Lab Assignment 6

Nikhil Gawande 111408013
Anupam Godse 111408016

---

1. Write a program to take input from user for number of files to be scanned and word to be searched. write a multi threaded program to search the files and return pattern if found

Program:

```c
#include <dirent.h>
#include <pthread.h>
#include <string.h>
#include <stdio.h>
#include <semaphore.h>

int file_index = 0;         // index for array[500];

struct params {
    char file_names [255];
    char word[255];
};

void *thread(void *wData_element){
    struct params *temp = wData_element;

    FILE *fp;
    char line[255]="";
    fp = fopen(temp->file_names, "r");

    if(fp == NULL){
        perror("Error: File open failure.");
    }
    else{
        fgets(line,255, fp);
        char *token = strtok(line, " \t");
      while (token != NULL){
            //printf("%s\n", token);
            if (strstr(token,temp->word) != NULL){
                  printf("%s", strstr(token,temp->word));
                  fclose(fp);
                  return NULL;
            }
            token = strtok(NULL, " \t");
      }
    }
    fclose(fp);
    return NULL;
```

```c
}

int main(int argc, char const* argv[]) {
      char dir_path[255];
      char word[255];
    DIR * dir_pointer;          // define a dir pointer;
    struct dirent * entry;      // entry under dir;
     // data directory contains all files in which we want to search.
    int count;
      printf("Enter the path of dir containing all files:");
      scanf("%s",dir_path);
      printf("Enter number of files to be scanned:");
      scanf("%d",&count);
      printf("Enter word to be searched:");
      scanf("%s",word);
    struct params wData[count];
    pthread_t tid_array[count];

    if ((dir_pointer = opendir(dir_path)) == NULL) {
      printf("can't open directory\n");
      return 0;
      }
    while( (entry = readdir(dir_pointer)) != NULL){

        if(entry->d_type == DT_REG){ // avoid the . and .. dir;
            char full_path[255];
            full_path[0] = '\0';    // initilize the string;

            strcat(full_path, dir_path);  // concatenate file directory;
            strcat(full_path, entry->d_name);    // concatenate filename;
            strcpy(wData[file_index].file_names, full_path); // store file name into file_names array;
            strcpy(wData[file_index].word,word);
            pthread_create(&tid_array[file_index], NULL, thread, &wData[file_index]);

            file_index++;   // increase the file index for next file.
        }
```

```c
    }

    for(int i=0; i < count; i++){
        pthread_join(tid_array[i], NULL);
    }
    return 0;
}
```

Output:

```
anupam@anupam-Inspiron-7548:~/LAB6$ cc q1.c -o a -lpthread
anupam@anupam-Inspiron-7548:~/LAB6$ ./a
Enter the path of dir containing all files:./directory/
Enter number of files to be scanned:5
Enter word to be searched:include
include<stdio.h>includeinclude<stdio.h>
include<stdio.h>include<stdio.h>include<stdio.h>anupam@anupam-Inspiron-7548:~/LAB6$
```

2. Write a program to find number of CPUs, create that many threads and attach those threads to
CPUs

Program:

```c
#define _GNU_SOURCE
#include<sched.h>
#include<pthread.h>
#include<stdio.h>
#include<stdlib.h>
#include<errno.h>
#include<unistd.h>
typedef struct params {                                 /*parameter structure*/
        pthread_mutex_t mutex;
        pthread_cond_t done;
        int cpu_id;
}params;
void* hello(void *arg){
      pthread_mutex_lock(&(*(params*)(arg)).mutex);   /*mutext lock on arguements*/
    int cpu = (*(params*)(arg)).cpu_id;
      pthread_t thread = pthread_self();                    /*get thread self_id*/
      cpu_set_t cpuset;
      CPU_ZERO(&cpuset);
      CPU_SET(cpu, &cpuset);
      //int k = 2000000000;
      int s = pthread_setaffinity_np(thread, sizeof(cpu_set_t), &cpuset);
      if (s != 0)
            printf("Error for CPU %d\n", cpu);
      if(CPU_ISSET(cpu,&cpuset) != 0){
            printf("CPU %d is assigned to Thread %lu\n",cpu,thread);
      }
      //while(k){                                                 // dummy loop to observe o/p
      //    k--;                                                // using system monitor
      //}
      pthread_mutex_unlock(&(*(params*)(arg)).mutex);
    pthread_cond_signal(&(*(params*)(arg)).done);
    pthread_exit(NULL);
}
int main(){
      params pars;
    pthread_mutex_init(&pars.mutex , NULL);
    pthread_cond_init(&pars.done, NULL);
    pthread_mutex_lock (&pars.mutex);
```

```c
    int j;
      long int n_cpu =  sysconf(_SC_NPROCESSORS_ONLN);
      printf("There are %ld CPUs.\n",n_cpu);
      pthread_t threads[n_cpu];
      for (j = 0; j < n_cpu; j++){
            pars.cpu_id = j;
            pthread_create(&threads[j], NULL, hello, &pars);
            pthread_cond_wait (&pars.done, &pars.mutex);
      }
      for(j = 0; j < n_cpu ; j++) {
            pthread_join(threads[j], NULL);
    }
    pthread_mutex_destroy (&pars.mutex);
    pthread_cond_destroy (&pars.done);
    return 0;
}
```

Output:

```
anupam@anupam-Inspiron-7548:~/LAB6$ cc q2.c -o a -lpthread
anupam@anupam-Inspiron-7548:~/LAB6$ ./a
There are 4 CPUs.
CPU 0 is assigned to Thread 139853224122112
CPU 1 is assigned to Thread 139853215729408
CPU 2 is assigned to Thread 139853205149440
CPU 3 is assigned to Thread 139853127743232
anupam@anupam-Inspiron-7548:~/LAB6$
```

3. Write a short program that creates 5 threads which print a tread "id" that is passed to thread
function by pointer.

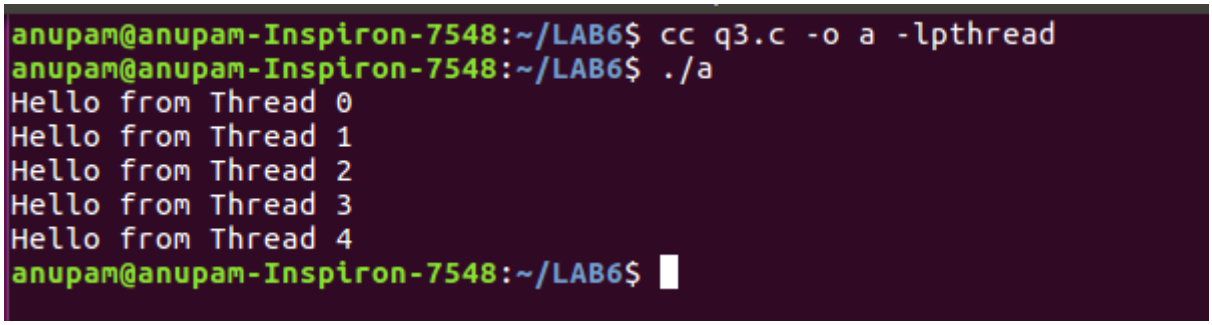
Program:

```c
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
typedef struct params {
        pthread_mutex_t mutex;                          // mutex lock for sync
        pthread_cond_t done;                                    // done conditional variable
        int id;                                                         //    id variable which we want to print
}params;
void* printId(void* arg){
    int id;
    pthread_mutex_lock(&(*(params*)(arg)).mutex);     /* Lock.  */
    id = (*(params*)(arg)).id;
    printf("Hello from Thread %d\n", id);
    pthread_mutex_unlock(&(*(params*)(arg)).mutex);
    pthread_cond_signal(&(*(params*)(arg)).done);

}
int main() {
    pthread_t threads[5];
    params pars;
    pthread_mutex_init (&pars.mutex , NULL);
    pthread_cond_init (&pars.done, NULL);

    pthread_mutex_lock (&pars.mutex);       /* Obtain a lock on the parameter.  */
    int i;
    for(i = 0; i < 5; i++) {
            pars.id = i;
            pthread_create(&threads[i], NULL, printId, &pars);
            pthread_cond_wait (&pars.done, &pars.mutex); /* Give up the lock, wait till thread is 'done',
                                                                                    then reacquire the lock.  */
    }
    for(i = 0; i < 5; i++) {
            pthread_join(threads[i], NULL);
    }
    /* Destroy all synchronization primitives.  */
    pthread_mutex_destroy (&pars.mutex);
    pthread_cond_destroy (&pars.done);
    return 0;
}
```

Output:


anupam@anupam-Inspiron-7548:~/LAB6$ cc q3.c -o a -lpthread
anupam@anupam-Inspiron-7548:~/LAB6$ ./a
Hello from Thread 0
Hello from Thread 1
Hello from Thread 2
Hello from Thread 3
Hello from Thread 4
anupam@anupam-Inspiron-7548:~/LAB6$