Mutex locking protects code in the ‘critical’ section when creating new threads. The variables are in safekeeping when locking the process that's being used in multiple threads. Underneath, line 16 - 30 are the critical section and sandwiched by the locking/unlocking mechanism. This is the perfect example of protecting the critical section with a mutex mechanism. Within the main function of the code, starting at 35, you can see the creation and joining of the threads that would then use the other function ‘incr’.

The second screenshot next to it is the initial array to show what the array looks like to reference after the incrementing functions. The first output is the array after adding 3 then the second is adding 2.

```c
// CS3600 Lab 4 : File: mutex2.c
//Objective to understad locks in threads

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

pthread_mutex_t mutex1 = PTHREAD_MUTEX_INITIALIZER;

int arr[5];


void *incr(int n){

  pthread_mutex_lock(&mutex1); // comment out

  int k = 0;

  for(k = 0; k < 5; k++){
        arr[k] += n;
        printf("%d ", arr[k]);
        sleep(2);

  }


    printf("\n");

  pthread_mutex_unlock(&mutex1); // comment out
}



int main() {

        int i;
        for(i = 0; i < 5; i++){
        arr[i] = 1;
        }

        printf("Initial Array: ");
                for(int i = 0; i < 5; i++){
                        printf("%d ", arr[i]);
                }
        printf("\n");

  pthread_t t1, t2; //Initlizing the threa

 pthread_create(&t1, NULL, incr, 2);
 pthread_create(&t2, NULL, incr, 3);

pthread_join(t1,NULL);
pthread_join(t2,NULL);
  exit(0);
}
```

```
daniel@daniel-VirtualBox:~$ ./mutex1
Initial Array: 1 1 1 1 1
4 4 4 4 4
6 6 6 6 6
```

This second pair of screenshots what happens when commenting out the locks. The outputs come out skewed and mixed compared to earlier.

```c
// CS3600 Lab 4 : File: mutex2.c
//Objective to understad locks in threads

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

pthread_mutex_t mutex1 = PTHREAD_MUTEX_INITIALIZER;

int arr[5];


void *incr(int n){

  //pthread_mutex_lock(&mutex1); // comment out

  int k = 0;

  for(k = 0; k < 5; k++){
        arr[k] += n;
        printf("%d ", arr[k]);
        sleep(2);

  }


    printf("\n");

  //pthread_mutex_unlock(&mutex1); // comment out
}



int main() {

        int i;
        for(i = 0; i < 5; i++){
        arr[i] = 1;
        }

        printf("Initial Array: ");
                for(int i = 0; i < 5; i++){
                        printf("%d ", arr[i]);
                }
        printf("\n");

  pthread_t t1, t2; //Initlizing the threa

 pthread_create(&t1, NULL, incr, 2);
 pthread_create(&t2, NULL, incr, 3);

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

```
daniel@daniel-VirtualBox:~$ ./mutex1
Initial Array: 1 1 1 1 1
4 6 4 6 4 6 4 6 4 6
```

RAW CODE

```c
// CS3600 Lab 4 : File: mutex2.c
//Objective to understad locks in threads

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

pthread_mutex_t mutex1 = PTHREAD_MUTEX_INITIALIZER;

int arr[5];


void *incr(int n){

  pthread_mutex_lock(&mutex1); // comment out

  int k = 0;

  for(k = 0; k < 5; k++){
    arr[k] += n;
    printf("%d ", arr[k]);
    sleep(2);

  }


    printf("\n");

  pthread_mutex_unlock(&mutex1); // comment out
}



int main() {

    int i;
    for(i = 0; i < 5; i++){
    arr[i] = 1;
```

```
    }

    printf("Initial Array: ");
        for(int i = 0; i < 5; i++){
            printf("%d ", arr[i]);
        }
    printf("\n");

  pthread_t t1, t2; //Initlizing the threa

 pthread_create(&t1, NULL, incr, 2);
 pthread_create(&t2, NULL, incr, 3);

pthread_join(t1,NULL);
pthread_join(t2,NULL);
  exit(0);
}
```