printf("\n");

    return 0;
}

/**
 * Sorting thread.
 *
 * This thread can essentially use any algorithm for sorting
 */

void *sorter(void *params)
{
        int i;
        parameters* p = (parameters *)params;

        int begin = p->from_index;
        int end = p->to_index;

        int swapped = 1;
        int j = 0;
        int temp;

        while (swapped == 1) {
                swapped = 0;
                j++;

                for (i = begin; i <= end - j; i++) {
                        if (list[i] > list[i+1]) {
                                temp = list[i];
                                list[i] = list[i+1];
                                list[i+1] = temp;
                                swapped = 1;
                        }
                }
        }

        pthread_exit(0);
}
```

```
/**
 * Merge thread
 *
 * Uses simple merge sort for merging two sublists
 */

void *merger(void *params)
{
        parameters* p = (parameters *)params;

        int i,j;

        i = p->from_index;
        j = p->to_index;

        int position = 0;       /* position being inserted into result list */

        while (i < p->to_index && j < SIZE) {
                if (list[i] <= list[j]) {
                        result[position++] = list[i];
                        i++;
                }
                else {
                        result[position++] = list[j];
                        j++;
                }
        }

        /* copy the remainder */
        if (i < p->to_index) {
                while (i < p->to_index) {
                        result[position] = list[i];
                        position++;
                        i++;
                }
        }
        else {
                while (j < SIZE) {
                        result[position] = list[j];
                        position++;
                        j++;
                }
        }



        pthread_exit(0);
}
```