1.Да се напише програма во C која работи со процеси. Програмата (главниот процес) треба да дозволи да се внесе целобројна вредност од тастатура K (најмногу 100). Потоа, главниот процес треба да дозволи да се внесат K наредби од тастатура, притоа секоја наредба што се внесува има име на наредбата и три аргументи. Главниот процес, за секоја наредба внесена од тастатура треба да креира дете процес, на кое ја испраќа наредбата (заедно со трите аргументи), притоа, дете процесот треба да ја изврши таа наредба.Секое од деца процесите, пред да ја изврши наредбата, одбира случаен број од 1 до 20 (со наредбата rand()%20+1) и чека толку секунди пред да почне да ја извршува наредбата. Дете процесот печати колку секунди ќе чека пред да почне со чекањеГлавниот процес, ги чека деца процесите да завршат со извршување, потоа печати на екран дека и тој завршува.

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>
#include <time.h>

int main() {
    int k;  // Број на наредби
    char command[100];  // Име на наредбата
    char arg1[50], arg2[50], arg3[50];  // Аргументи на наредбата
    pid_t dete;  // Идентификатор на процесот

    // Иницијализација на генератор за случајни броеви
    srand(time(NULL));

    // Внесување на бројот на наредби
    printf("Внеси број на наредби (K): ");
    scanf("%d", &k);

    // За секоја наредба
    for (int i = 0; i < k; i++) {
        printf("Внеси наредба %d (име на наредба и три аргументи): ", i + 1);
        scanf("%s %s %s %s", command, arg1, arg2, arg3);  // Внесување на
наредбата и аргументите

        // Креирање на дете процес
        dete = fork();

        if (pid == 0) {
```

```
            // Ова е кодот што го извршува дете процесот

            // Генерирање на случаен број за време на чекање (помеѓу 1 и 20
секунди)
            int wait_time = rand() % 20 + 1;  // Генерира случаен број помеѓу 1 и
20
            printf("Дете процесот ќе чека %d секунди пред да ја изврши наредбата:
%s %s %s %s\n", wait_time, command, arg1, arg2, arg3);
            sleep(wait_time);  // Чекање толку секунди

            // Извршување на наредбата
            execlp(command, command, arg1, arg2, arg3, NULL);  // Извршување на
наредбата со аргументите
            // Ако се стигне тука, значи не успеало извршувањето на наредбата
            perror("execlp не успеа");  // Испечатува грешка ако наредбата не
може да се изврши
            exit(1);
        }
        else if (pid < 0) {
            // Ако не може да се креира дете процес
            perror("Неуспешен fork");  // Испечатува грешка при креирање на
процес
            exit(1);
        }
    }

    // Родителот чека сите дете процеси да завршат
    for (int i = 0; i < k; i++) {
        wait(NULL);  // Чекање на било кој дете процес да заврши
    }

    printf("Сите наредби се извршени.\n");  // Печатење по завршувањето на сите
наредби
    return 0;
}
```

2.Да се напише програма во C која работи со процеси и нитки. Главната програма (родител процесот) треба да креира дете процес, на кого ќе му прати низа од 100 цели броеви. Дете процесот најпрво треба да ја пополни низата од 100 цели броеви со нули. Потоа, дете процесот треба да креира N нитки (N се внесува од тастатура во родител процесот), притоа на секоја нитка дете процесот и испраќа (како аргумент) случаен позитивен цел број „pobrzo

zavrshi prebaruvanjeto so deca procesi“, инаку печати „pobrzo zavrsi prebaruvanjetoK“ (најмногу 500).Секоја нитка прави „pobrzo zavrshi prebaruvanjeto so deca procesi“, инаку печати „pobrzo zavrsi prebaruvanjetoK“ промени во низата и потоа завршува со работа. Секоја една промена во низата значи случајно одбирање на еден елемент од низата и менување на неговата вредност. Првата половина од нитките ја менуваат вредноста на елементот со зголемување на неговата вредност за 1, додека пак втората половина на нитките ја намалуваат вредноста на елементот за 1. Откако ќе завршат со работа сите нитки, главната нитка (дете процесот) печати на екран колку елементи од низата ја имаат променето својата вредност (т.е. не се повеќе нула). Родител процесот завршува откако дете процесот ќе заврши. Генерирањето на случајни броеви се прави со помош на функцијата rand( ).

```c
#include <pthread.h>
#include <stdio.h>
#include <unistd.h>
#include <signal.h>
#include <sys/wait.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>

int niza[100];  // Низа со 100 елементи

// Функција која ја зголемува вредноста на случаен елемент во низата
void *zgolemi(void *id) {
    int K = *((int *)id);  // Бројот на операции што треба да се направат
    int i, j;
    for (i = 0; i < K; i++) {
        j = rand() % 100;  // Генерира случаен индекс на низа
        niza[j]++;  // Зголемување на вредноста на елементот
    }
    pthread_exit(NULL);  // Завршува со работа
}

// Функција која ја намалува вредноста на случаен елемент во низата
void *namali(void *id) {
    int K = *((int *)id);  // Бројот на операции што треба да се направат
    int i, j;
    for (i = 0; i < K; i++) {
```

```
        j = rand() % 100;  // Генерира случаен индекс на низа
        niza[j]--;  // Намалување на вредноста на елементот
    }
    pthread_exit(NULL);  // Завршува со работа
}

int main() {
    srand(time(0));  // Иницијализација на генератор за случајни броеви
    pthread_t nitki[100];  // Низа од нитки
    int K[100];  // Низа за бројот на операции по нитка
    int N, i;
    pid_t dete;

    dete = fork();  // Креирање на дете процес
    if (dete == 0) {
        // Дете процес
        for (i = 0; i < 100; i++) {
            niza[i] = 0;  // Пополнување на низата со 0
        }

        printf("Внеси број на нитки (N): ");
        scanf("%d", &N);  // Внесување на бројот на нитки

        // Креирање на нитки
        for (i = 0; i < N; i++) {
            K[i] = rand() % 500 + 1;  // Генерирање на случаен број на операции
за секоја нитка

            // Првата половина од нитките ја зголемуваат вредноста
            if (i < N / 2) {
                pthread_create(&nitki[i], NULL, zgolemi, (void *)(K + i));
            } else {
                // Втората половина од нитките ја намалуваат вредноста
                pthread_create(&nitki[i], NULL, namali, (void *)(K + i));
            }
        }

        // Чекање сите нитки да завршат
        for (i = 0; i < N; i++) {
            pthread_join(nitki[i], NULL);
        }

        // Родителскиот процес ја проверува бројката на променети елементи во
низата
        int vk = 0;
```

```
        for (i = 0; i < 100; i++) {
            if (niza[i] != 0) {
                vk++;  // Броење на елементите кои се променети
            }
        }
        printf("Вкупно променети елементи во низата: %d\n", vk);

    } else {
        // Родителски процес
        wait(NULL);  // Чекање дете процесот да заврши
        printf("Дете процесот заврши, сега и родителот ќе заврши.\n");
    }

    return 0;
}
```

3. Да се напише програма во C која работи со процеси и нитки. Главната програма (главната нитка) како аргумент добива име на влезна датотека. Главната нитка треба да креира онолку нитки колку што треба, така што, секоја нитка да добие по 10 линии од влезната датотека (нема повеќе од 1000 линии, а притоа последната нитка може да добие и помалку од 10 линии). Секоја една од нитките, ги изминува своите 10 линии од датотеката и брои колку има големи а колку мали букви. Откако ќе завршат нитките, главната нитка печати на екран колку секоја нитка нашла големи и мали букви и печати колку вкупно големи и мали букви биле пронајдени. Не е дозволено содржината на целата датотека да биде прочитана во низа т.е. секоја од нитките мора да работи со FILE * покажувач за изминување на датотеката т.е. на линиите од датотеката.

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

// Глобална променлива за име на датотека
char dat[30];

// Глобални променливи за чување на бројот на големи и мали букви по нитка
int mali[100], golemi[100];

// Функција која ќе биде извршена од секоја нитка
void *prebaraj(void *data) {
```

```
    // Преку аргументот добиваме индексот на нитката
    int i = *((int *)data);

    // Пресметуваме од која линија треба да започне и до која линија да стигне
нитката
    int from = i * 10;  // Почетна линија
    int to = (i + 1) * 10;  // Крајна линија (не вклучувајќи ја)

    char c;
    // Иницијализираме броевите за големи и мали букви
    mali[i] = 0;
    golemi[i] = 0;

    // Отворање на датотека за читање
    FILE *fp = fopen(dat, "r");
    if (fp == NULL) {
        printf("Нитка %d не успеа да ја отвори датотеката %s\n", i, dat);
        pthread_exit(NULL);  // Ако не може да ја отвори датотеката, нитката
завршува
    }

    // Прескокнување на линиите до започнување на обработката за оваа нитка
    int current_line = 0;
    while (current_line < from) {
        // Читање по една линија за да стигнеме до местото каде нитката треба да
започне
        while ((c = fgetc(fp)) != EOF && c != '\n');
        current_line++;
    }

    // Обработка на линиите кои припаѓаат на оваа нитка
    while (current_line < to && (c = fgetc(fp)) != EOF) {
        // Ако е нов ред, ја зголемуваме бројката на линијата
        if (c == '\n') {
            current_line++;
        } else if (isalpha(c)) {  // Проверуваме дали карактерот е буква
            if (isupper(c)) {  // Ако е голема буква
                golemi[i]++;  // Увекуваме број на големи букви за оваа нитка
            } else {  // Ако е мала буква
                mali[i]++;  // Увекуваме број на мали букви за оваа нитка
            }
        }
    }

    // Затворање на датотеката по обработката на линиите
```

```c
    fclose(fp);

    // Завршување на нитката
    pthread_exit(NULL);
}

int main(int argc, char *argv[]) {
    // Декларирање на низи за нитки и за аргументи за секоја нитка
    pthread_t nitki[100];
    int data[100];
    char c;
    int linii = 0, j;

    // Проверка за правилен број на аргументи
    if (argc < 2) {
        printf("Премалку аргументи\n");
        return 0;
    }

    // Пренос на аргументот за име на датотека во глобалната променлива
    strcpy(dat, argv[1]);

    // Отворање на датотека за броење на линиите
    FILE *fp = fopen(argv[1], "r");
    if (fp == NULL) {
        printf("Не можам да ја отворам датотеката\n");
        return 0;
    }

    // Броење на линиите во датотеката
    while ((c = fgetc(fp)) != EOF) {
        if (c == '\n') {
            linii++;
        }
    }
    fclose(fp);  // Затворање на датотеката по броење на линиите

    // Пресметување на бројот на нитки што ќе треба да се креираат
    // Секој нитка ќе обработува до 10 линии, ако има помалку линии во последната
нитка, таа ќе обработува помалку
    int brN = linii % 10 != 0 ? linii / 10 + 1 : linii / 10;
    printf("Во датотеката %s има %d линии и ќе креирам %d нитки\n", dat, linii,
brN);
```

```
    // Креирање на нитки кои ќе се грижат за обработката на секој дел од
датотеката
    for (j = 0; j < brN; j++) {
        // Пренос на индекс за секоја нитка
        data[j] = j;
        pthread_create(&nitki[j], NULL, prebaraj, (void *)(data + j));
    }

    // Чекате сите нитки да завршат
    for (j = 0; j < brN; j++) {
        pthread_join(nitki[j], NULL);
    }

    // Печатење на резултатите по завршување на сите нитки
    printf("Завршија сите нитки\n");

    // Променливи за вкупниот број на големи и мали букви
    int vkg = 0, vkm = 0;

    // Печатење на резултатите за секоја нитка
    for (j = 0; j < brN; j++) {
        printf("Нитката %d најде %d големи и %d мали букви\n", j, golemi[j],
mali[j]);
        // Собирање на вкупниот број на букви
        vkg += golemi[j];
        vkm += mali[j];
    }

    // Печатење на вкупниот број на големи и мали букви
    printf("Вкупно големи букви: %d, вкупно мали букви: %d\n", vkg, vkm);

    return 0;
}
```

4.Да се напише програма во C која работи со процеси и нитки. Главната програма (родител процесот) пополнува низа од 1000 броеви на случаен начин со помош на rand() функцијата. Потоа, креира два деца процеси, така што, двата деца процеси вршат пребарување на бројот 0 во низата од 1000 броеви. Првото дете процес, пребарувањето го прави со помош на 10 деца процеси, додека пак второто дете, пребарувањето го прави со 10 нитки. Секоја нитка/дете процес добива дел од низата што треба да го пребара бројот 0 и печати на екран колку пати е пронајден бројот кај соодветната нитка/дете процес. Родител процесот чека да завршат двете деца процеси и

на екран печати кое од двете завршило прво. Доколку прво заврши дете процесот кое пребарувањето го прави со помош на деца процеси, тогаш на екран се печати „pobrzo zavrshi prebaruvanjeto so deca procesi“, инаку печати „pobrzo zavrsi prebaruvanjetopobrzo zavrshi prebaruvanjeto so deca procesi“, инаку печати „pobrzo zavrshi prebaruvanjeto so deca procesi“, инаку печати „pobrzo zavrsi prebaruvanjetopobrzo zavrsi prebaruvanjeto so nitki“.

```c
#include <pthread.h>
#include <stdio.h>
#include <unistd.h>
#include <signal.h>
#include <sys/wait.h>
#include <stdlib.h>

#define N 1000  // Големина на низата

// Низа со 1000 броеви која ќе биде пополнета со случајни вредности
int niza[N];

// Низа за чување на резултатите од пребарувањето на секоја нитка
int odgovor[10];

// Функција која ќе ја извршуваат нитките за пребарување на 0
void *prebaraj(void *id) {
    int i = *((int *)id);  // Пренос на индексот на нитката
    int j, found = 0;  // Променлива за чување на бројот на пронајдени 0
    // Пребарување на делот од низата кој е доделен на оваа нитка
    for (j = i * 100; j < i * 100 + 100; j++) {
        if (niza[j] == 0) {  // Ако се најде 0, го зголемуваме бројот на
пронајдени 0
            found++;
        }
    }
    odgovor[i] = found;  // Запишуваме резултатот во глобалната низа
    pthread_exit(NULL);  // Завршување на нитката
}

int main() {
    srand(time(0));  // Иницијализирање на генераторот за случајни броеви

    int thread_data[10];  // Низа за пренос на индексите на нитките
    int i, found, j;

    // Креирање на случајни броеви за низата
```

```
    for (i = 0; i < N; i++) {
        niza[i] = rand() % 10;  // Секој број во низата е случаен број од 0 до 9
    }

    pid_t deca[10];  // Масив за чување на ид-ата на детските процеси
    pid_t deteP, deteN;  // Ид-ата на процесите за пребарување со процеси и нитки

    // Креираме процес за пребарување со процеси
    deteP = fork();
    if (deteP == 0) {
        // Ова е детето за пребарување со процеси
        for (i = 0; i < 10; i++) {
            deca[i] = fork();  // Креираме 10 детски процеси
            if (deca[i] == 0) {  // Ако сме во детскиот процес
                break;
            }
        }

        if (i < 10) {
            // Ова е еден од 10-те детски процеси кои пребаруваат во дел од
низата
            found = 0;
            for (j = i * 100; j < i * 100 + 100; j++) {
                if (niza[j] == 0) {  // Пребарување на бројот 0
                    found++;
                }
            }
            // Печатење на резултатот за секој детски процес
            printf("Заврши со пребарување детето со ID=%d и најде %d\n", i,
found);

            // Запишување на резултатот во датотека
            FILE *fp = fopen("com.txt", "a+");
            if (fp == NULL) {
                printf("Дете процесот %d не успеа да ја отвори датотеката\n", i);
                return 0;
            }
            fprintf(fp, "%d ", found);  // Запишуваме колку пати е пронајден 0
            fclose(fp);
        } else {
            // Родителскиот процес во детето чека да завршат сите 10 дечиња
            for (i = 0; i < 10; i++) {
                wait(NULL);  // Чекање за секој детски процес
            }
            // Читање од датотеката за да се соберат резултатите од сите процеси
```

```
            int vkupno = 0;
            FILE *fp = fopen("com.txt", "r");
            if (fp == NULL) {
                printf("Дете процесот не успеа да ја отвори датотеката\n");
                return 0;
            }
            // Прочитување на резултатите за секој од 10-те детски процеси
            for (j = 0; j < 10; j++) {
                fscanf(fp, "%d", &found);
                printf("Пронајдено %d\n", found);
                vkupno += found;
            }
            fclose(fp);
            // Печатење на вкупниот број на пронајдени 0
            printf("Дете процеси ги дочекаа сите да завршат. Вкупно пронајдено
%d\n", vkupno);
        }
    } else {
        // Родителскиот процес креира второ дете за пребарување со нитки
        deteN = fork();
        if (deteN == 0) {
            // Ова е детето за пребарување со нитки
            for (j = 0; j < 10; j++) {
                thread_data[j] = j;  // Пренос на индексите на нитките
                pthread_create(&nitki[j], NULL, prebaraj, (void *)(thread_data +
j));  // Креирање на нитки
            }
            for (j = 0; j < 10; j++) {
                pthread_join(nitki[j], NULL);  // Чекање сите нитки да завршат
            }
            // Собирање на резултатите за пребарувањето
            int vkupno = 0;
            for (j = 0; j < 10; j++) {
                vkupno += odgovor[j];  // Сумираме колку пати е пронајден 0
            }
            printf("Јас сум дете со нитки и ги дочекав сите нитки да завршат.
Вкупно пронајдено е %d\n", vkupno);
        } else {
            // Родителскиот процес чека да завршат двата детски процеси
            wait(NULL);  // Чекање на првото дете
            wait(NULL);  // Чекање на второто дете

            // Испечатување кој од детските процеси завршил прв
            if (waitpid(deteP, NULL, WNOHANG) == deteP) {
                // Ако првиот детски процес завршил прв
```

```
                printf("Побрзо заврши пребарувањето со процеси.\n");
            } else {
                // Ако вториот детски процес завршил прв
                printf("Побрзо заврши пребарувањето со нитки.\n");
            }
            // Родителскиот процес чека двата деца да завршат
            printf("Родител дочека двата деца да завршат и тој завршува.\n");
        }
    }
    return 0;
}
```

5. Да се напише програма во C која работи со процеси и нитки. Главната програма (родител процесот) добива низа од наредби кои треба да ги изврши како аргументи од командна линија. Родител процесот треба, најпрво, да креира онолку деца процеси колку што има наредби наведено од командна линија (наредбите се без аргументи). Пота, треба да креира онолку нитки, колку што има наредби, така што, секоја нитка ќе чека и ќе брои колку секунди му било потребно на соодветниот процес да заврши. Тоа значи дека, првата нитка ќе биде задолжена за првата наредба т.е. за првиот процес, втората за вториот и т.н. Секоја нитка брои колку време се извршувал нејзиниот процес (наредба) и кога ќе заврши кажува колку вкупно секунди му требало да заврши, а потоа и самата нитка завршува. Откако ќе завршат сите процеси/нитки, тогаш главниот процес/нитка печати на екран колку време и требало на секоја наредба да се изврши.

```
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <math.h>
#include <string.h>  // За користење на strcpy

#define MAX_COMMANDS 100  // Ограничување на бројот на наредби

int sekundi[MAX_COMMANDS];  // Низа за чување на времето за секоја наредба
pid_t deca[MAX_COMMANDS];  // Низа за чување на процесите

// Функција која се извршува во нитките за броење на времето на процесите
void *broi(void *t) {
    int i = *((int *)t);  // Пренос на индексот на наредбата
    sekundi[i] = 0;  // Иницијализирај ја секунда за оваа наредба
    pid_t dete;
```

```
    while (1) {
        // Проверка дали процесот завршил
        if (deca[i] == waitpid(deca[i], NULL, WNOHANG)) {
            break;  // Ако процесот завршил, прекинуваме ја нитката
        }
        sleep(1);  // Спијеме 1 секунда
        sekundi[i]++;  // Уголемувањето на времето на извршување на процесот
    }
    pthread_exit(NULL);  // Завршување на нитката
}

int main(int argc, char *argv[]) {
    int data[MAX_COMMANDS];  // Низа за пренос на индексите во нитките
    pthread_t nitki[MAX_COMMANDS];  // Низа за нитките
    int pom;
    char poms[100];  // Променлива за помош при замена на наредбите
    int n;

    n = argc - 1;  // Влегуваат само наредбите како аргументи (без првиот
аргумент кој е името на програмата)

    // Проверка дали има понудено наредби
    if (n == 0) {
        printf("Нема наредби за извршување.\n");
        return 1;
    }

    // Креирање на процеси за секоја наредба
    for (int i = 0; i < n; i++) {
        deca[i] = fork();  // Креирање на нов процес
        if (deca[i] == 0) {
            // Во детскиот процес, извршуваме конкретната наредба
            execlp(argv[i + 1], argv[i + 1], NULL);
            // Ако execlp не успее, печатиме порака за грешка
            printf("Неуспешно се изврши наредбата %s\n", argv[i + 1]);
            exit(1);
        }
    }

    // Во родителскиот процес креираме нитки кои ќе ја броят времето за секоја
наредба
    for (int i = 0; i < n; i++) {
        data[i] = i;  // Пренос на индекс на наредбата за која нитката ќе се
грижи
```

```
        pthread_create(&nitki[i], NULL, broi, (void *)(data + i));  // Креирање
на нитка за секоја наредба
    }

    // Чекање сите нитки да завршат
    for (int i = 0; i < n; i++) {
        pthread_join(nitki[i], NULL);  // Чекање за секоја нитка
    }

    // Сортирање на наредбите по времето на извршување
    for (int i = 0; i < n; i++) {
        for (int j = i + 1; j < n; j++) {
            if (sekundi[i] > sekundi[j]) {
                // Размена на времето на извршување на наредбите
                pom = sekundi[i];
                sekundi[i] = sekundi[j];
                sekundi[j] = pom;

                // Размена на наредбите во низата
                strcpy(poms, argv[i + 1]);
                strcpy(argv[i + 1], argv[j + 1]);
                strcpy(argv[j + 1], poms);
            }
        }
    }

    // Печатење на наредбите и времето на нивното извршување
    for (int i = 0; i < n; i++) {
        printf("Наредбата %s и беа потребни %d секунди за извршување\n", argv[i +
1], sekundi[i]);
    }

    return 0;
}
```

6. Да се напише програма во C коа работи со процеси и нитки. Програмата (главната нитка) како аргумент од командна линиа добива име на влезна датотека. Во влезната датотека, во секо ред одделно, се сместени IP адреси (формат: x.x.x.x, каде x е бро од 0 до 255). Главната нитка треба да кренра онолку нитки колку што има IP адреси во датотеката. Секоа нитка, пресметува колку можни IP адреси може да генерира, почнуваки од IP адресата што а добива, до 255 броот за Shary Alska секо дел од x.y.z.k. Генерираьето на IP адресите се прави така што к оди до 255, па потоа се зголемува z, а к почнува

од 0 и т.н. Секоа нитка треба да пресмета колку може да генерира IP адреси. На краз, главниот процес(нитка) печати на екран, колку вкупно (од сите нитки) може да се генерираат IP адреси.

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

// Максимален број на IP адреси
#define MAX_IP 1000

int vkupno_ip = 0;

// Функција која ја извршуваат нитките
void *broj_ip(void *arg) {
    char *ip = (char *)arg;
    int a, b, c, d;

    // Парсирање на IP адресата
    sscanf(ip, "%d.%d.%d.%d", &a, &b, &c, &d);

    // Пресметување на бројот на можни IP адреси
     int broj = 0;
    for (int i = c; i <= 255; i++) {
        for (int j = (i == c ? d : 0); j <= 255; j++) {
            broj++;
        }
    }

    // Додавање на бројот во глобалната променлива (без мутекс)
    vkupno_ip += broj;

    pthread_exit(NULL);
}

int main(int argc, char *argv[]) {
    if (argc < 2) {
        printf("Користење: %s <име_на_датотека>\n", argv[0]);
        return 1;
    }

    // Отвори ја датотеката
    FILE *datoteka = fopen(argv[1], "r");
    if (datoteka == NULL) {
```

```
        perror("Не може да се отвори датотеката");
        return 1;
    }

    // Читање на IP адреси од датотеката
    char *ip_adresi[MAX_IP];
    int broj_ip_adresi = 0;
    char linija[20];

    while (fgets(linija, sizeof(linija), datoteka)) {
        linija[strcspn(linija, "\n")] = 0; // Отстрани нов ред
        ip_adresi[broj_ip_adresi] = strdup(linija); // Зачувај ја IP адресата
        broj_ip_adresi++;
    }
    fclose(datoteka);

    // Креирање на нитки
    pthread_t nitki[MAX_IP];
    for (int i = 0; i < broj_ip_adresi; i++) {
        pthread_create(&nitki[i], NULL, broj_ip, (void *)ip_adresi[i]);
    }

    // Чекање на сите нитки
    for (int i = 0; i < broj_ip_adresi; i++) {
        pthread_join(nitki[i], NULL);
        free(ip_adresi[i]); // Ослободување на меморијата
    }

    // Печати го вкупниот број на IP адреси
    printf("Вкупно можни IP адреси: %d\n", vkupno_ip);

    return 0;
}
```