CS 4504
Parallel and Distributed Computing

Project 2 Lab

Kun Suo

Computer Science, Kennesaw State University

https://kevinsuo.github.io/

Assignment (Part 1)

- Given two emoji strings s1 and s2. Write a Pthread program to find out the number of sub-emoji-strings, in string s1, that is exactly the same as s2.

S1= 🍈🍉🦉🦉❤️🦉🥭🍒🥭🫐❤️🦉

S2= ❤️🦉

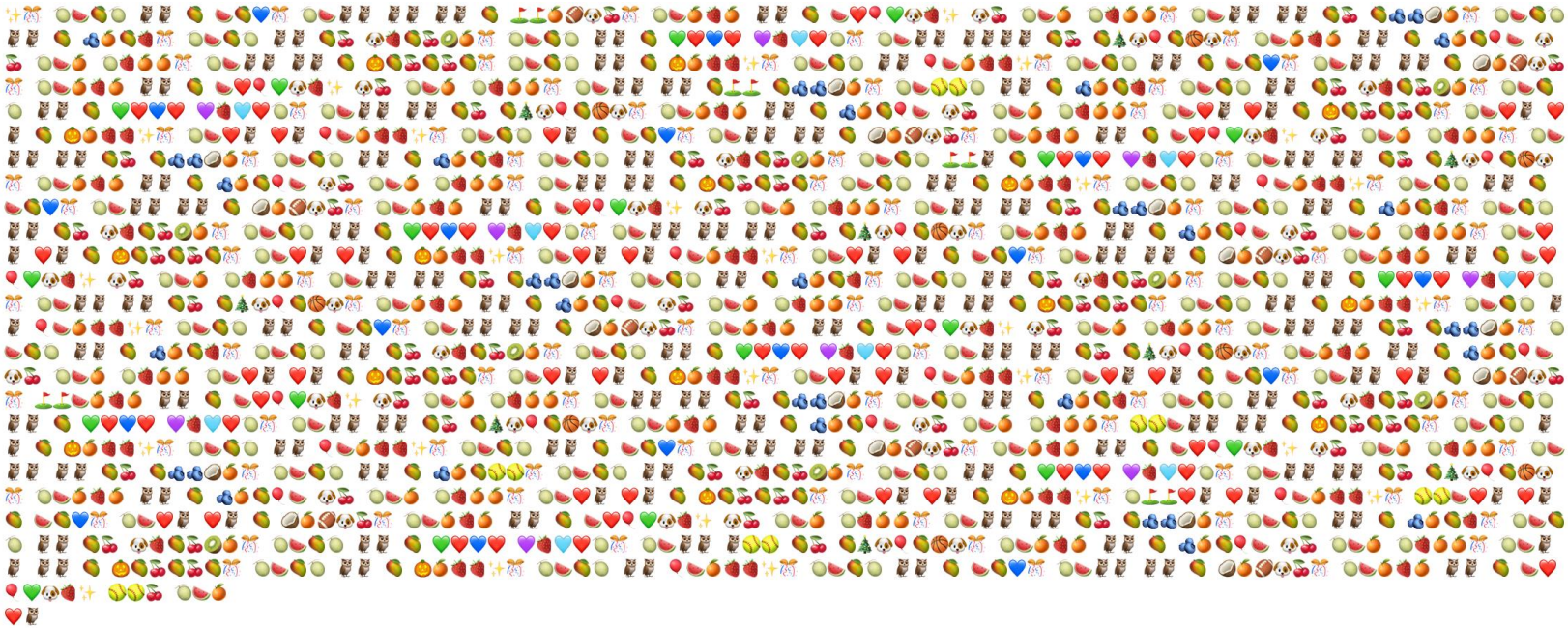
Assignment Examples

- number_subEmojiStrings("⚾🍈🍉🍊🎉🍈🍉🥭🍈", "🍈🍉") = 2
- number_subEmojiStrings("⚾⚾⚾", "⚾") = 3
- number_subEmojiStrings("💚❤️💙❤️", "💚💙") = 0

Subsequence not substring

Assignment

- Input file: https://raw.githubusercontent.com/kevinsuo/CS4504/main/emoji.txt



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

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

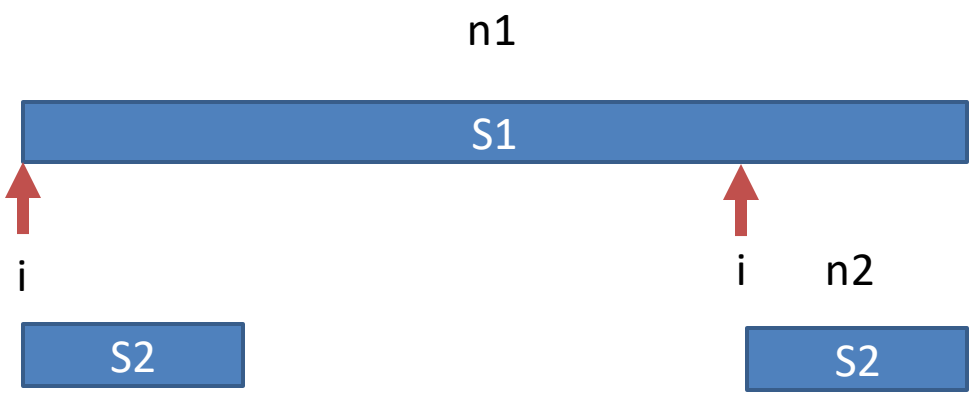
S1= 🍈🍉🦉🦉 … … … 🥭🫐❤️🦉

S2= ❤️🦉

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

int readf(FILE *fp)
{
    if((fp=fopen("strings.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(s2==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;
    return 0;
}
```

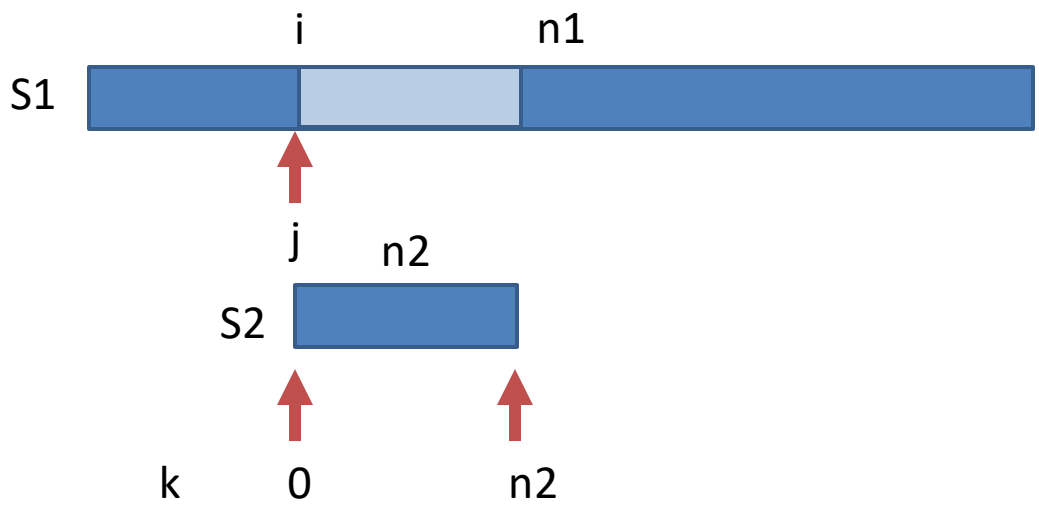
```c
int main(int argc, char *argv[])
{
        int count;

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

```c
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;
}
```



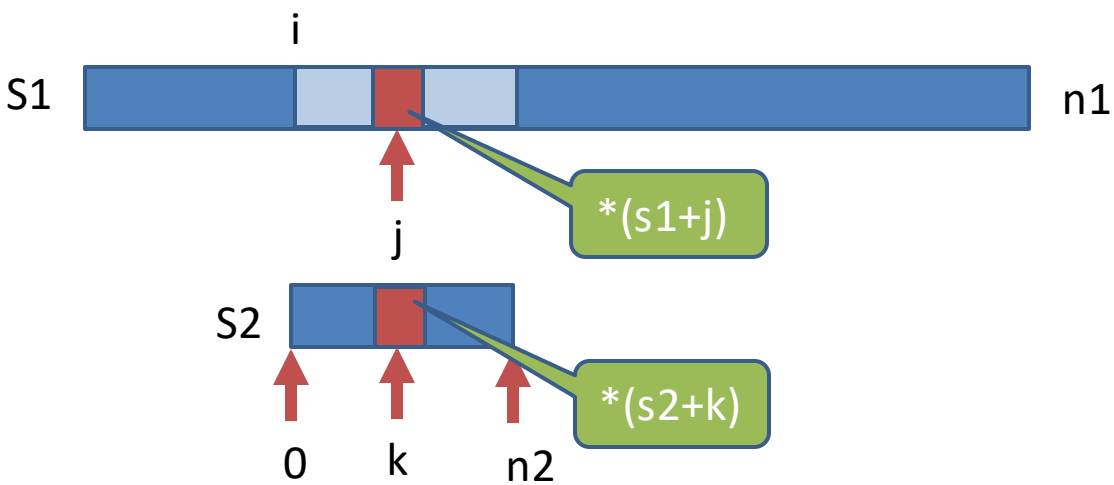
```c
int main(int argc, char *argv[])
{
        int count;

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

```c
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;
}
```



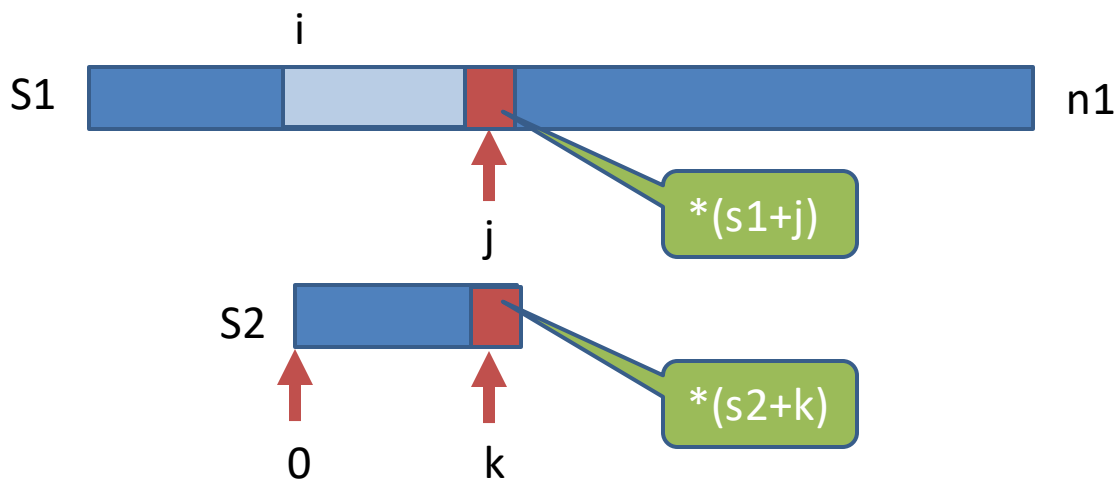
```c
int main(int argc, char *argv[])
{
        int count;

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

```c
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;
}
```



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

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

```c
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;
}
```



Assignment

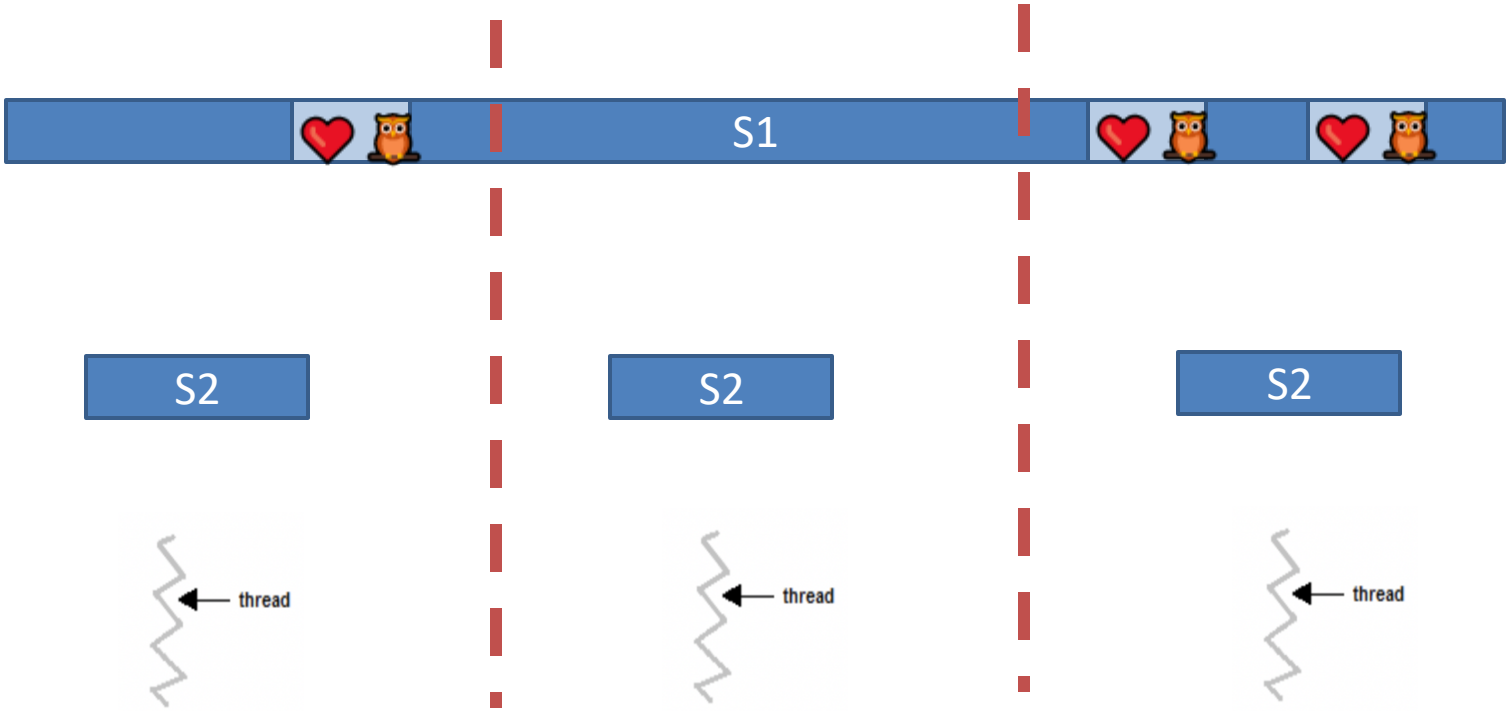
```c
int main(int argc, char *argv[])
{
        int count;

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

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

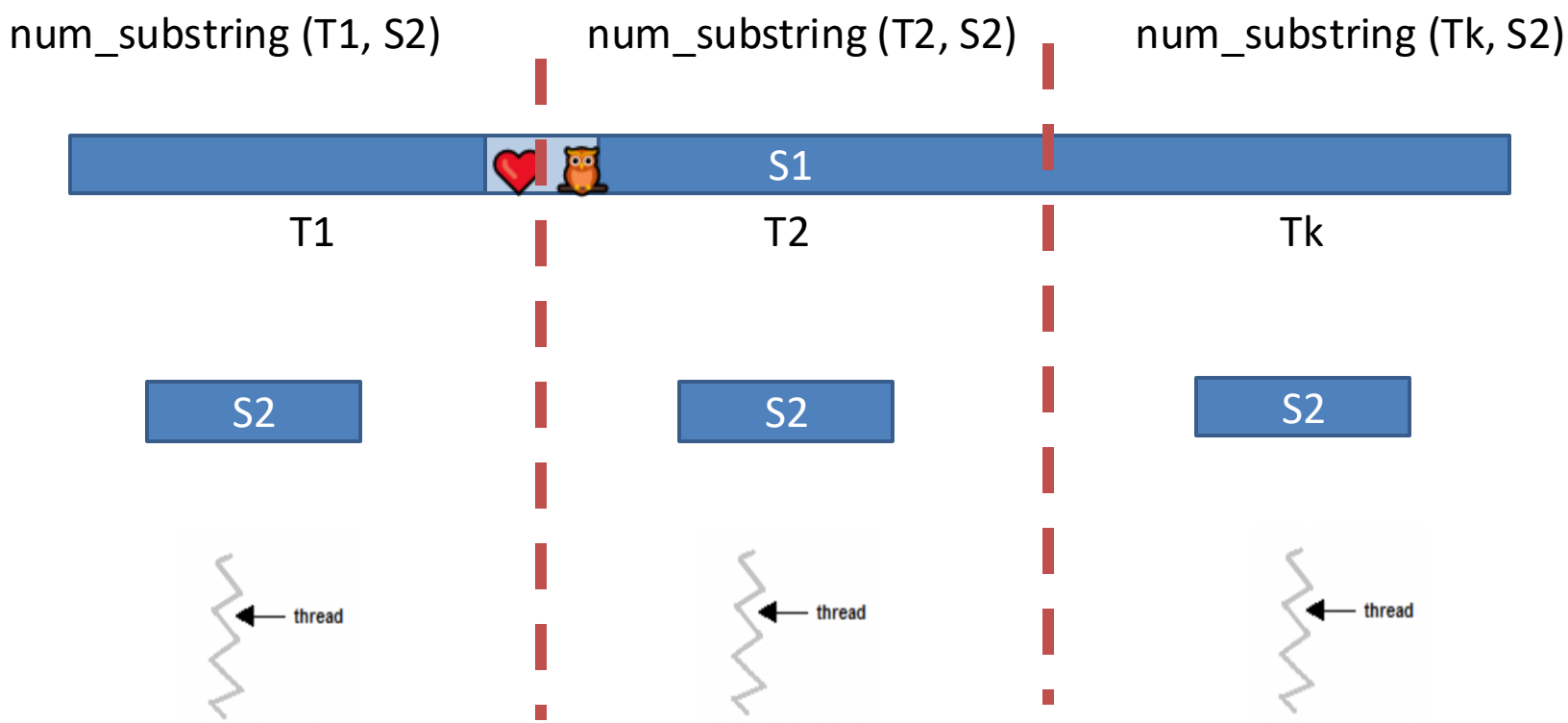
```
ksuo@LinuxKernel2 ~> ./project-pthread.o
The number of substrings is: 320
```

Idea

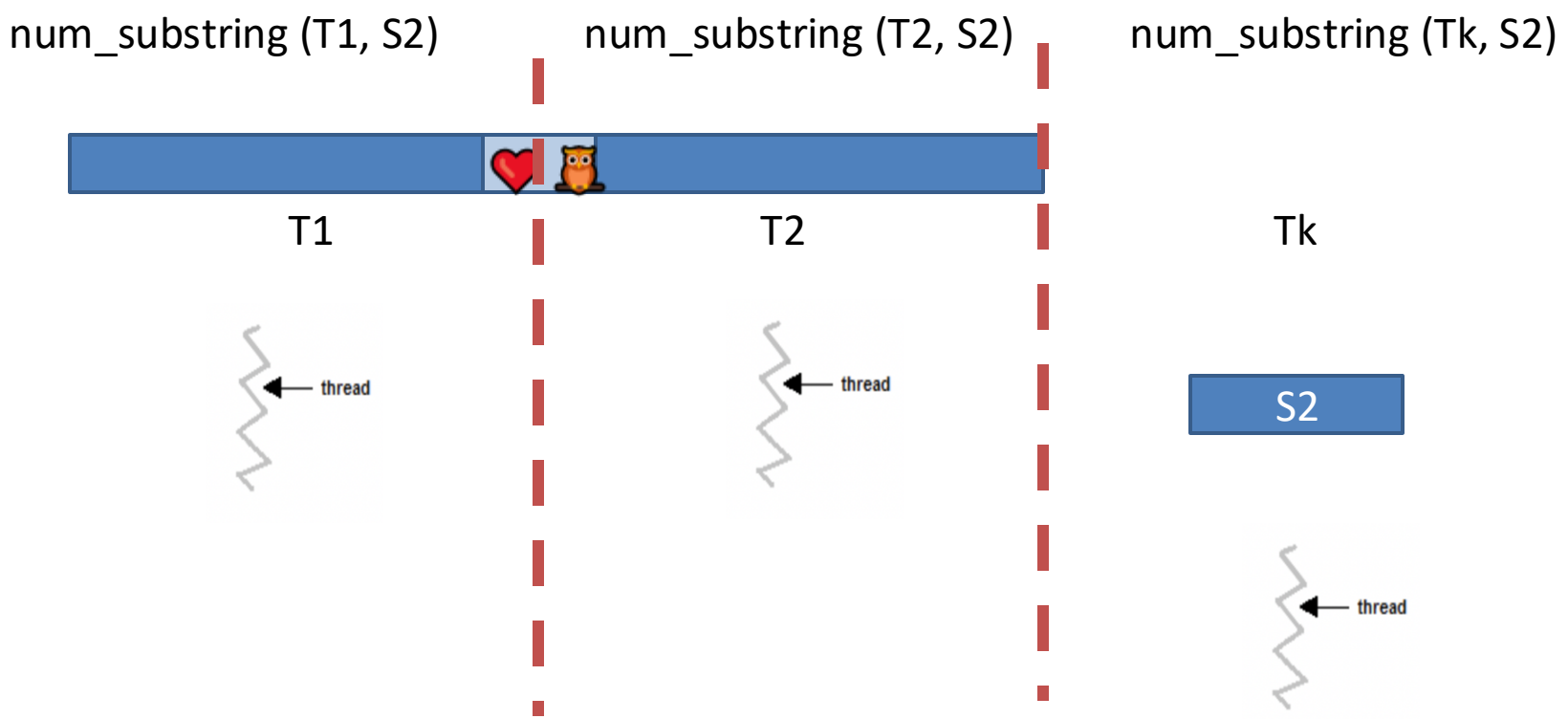
- Write a parallel program using Pthread based on this sequential solution.



Corner Case


num_substring (T1, S2)
num_substring (T2, S2)
num_substring (Tk, S2)
S1
T1
T2
Tk
S2
S2
S2
thread
thread
thread

Corner Case


num_substring (T1, S2)
num_substring (T2, S2)
num_substring (Tk, S2)
T1
T2
Tk
thread
thread
S2
thread

Verify whether your parallel thread is correct

- Modify the emoji.txt by yourself

- Compare the sequential and parallel program results that whether the total numbers are the same

```
ksuo@ltksup66583mac ~/Desktop> ./parallel.o
This is thread 0, num of substring 🦉 is
This is thread 1, num of substring 🦉 is
This is thread 2, num of substring 🦉 is
This is thread 3, num of substring 🦉 is
This is thread 4, num of substring 🦉 is
This is thread 5, num of substring 🦉 is
This is thread 6, num of substring 🦉 is
This is thread 7, num of substring 🦉 is
This is thread 8, num of substring 🦉 is
This is thread 9, num of substring 🦉 is
This is thread 10, num of substring 🦉 is
This is thread 11, num of substring 🦉 is
This is thread 12, num of substring 🦉 is
This is thread 13, num of substring 🦉 is
This is thread 14, num of substring 🦉 is
This is thread 15, num of substring 🦉 is
This is thread 16, num of substring 🦉 is
This is thread 17, num of substring 🦉 is
This is thread 18, num of substring 🦉 is
This is thread 19, num of substring 🦉 is
The number of substrings is:
```

Submission (Part 1)

1. source code
2. output screenshot of your parallel code
3. a report describe your code logic

Assignment (Part 2)

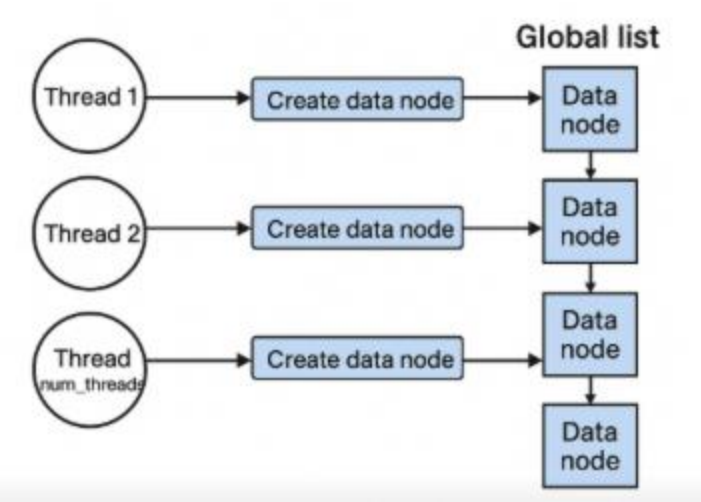
- Read the following program and modify the program to improve its performance.

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

  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.

Assignment

- 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.




```
Thread 1:  new → insert → new → insert → … → new → insert
                    ↘        ↘                  ↘
                      ↘      ↘                    ↘
Thread 2:        new → insert → new → insert → … → new → insert
                        ↘        ↘                  ↘
                          ↘      ↘                    ↘
  …
Thread N:        new → insert → new → insert → … → new → insert
                        ↘        ↘                  ↘
                          ↘      ↘                    ↘

                 Global List head → [node] → [node] → [node] → …
```

Assignment

Start time

Main thread
Create N child threads

Main thread will wait here until the child thread finishes

End time

```c
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) *
    return 0;
}
```

Assignment

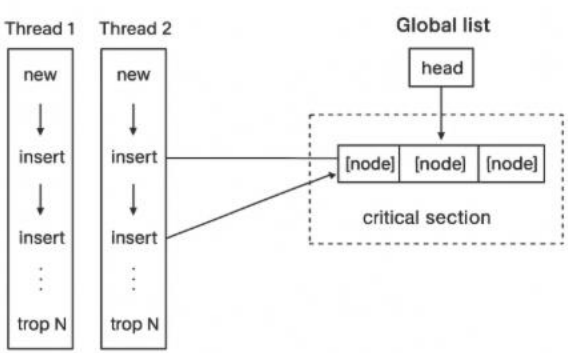
Generate a node

Enter the critical section

Put the node to the tail of global list

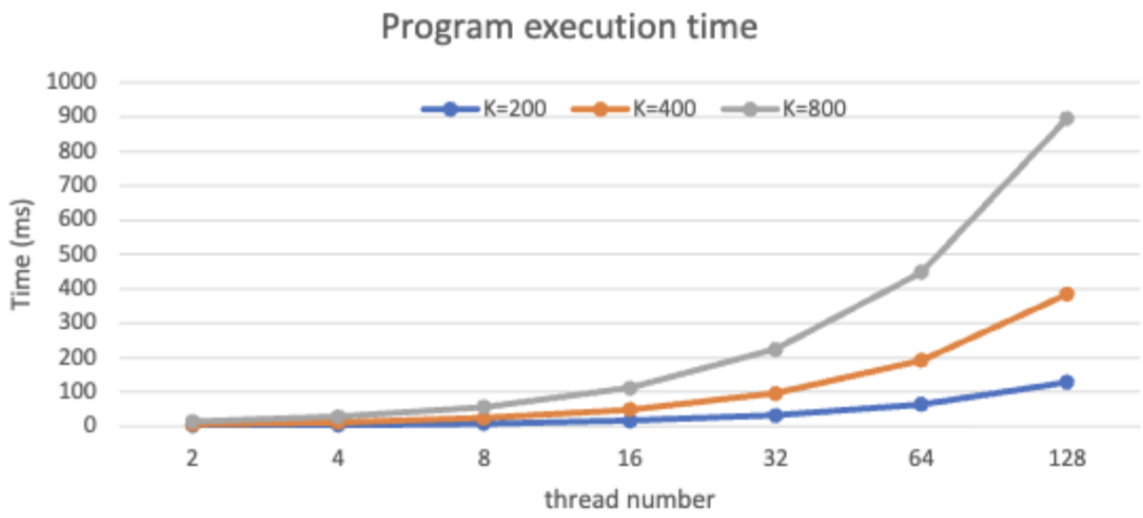
Leave the critical section

```
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;
    }
}
```



Assignment

- (1) Verify that your program achieves better performance than the original version by using different combinations of K and num_threads.



Assignment

pthread_mutex_trylock
v.s.
pthread_mutex_lock

pthread_mutex_lock (mutex) is a blocking call.

pthread_mutex_trylock (mutex) is a non-blocking call, useful in preventing the deadlock conditions (priority-inversion problem)

```
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;
    }
}
```

Assignment

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