פרויקט שעבדתי עליו:
הפרויקט היה במערכות הפעלה,
פיתחו web server כאשר נתנו לנו איזשהו web server בסיסי ואנחנו התבקשנו לבנות עליו עוד דברים.

הweb server שקיבלנו עבד רק עם חוט יחיד, ואנחנו התבקשנו להפוך את המערכת לmulti thread.
אז כמובן החיסרון הגדול ביותר של web server שעובד עם חוט יחיד זה כאשר יש יותר מבקשה אחת, אז כל שאר הבקשות נדחות.

אז אכן התבקשנו לעשות את המערכת לmulti thread.
אז האופן הטריוויאלי של להפוך את המערכת לmulti thread היא שהמערכת הפעלה תייצר חוט חדש עבור כל בקשה שמגיע לשרת. מערכת ההפעלה תתזמן כל אחד מהבקשות, וככה בעצם נפתור את הבעיה שהייתה עם החוט היחיד – אף בקשה לא צריכה לחכות לאף בקשה אחרת שתסתיים – כל בקשה תקבל חוט ישר.
החיסרון בגישה זו היא שהשרת ישלם את הoverload של יצירת חוט בכל פעם שבקשה חדשה מגיעה, וזה בעצם היה עיקר התרגיל – לחסוך את הoverload הזה, ואת זה פתרנו על ידי כך שיצרו **pool thread**.

הpool thread שיצרנו היה בגודל קבוע וידוע מראש, ובעצם כבר בבניית המערכת יצרנו מספר קבוע של חוטים, שהם בעצם היוו את הthread pool.
בגישה זו, בעצם כאשר מגיעה בקשה חדשה לשרת, אז הבקשה הזאת נדחת עד אשר יהיה חוט פנוי מבריכת החוטים לטפל בבקשה הזאת.
בעצם אפשר להסתכל על הבעיה הזאת עם שני שחקנים:
- יש את בריכת החוטים, שמכילה בתוכה את הworker thread והם אלו שיבצעו את הבקשות.
- יש את הrequest עצמם שמגיעים לשרת, שהם מחכים לחוטים שיבצעו אותם.

אז היה לנו **master thread** שבעצם יצר את בריכת החוטים, והוא זה שבעצם אחראי לקבל את הבקשות, ולשים את הבקשות האלה בתור כאשר גודל התור גם היה ידוע לנו מראש.
בעצם חוט המסטר מקבל בקשות http ושם אותם בתור כלשהו, ולאחר מכן הוא משייך כל בקשה בתורה לחוט עבד, אשר חוט העבד מבצע את הבקשה עצמה.
עבדנו בצורה של FIFO מבחינת תור הבקשות.

כעת, לגבי **המידע המשותף** בין חוט המסטר לחוטי העבד:
הם היו ביחסי producer-consumer וזה דרש מאיתנו שיהיה סנכרון בין הגישות לאמצעים המשותפים שלהם.
אמצעים משותפים שהיה להם:
1. תור הבקשות: חוט המסטר הוא זה שהכניס לתור הזה, וחוטי העבד הם אלו שקראו מהתור הזה.
אם למשל תור הבקשות הינו מלא אז חוט המסטר חייב להיות חסום,
ואם למשל התור ריק, אז חוטי העבד חייבים להמתין.

בנוסף, השתמשנו גם בconditional variables – כלומר לא השתמשנו בbusy wait.

```
pthread_mutex_t Lock;
pthread_cond_t WaitingQueueEmpty;
pthread_cond_t QueuesFull;
RequestManager requestsManager;
//
```

ניתן לראות בפרויקט זה היה לנו 2 משתני תנאי:
1. WaitingQueueEmpty – שזה בעצם עבור מצב של תור בקשות ריק – במצב זה חוטי העבדים אמורים להמתין.
במצב שהמשתנה הזה הוא אמת – זה אומר שהחוטים אמורים להמתין, והם מחכים לסיגנל מחוט המסטר.

```
void* thread_function(void* thread)
{
    WorkerThread* this_thread = (WorkerThread*)thread;
    while (1)
    {
        pthread_mutex_lock(&Lock);
        while(!requestManagerHasWaitingRequests(requestsManager))
        {
            pthread_cond_wait(&WaitingQueueEmpty, &Lock);
        }
```

כאן ניתן לראות את חוטי העבד אשר ממתינים על משתנה התנאי הזה – הם יישלחו להמתנה כאשר לא יהיו בקשות עבורם לבצע.

```
void addSignalAndUnlock(RequestObject requestObject){
    requestManagerAddPendingRequest(requestsManager, requestObject);
    pthread_cond_signal(&WaitingQueueEmpty);
    pthread_mutex_unlock(&Lock);

//    printf("\n\n*******request (%d) added to w.q*******\n\n", requestObject->val);
//    requestManagerPrint(requestsManager);
}
```

בפונקציה הזאת ניתן לראות את חוט המסטר שולח סיגנל למשתנה תנאי זה. המסטר קורא לפונקציה כאשר הוא מוסיף בקשה חדשה – ואז אם יש חוטים שהם ישנים, הם יתעוררו כי המסטר שולח להם סיגנל על משתנה תנאי הזה.

2. QueuesFull – שזה בעצם עבור מצב של אין יותר מקום לבקשות חדשות.

```
        requestHandle(fd, this_thread, arrival_time, dispatch_interval);
        Close(fd);

        pthread_mutex_lock(&Lock);
        requestManagerRemoveFinishedRequest(requestsManager, requestObject);

//        printf("\n\n*******request(%d) finished*******\n\n", requestObject->val);
//        requestManagerPrint(requestsManager);
        pthread_cond_signal(&QueuesFull);
        pthread_mutex_unlock(&Lock);
    }
```

פה מודגש בצבע ירוק קטע קוד של חוט העבד, שאחרי שהוא מסיים את ביצוע המשימה שלו, ומשחרר את המנעול, הוא שולח סיגנל על משתנה תנאי QueueFull שבעצם מה שזה עושה זה להעיר את חוט המסטר ומאפשר לו להכניס בקשות(במידה וקיימות בקשתות ממתינות לחוטי עבד) אל תור הבקשות.

```
        while (!requestManagerCanAcceptRequests(requestsManager))
        {
            pthread_cond_wait(&QueuesFull, &Lock);
        }
        RequestObject requestObject = createRequestObject(connfd);
        addSignalAndUnlock(requestObject);
    }
```

מצד שני, זה קטע קוד שנמצע בmain כלומר זה קטע קוד של חוט המסטר, ובו ניתן לראות את משתנה המצב requestMangerCanAcceptRequest אשר מה שהוא מחזיר זה בעצם אם ניתן להכניס בקשות או לא.
במידה ולא ניתן להכניס בקשות, אז חוט המסטר בעצם נשלח להמתנה על משתנה התנאי QueusFull והוא מחכה שאיזשהו חוט עבד יעיר אותו ויאפשר לו להכניס בקשה חדשה לתוך תור ההמתנה של הבקשות.

```
void* thread_function(void* thread)
{
    WorkerThread* this_thread = (WorkerThread*)thread;
    while (1)
    {
        pthread_mutex_lock(&Lock);
        while(!requestManagerHasWaitingRequests(requestsManager))
        {
            pthread_cond_wait(&WaitingQueueEmpty, &Lock);
        }
        RequestObject requestObject = requestManagerGetReadyRequest(requestsManager);
        requestManagerAddReadyRequest(requestsManager, requestObject);

//        printf("\n\n*******request(%d) is now running*******\n\n", requestObject->val);
//        requestManagerPrint(requestsManager);

        int fd = requestObject->val;
        struct timeval arrival_time = requestObject->time_arrive;
        struct timeval dispatch_interval = requestObject->disp;

        pthread_mutex_unlock(&Lock);

//        sleep(25);

        requestHandle(fd, this_thread, arrival_time, dispatch_interval);
        Close(fd);

        pthread_mutex_lock(&Lock);
        requestManagerRemoveFinishedRequest(requestsManager, requestObject);

//        printf("\n\n*******request(%d) finished*******\n\n", requestObject->val);
//        requestManagerPrint(requestsManager);
        pthread_cond_signal(&QueuesFull);
        pthread_mutex_unlock(&Lock);
    }
```

למשל פה נוכל לראות את הפונקציה של כל חוט עבד – אז פה אכן ניתן לראות את משתנה המצב שמסומן בצהוב שזה הוא המשתנה שיש לנו בתוך הwhile , ומה שהוא עושה זה בעצם – כל עוד אין בקשה לבצע – הוא משחרר את החוט מהמנעול שהוא תפס, ושם אותו בתור המתנה. ברגע שתגיע בקשה – חוט המסטר יעיר את אחד החוטים, אשר יבצעו את המשימה.
מה שניתן לראות פה בנוסף, זה בעצם אחרי שהחוט העבד מתעורר, הוא יוצר את הבקשה, ומוסיף את הקשה לתור אשר משותף לחוטים. על מנת לשמור על סנכרון, הוא עושה את כל זה תוך כדי שהמנעול נמצא בידיים שלו.
לאחר שהוא מסיים את הגישה לנתונים המשותפים, ניתן לראות שהוא משחרר את המנעול, מבצע את הבקשה, ולאחר מכאן שוב מבקש את המנעול על מנת להוציא את הבקשה מתוך מבנה הנתונים שמכיל את הבקשות.