LAB 12
Breeha Qasim

2.1

Before changing INC_SIZE

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

#define INC_SIZE 10000000

static volatile int glob = 0;

void *access_global(void *args) {
    char *p = (char *)args;
    int loc = 0;

    printf("%s: thread started\n", p);
    for (int i = 0; i < INC_SIZE; i++) {
        loc = glob;
        loc++;
        glob = loc;
    }
    printf("%s: thread ending\n", p);
    return NULL;
}

int main(int argc, char *argv[]) {
    pthread_t t1, t2;

    printf("main: creating threads\n");
    pthread_create(&t1, NULL, access_global, "T0");
    pthread_create(&t2, NULL, access_global, "T1");
    printf("main: created threads\n");

    pthread_join(t1, NULL);
    pthread_join(t2, NULL);

    printf("main: joined threads\n");
    printf("main: glob = %d\n", glob);

    return 0;
}
```

After increasing INC_SIZE value

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

#define INC_SIZE 20000000

static volatile int glob = 0;

void *access_global(void *args) {
    char *p = (char *)args;
    int loc = 0;

    printf("%s: thread started\n", p);
    for (int i = 0; i < INC_SIZE; i++) {
        loc = glob;
        loc++;
        glob = loc;
    }
    printf("%s: thread ending\n", p);
    return NULL;
}

int main(int argc, char *argv[]) {
    pthread_t t1, t2;

    printf("main: creating threads\n");
    pthread_create(&t1, NULL, access_global, "T0");
    pthread_create(&t2, NULL, access_global, "T1");
    printf("main: created threads\n");

    pthread_join(t1, NULL);
    pthread_join(t2, NULL);

    printf("main: joined threads\n");
    printf("main: glob = %d\n", glob);

    return 0;
}
```

```
hp@hp-HP-Pavilion-x360-2-in-1-Laptop-14-ek0xxx:~/Documents/bqLab12$ nano list1.c
hp@hp-HP-Pavilion-x360-2-in-1-Laptop-14-ek0xxx:~/Documents/bqLab12$ gcc list1.c -o list1 -lpthread
hp@hp-HP-Pavilion-x360-2-in-1-Laptop-14-ek0xxx:~/Documents/bqLab12$ ./list1
main: creating threads
main: created threads
T1: thread started
T0: thread started
T0: thread ending
T1: thread ending
main: joined threads
main: glob = 11791154
hp@hp-HP-Pavilion-x360-2-in-1-Laptop-14-ek0xxx:~/Documents/bqLab12$ nano list1.c
hp@hp-HP-Pavilion-x360-2-in-1-Laptop-14-ek0xxx:~/Documents/bqLab12$ gcc list1.c -o list1 -lpthread
hp@hp-HP-Pavilion-x360-2-in-1-Laptop-14-ek0xxx:~/Documents/bqLab12$ ./list1
main: creating threads
main: created threads
T1: thread started
T0: thread started
T1: thread ending
T0: thread ending
main: joined threads
main: glob = 22061259
```

Explanation:
The global variable value changes if you increase the size of INC_SIZE variable

3

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

#define INC_SIZE 10000000

static volatile int glob = 0;
static pthread_mutex_t m;

void *access_global(void *args) {
    char *p = (char *)args;
    int loc = 0;

    printf("%s: thread started\n", p);

    for (int i = 0; i < INC_SIZE; i++) {
        // Try to acquire the lock without blocking
        if (pthread_mutex_trylock(&m) == 0) {
            loc = glob;
            loc++;
            glob = loc;
            pthread_mutex_unlock(&m);
        } else {
            // If the trylock fails, attempt a timed lock
            struct timespec ts;
            clock_gettime(CLOCK_REALTIME, &ts);
            ts.tv_nsec += 1000000; // wait 1 millisecond

            if (pthread_mutex_timedlock(&m, &ts) == 0) {
                loc = glob;
                loc++;
                glob = loc;
                pthread_mutex_unlock(&m);
            }
        }
    }

    printf("%s: thread ending\n", p);
    return NULL;
}
```

```c
int main(int argc, char *argv[]) {
    pthread_t t1, t2;

    // Initialize the mutex
    if (pthread_mutex_init(&m, NULL) != 0) {
        fprintf(stderr, "Failed to initialize mutex\n");
        return EXIT_FAILURE;
    }

    printf("main: creating threads\n");
    pthread_create(&t1, NULL, access_global, "T0");
    pthread_create(&t2, NULL, access_global, "T1");
    printf("main: created threads\n");

    pthread_join(t1, NULL);
    pthread_join(t2, NULL);

    printf("main: joined threads\n");
    printf("main: glob = %d\n", glob);

    // Destroy the mutex
    pthread_mutex_destroy(&m);

    return 0;
}
```

4

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

#define INC_SIZE 10000000

static volatile int glob1 = 0;
static volatile int glob2 = 0;

static pthread_mutex_t m;

// function without mutex
void *function_no_mutex(void *args) {
    for (int i = 0; i < INC_SIZE; i++) {
        int loc = glob1;
        loc++;
        glob1 = loc;
    }
    //printf("%s: thread1 ending \n", p);
    return NULL;
}


void *function_with_mutex(void *args) {
    for (int i = 0; i < INC_SIZE; i++) {
        pthread_mutex_lock(&m);
        int loc = glob2;
        loc++;
        glob2 = loc;
        pthread_mutex_unlock(&m);
    }
    //printf("%s: thread ending \n", p);
    return NULL;
}
```

```c
double measure_time(void *(*func)(void *)) {
    pthread_t t1, t2;
    struct timespec start, end;

    //staring time
    clock_gettime(CLOCK_MONOTONIC, &start);

    pthread_create(&t1, NULL, func, NULL);
    pthread_create(&t2, NULL, func, NULL);

    pthread_join(t1, NULL);
    pthread_join(t2, NULL);

    //ending time
    clock_gettime(CLOCK_MONOTONIC, &end);

    //calculating elapsed time in seconds
    double elapsed = (end.tv_sec - start.tv_sec) + (end.tv_nsec - start.tv_nsec) / 1e9;
    return elapsed;
}

int main() {

    pthread_mutex_init(&m, NULL);

    //printf("Time without mutex...\n");
    double time_no_mutex = measure_time(function_no_mutex);
    printf("Time without mutex: %f seconds\n", time_no_mutex);

    //printf("Time with mutex...\n");
    double time_with_mutex = measure_time(function_with_mutex);
    printf("Time with mutex: %f seconds\n", time_with_mutex);

    // Destroy the mutex
    pthread_mutex_destroy(&m);

    return 0;
}
```

```
hp@hp-HP-Pavilion-x360-2-in-1-Laptop-14-ek0xxx:~/Documents/bqLab12$ nano exec_time.c
hp@hp-HP-Pavilion-x360-2-in-1-Laptop-14-ek0xxx:~/Documents/bqLab12$ gcc exec_time.c -o exec_time -lpthread
hp@hp-HP-Pavilion-x360-2-in-1-Laptop-14-ek0xxx:~/Documents/bqLab12$ ./exec_time
Time without mutex: 0.054233 seconds
Time with mutex: 0.944963 seconds
```

Explanation:
The executed time for both function with mutex and without mutex is different. Because threads must wait to properly access shared resources, synchronization overhead causes programs with mutexes to execute more slowly. Threads operate more quickly without mutexes, however because of simultaneous access, there is a chance of data inconsistency.

5.1

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

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

Node *head = NULL;

void insert(int value) {
    Node *new_node = (Node *)malloc(sizeof(Node));
    new_node->data = value;
    new_node->next = head;
    head = new_node;
}

void delete(int value) {
    Node *temp = head, *prev = NULL;
    while (temp != NULL && temp->data != value) {
        prev = temp;
        temp = temp->next;
    }
    if (temp == NULL) return;
    if (prev == NULL) head = temp->next;
    else prev->next = temp->next;
    free(temp);
}

int count() {
    int count = 0;
    Node *temp = head;
    while (temp != NULL) {
        count++;
        temp = temp->next;
    }
    return count;
}

int main() {
    for (int i = 0; i < 100000; i++) insert(i);
    for (int i = 0; i < 1000; i++) delete(i);
    printf("The total elements in the list are %d\n", count());
    return 0;
}
```

Save modified buffer?

```
hp@hp-HP-Pavilion-x360-2-in-1-Laptop-14-ek0xxx:~/Documents/bqLab12$ gcc listing1.c -o list1 -lpthread
hp@hp-HP-Pavilion-x360-2-in-1-Laptop-14-ek0xxx:~/Documents/bqLab12$ ./list1
The total elements in the list are 99000
```

Explanation:
This code creates a basic linked list in which the number of members is tallied, values are added at the head, and values are removed by value. The main function prints the remaining count after inserting 100,000 data and deleting the first 1,000.

5.2

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

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

Node *head = NULL;
pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;

void insert(int value) {
    pthread_mutex_lock(&mutex);
    Node *new_node = (Node *)malloc(sizeof(Node));
    new_node->data = value;
    new_node->next = head;
    head = new_node;
    pthread_mutex_unlock(&mutex);
}

void delete(int value) {
    pthread_mutex_lock(&mutex);
    Node *temp = head, *prev = NULL;
    while (temp != NULL && temp->data != value) {
        prev = temp;
        temp = temp->next;
    }
    if (temp != NULL) {
        if (prev == NULL) head = temp->next;
        else prev->next = temp->next;
        free(temp);
    }
    pthread_mutex_unlock(&mutex);
}

int count() {
    pthread_mutex_lock(&mutex);
    int count = 0;
    Node *temp = head;
    while (temp != NULL) {
        count++;
        temp = temp->next;
    }
    pthread_mutex_unlock(&mutex);
    return count;
}
```

[Read 71 lines]

```c
int main() {
    int num_threads = 4;
    pthread_t threads[num_threads];

    for (int i = 0; i < num_threads; i++)
    {
        pthread_create(&threads[i], NULL, thread_insert, NULL);
    }
    for (int i = 0; i < num_threads; i++)
    {
        pthread_join(threads[i], NULL);
    }

    printf("The total elements in the list are %d\n", count());
    return 0;
}
```

```
hp@hp-HP-Pavilion-x360-2-in-1-Laptop-14-ek0xxx:~/Documents/bqLab12$ gcc listing2.c -o list2 -lpthread
hp@hp-HP-Pavilion-x360-2-in-1-Laptop-14-ek0xxx:~/Documents/bqLab12$ ./list2
The total elements in the list are 400000
```

Explanation:
Four threads each contribute 100,000 elements to a shared linked list in this code, which describes a multi-threaded program. Thread-safe access to the list is ensured by a mutex.