# Kennesaw State University

# Parallel and Distributed Computing

# Project - Pthread

Instructor: Kun Suo
Points Possible: 100
Difficulty: ★★★★☆

## Part 1: (50 points)

S1= 🍈🍉🦉🦉❤️🦉🥭🍒🥭🫐❤️🦉

S2= ❤️🦉

Emoji is popular in today's world. Given two emoji strings s1 and s2. Write a Pthread program to find out the number of subEmojiStrings, in string s1, that is exactly the same as s2.

For example, suppose number_subEmojiStrings(s1, s2) implements the function, then
number_subEmojiStrings("🥎🍈🍉🍊🎊🍈🍉🥭🍈", "🍈🍉") = 2,
number_subEmojiStrings("🥎🥎🥎", "🥎") = 3,
number_subEmojiStrings("💚❤️💙❤️", "💚💙") = 0.

The size of s1 and s2 (n1 and n2) as well as their data are input by user file. Assume that
n1 mod NUM_THREADS = 0
and
n2 < n1/NUM_THREADS.

The following is a sequential solution of the problem. read_f() reads the two emoji strings from a file named "emoji.txt and num_subEmojiString() calculates the number of substrings. Here s1 is a string with thousands of emojis and s2 = "❤️🦉".

https://github.com/kevinsuo/CS4504/blob/main/project-pthread.c

---

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

```c
#define MAX 102400

int total = 0;
int n1,n2;
char *s1,*s2;
FILE *fp;

int readf(FILE *fp)
{
	if((fp=fopen("emoji.txt", "r"))==NULL){
		printf("ERROR: can't open string.txt!\n");
		return 0;
	}
	s1=(char *)malloc(sizeof(char)*MAX);
	if(s1==NULL){
		printf("ERROR: Out of memory!\n");
		return -1;
	}
	s2=(char *)malloc(sizeof(char)*MAX);
	if(s1==NULL){
		printf("ERROR: Out of memory\n");
		return -1;
	}
	/*read s1 s2 from the file*/
	s1=fgets(s1, MAX, fp);
	s2=fgets(s2, MAX, fp);
	n1=strlen(s1);  /*length of s1*/
	n2=strlen(s2); /*length of s2*/
	if(s1==NULL || s2==NULL || n1<n2)  /*when error exit*/
		return -1;
}

int num_subEmojiString(void)
{
	int i,j,k;
	int count;

	for (i = 0; i <= (n1-n2); i++){
		count=0;
		for(j = i,k = 0; k < n2; j++,k++){  /*search for the next string of size of n2*/
			if (*(s1+j)!=*(s2+k)){
				break;
			}
			else
				count++;
			if(count==n2)
				total++;		/*find a substring in this step*/
		}
	}
	return total;
}

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

	readf(fp);
	count = num_subEmojiString();
	printf("The number of substrings is: %d\n", count);
	return 1;
}
```

---

You can find an example of the “emoji.txt” here:
https://raw.githubusercontent.com/kevinsuo/CS4504/main/emoji.txt

To compile the program with Pthread, use:
$ gcc project-pthread.c -o project-pthread.o -pthread

Current output:

```
administrator@CS3502-14 ~/p2> ./project-pthread.o
The number of substrings is: 55
```

(Different text files output is also different. For the emoji.txt, the output is 55)

Download the emoji.txt:
$ wget https://raw.githubusercontent.com/kevinsuo/CS4504/main/emoji.txt

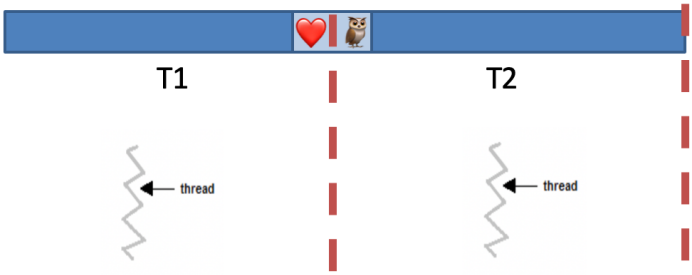
Write a parallel program using Pthread based on this sequential solution. Please set the thread number as **20** in your code. You can start with this template code:
https://github.com/kevinsuo/CS4504/blob/main/parallel-template.c

To compile the program with Pthread, use:
$ gcc file.c -o file.o -pthread
Here file refers to your source code name.

Expected output (the thread order can be random):
Suppose thread $i$ finds $n_i$ substrings, the total number of substrings should be equal to $\sum_{i=0}^{19} n_i$

HINT: Strings s1 and s2 are stored in a file named “emoji.txt”. String s1 is evenly partitioned for *NUM_THREADS* threads to concurrently search for matching with string s2. After a thread finishes its work and obtains the number of local matchings, this local number is added into a global variable showing the total number of matched substrings in string s1. Finally, this total number is printed out. Please make sure the number of substrings of parallel program is the same as the serial program.

HINT: A corner case. When search “❤️🦉” using multiple threads, please do not ignore corner case like below. The substring “❤️🦉” could be split into two parts when searched by two threads. Make sure your program can deal with it.




## Submission

Submit your assignment file through D2L using the appropriate link.
The submission must include the ***source code***, and ***a report describe your code logic***. ***Output screenshot of your code*** should be included in the report.

## Part 2 (50 pts)

Read the following program and modify the program to improve its performance.

https://github.com/kevinsuo/CS4504/blob/main/project-2-2.c

---

```
/*
  Each thread generates a data node, attaches it to a global list. This is reapeated for K times.
  There are num_threads threads. The value of "num_threads" is input by the student.
*/
#define _GNU_SOURCE
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
```

```c
#include <pthread.h>
#include <sys/time.h>
#include <sys/param.h>
#include <sched.h>

#define K 800 // genreate a data node for K times in each thread

struct Node
{
    int data;
    struct Node* next;
};

struct list
{
     struct Node * header;
     struct Node * tail;
};

pthread_mutex_t    mutex_lock;

struct list *List;

struct Node* generate_data_node()
{
    struct Node *ptr;
    ptr = (struct Node *)malloc(sizeof(struct Node));

    if( NULL != ptr ){
        ptr->next = NULL;
    }
    else {
        printf("Node allocation failed!\n");
    }
    return ptr;
}

void * producer_thread( void *arg)
{
    struct Node * ptr, tmp;
    int counter = 0;

    /* generate and attach K nodes to the global list */
    while( counter  < K )
    {
        ptr = generate_data_node();

        if( NULL != ptr )
        {
            while(1)
            {
                /* access the critical region and add a node to the global list */
                if( !pthread_mutex_trylock(&mutex_lock) )
                {
                    ptr->data  = 1;//generate data
                    /* attache the generated node to the global list */
                    if( List->header == NULL )
                    {
                        List->header = List->tail = ptr;
                    }
                    else
                    {
                        List->tail->next = ptr;
                        List->tail = ptr;
                    }
                    pthread_mutex_unlock(&mutex_lock);
                    break;
                }
            }
        }
        ++counter;
    }
}

int main(int argc, char* argv[])
{
```

```
    int i, num_threads;

    struct Node  *tmp,*next;
    struct timeval starttime, endtime;

    num_threads = atoi(argv[1]); //read num_threads from user
    pthread_t producer[num_threads];

    pthread_mutex_init(&mutex_lock, NULL);

    List = (struct list *)malloc(sizeof(struct list));
    if( NULL == List )
    {
       printf("End here\n");
       exit(0);
    }
    List->header = List->tail = NULL;

    gettimeofday(&starttime,NULL); //get program start time
    for( i = 0; i < num_threads; i++ )
    {
        pthread_create(&(producer[i]), NULL, (void *) producer_thread, NULL);
    }

    for( i = 0; i < num_threads; i++ )
    {
        if(producer[i] != 0)
        {
            pthread_join(producer[i],NULL);
        }
    }

    gettimeofday(&endtime,NULL); //get the finish time

    if( List->header != NULL )
    {
        next = tmp = List->header;
        while( tmp != NULL )
        {
           next = tmp->next;
           free(tmp);
           tmp = next;
        }
    }
    /* calculate program runtime */
    printf("Total run time is %ld microseconds.\n", (endtime.tv_sec-starttime.tv_sec) *
1000000+(endtime.tv_usec-starttime.tv_usec));
    return 0;
}
```

---

In this program there are num_threads threads. Each thread creates a data node and attaches it to a global list. This operation is repeated for K times by each thread. The performance of this program is measured by the program runtime (in microsecond). Apparently, the operation of attaching a node to the global list needs to be protected by a lock and the time to acquire the lock contributes to the total run time. Try to modify the program in order to reduce the program runtime.

To compile the program with Pthread, use:
$ gcc program-name.c -o program-name.o -pthread

To run the program, use
$ ./program-name.o NUM_THREADS

Here NUM_THREADS is the user input thread number

**Instructions to set multicore for local VMs**

(1) Create a Linux VM on your local laptop with 4 vCPUs;

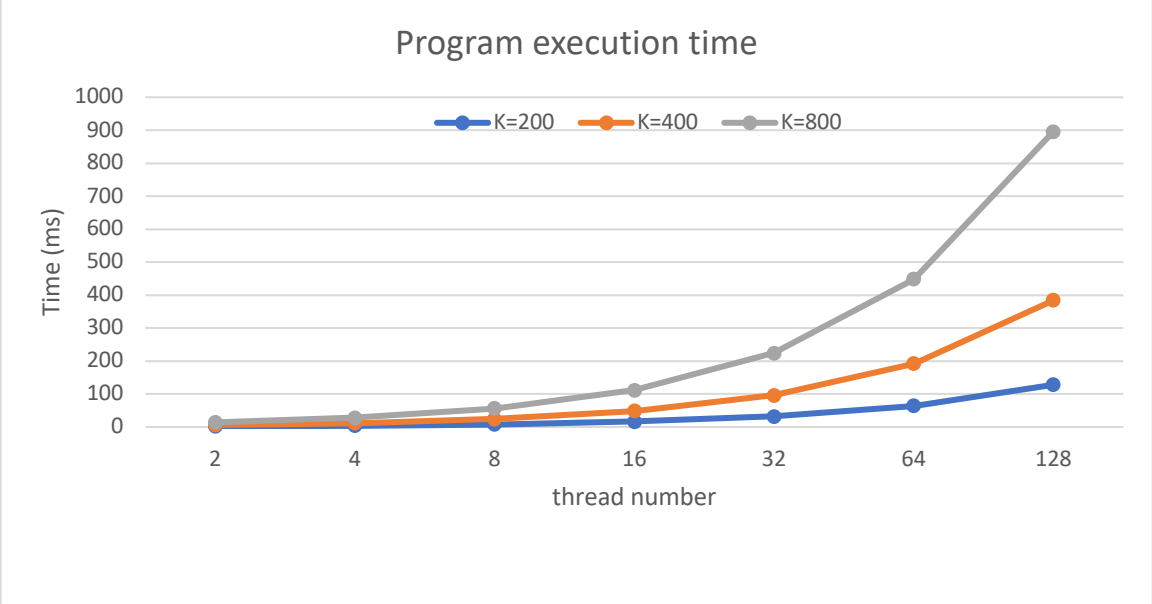
(2) Verify that you VM has 4 vCPUs:

```
$ cat /proc/cpuinfo
```

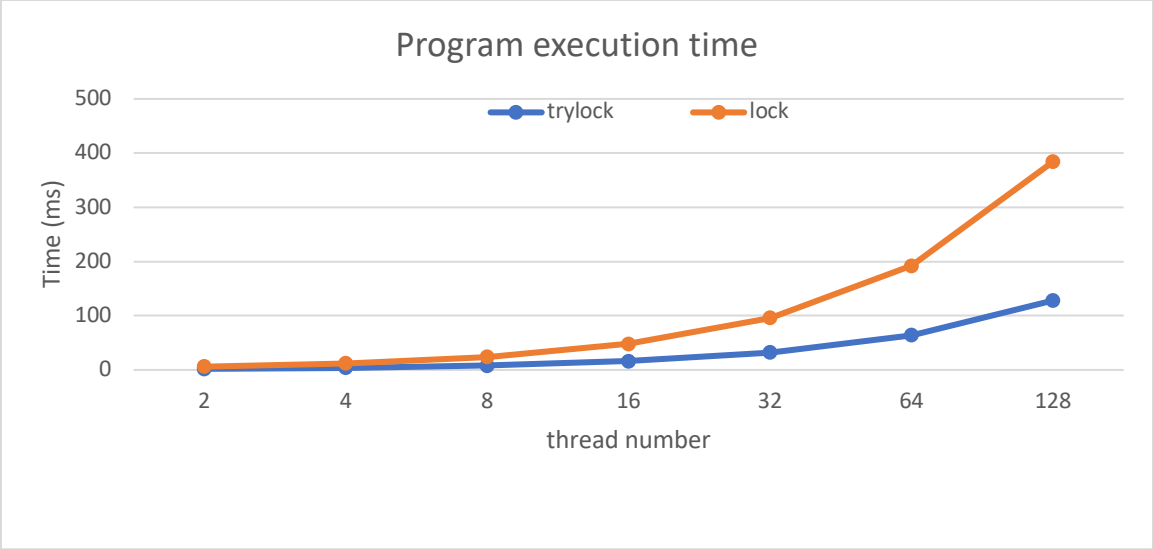
You should have 4 CPUs (processor: 0-3).

**Your tasks**

(1) Verify that your program achieves better performance than the original version by using different combinations of K and num_threads. Typical values of K could be 200, 400, 800, ... Typical values of num_threads could be 2, 4, 8, 16, 32, 64, 128, ... Draw figures to show the performance trend. To avoid the variation, please run 5 times and calculate the average time for each K and number of threads. (The figure below is just sketch, not the real data)




(2) The original program uses pthread_mutex_trylock. Will the use of pthread_mutex_lock make a difference? Why? Please try different number of threads. (The figure below is just sketch, not the real data)




(3) Since the problem does not require a specific order of the nodes in the global list, there are two ways to add nodes.

First, a node could be added to the global list immediately after it is created by a thread. https://github.com/kevinsuo/CS4504/blob/main/project-2-2.c

Alternatively, a thread could form a local list of K nodes and add the local list to the global list in one run. https://github.com/kevinsuo/CS4504/blob/main/project-2-2-new.c

Will the choice in how to add nodes make a difference? Why? To compare the difference between the two files, you can use diff command.

Draw figures to show the performance difference. Typical values of K could be 200, 400, 800, ... Typical values of num_threads could be 2, 4, 8, 16, 32, 64, 128, 256, 1024 ...

## Submitting Assignment

For Part 2, please submit a report with all the figures and analysis included.