LAB 6:
111403003 Akash Sarda
111403019 Jassim Rahman
111403023 Aditya Malu

---

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

```
aditya:lab6 $ gcc 1.c -lpthread
aditya:lab6 $ ./a.out stdlib 1.c 2.c 3.c
file 2.c has it.
file 1.c has it.
file 3.c has it.
aditya:lab6 $
```

Usage: ./a.out <word> <files>

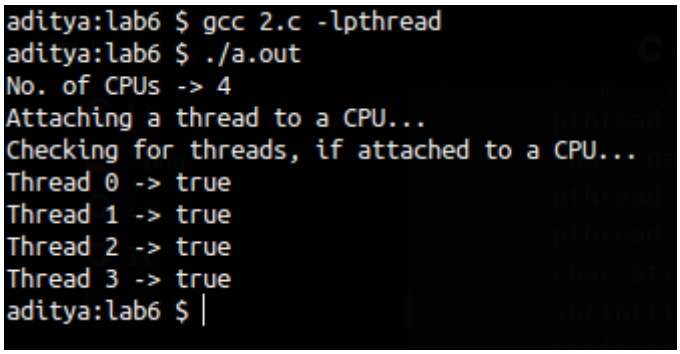
CODE:

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

typedef struct K{
     char *pattern;
     char *filename;
     pthread_mutex_t mutex;
     pthread_cond_t done;
}K;

void *search(void *k){
     K *p = (K *)k;
     pthread_mutex_lock(&p->mutex);
     char *name = p->filename, *pattern = p->pattern;
     pthread_mutex_unlock(&p->mutex);
     pthread_cond_signal(&p->done);
     char str[50];
     sprintf(str, "grep %s %s > /dev/null", pattern, name);
     if(!system(str)){
          printf("file %s has it. \n", name);
     }
     return NULL;
}

int main(int argc, char *argv[]){
     pthread_t th[3];
     int i;
     struct K k;
     pthread_mutex_init(&k.mutex, NULL);
     pthread_cond_init(&k.done, NULL);
```

```
    for(i = 0; i < argc - 2; i++){
        k.pattern = argv[1];
        k.filename = argv[i + 2];
        pthread_create(&th[i], NULL, search, &k);
        pthread_cond_wait(&k.done, &k.mutex);
    }

    for(i = 0; i < argc - 2; i++){
        pthread_join(th[i], NULL);
    }
}
```

---

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



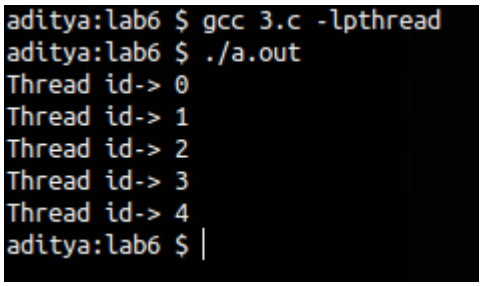
CODE:

```
#define _GNU_SOURCE             /* See feature_test_macros(7) */
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <unistd.h>

void* hello(void *cpu_set){
    pthread_t thread = pthread_self();
    cpu_set_t *cpuset = (cpu_set_t *)cpu_set;
    int s = pthread_setaffinity_np(thread, sizeof(cpu_set_t),
cpuset);
    if (s != 0)
        printf("Error for CPU");
}

int main(){
    long int n_cpu =  sysconf(_SC_NPROCESSORS_ONLN);
    printf("No. of CPUs -> %ld\n", n_cpu);
    int j, s;
    pthread_t threads[n_cpu];
```

```
    cpu_set_t cpuset;
    CPU_ZERO(&cpuset);
    for (j = 0; j < n_cpu; j++){
        CPU_SET(j, &cpuset);
    }
    printf("Attaching a thread to a CPU...\n");
    for (j = 0; j < n_cpu; j++){
        pthread_create(&threads[j], NULL, hello, &cpuset);
    }

    printf("Checking for threads, if attached to a CPU...\n");
    for (j = 0; j < n_cpu; j++){
        s = pthread_getaffinity_np(threads[j],
sizeof(cpu_set_t), &cpuset);
        printf("Thread %d -> %s\n", j, !s?"true":"false");
    }

    for(j = 0; j < n_cpu; j++) {
        pthread_join(threads[j], NULL);
    }
    exit(EXIT_SUCCESS);
}
```

---

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



CODE:

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#define N 5

struct params {
    pthread_mutex_t mutex;
    pthread_cond_t done;
    int id;
};
typedef struct params params_t;
```

```
void* hello(void* arg){
     int id;
     pthread_mutex_lock(&(*(params_t*)(arg)).mutex);
     id = (*(params_t*)(arg)).id;
     printf("Thread id-> %d\n", id);
     pthread_mutex_unlock(&(*(params_t*)(arg)).mutex);
     pthread_cond_signal (&(*(params_t*)(arg)).done);
}

int main() {
     pthread_t threads[10];
     params_t params;
     pthread_mutex_init (&params.mutex , NULL);
     pthread_cond_init (&params.done, NULL);

     int i;
     for(i = 0; i < N; i++) {
          params.id = i;
          pthread_create(&threads[i], NULL, hello, &params);
          pthread_cond_wait (&params.done, &params.mutex);
     }

     for(i = 0; i < N; i++) {
          pthread_join(threads[i], NULL);
     }

     pthread_mutex_destroy (&params.mutex);
     pthread_cond_destroy (&params.done);

     return 0;
}
```
