

Proiect Programarea calculatoarelor si limbaje de programare II

Profesor Coordonator

Neculoiu Giorgian

Olteanu Gabriela

Student

Dincov Cristian-Eugen

Simulare de coadă de așteptare

Profesor Coordonator

Neculoiu Giorgian

Olteanu Gabriela

Student

Dincov Cristian-Eugen

INTRODUCERE

Motivarea alegere tema

Sistemele de cozi sunt întâlnite în numeroase domenii ale activității umane – de la administrația publică, instituții bancare și spitale, până la transporturi, telecomunicații și comerț electronic. Gestionarea ineficientă a acestor cozi duce nu doar la nemulțumirea utilizatorilor, ci și la pierderi semnificative de timp, resurse și productivitate.

Alegerea temei „**Simularea unei cozi de așteptare**” vine ca răspuns la nevoia de a înțelege și modela aceste procese complexe, oferind soluții concrete de optimizare. Am considerat relevantă această temă întrucât simulează un scenariu realist cu care ne confruntăm cu toții în viața de zi cu zi: așteptarea la un ghișeu. Dincolo de această simplitate aparentă, se ascund numeroase provocări legate de alocarea eficientă a resurselor, prioritizarea corectă a clienților și adaptarea la variațiile de trafic din timpul zilei.

În plus, introducerea **priorității clienților** în simulare adaugă un strat suplimentar de complexitate și realism. În multe instituții, anumite categorii de persoane – cum ar fi vârstnicii, persoanele cu dizabilități sau clienții premium – beneficiază de tratament preferențial. Implementarea unui sistem care ține cont de astfel de priorități permite evaluarea impactului acestor politici asupra timpului mediu de așteptare, asupra echității sistemului și asupra gradului general de satisfacție.

Un alt motiv important pentru alegerea acestei teme este caracterul său interdisciplinar. Proiectul îmbină noțiuni teoretice din teoria cozilor și ingineria sistemelor cu implementarea practică în limbajul de programare C.

Obiective propuse

Proiectul de față își propune să simuleze funcționarea unui sistem de cozi la ghișeu, în care clienții sunt generați aleatoriu, au niveluri diferite de prioritate, iar deservirea se realizează de către un număr configurabil de ghișee. În realizarea acestui sistem, au fost stabilite următoarele obiective principale:

Modelarea comportamentului unui sistem de cozi

Reprezentarea clienților printr-o structură de date care să conțină informații precum ora sosirii, timpul de servire, prioritatea și timpul de finalizare.

Reprezentarea ghișeelelor ca unități de deservire cu timp de disponibilitate individual.

Generarea aleatorie a clienților

Simularea unui aflax variabil de clienți, cu intensificare în intervalele de timp corespunzătoare orelor de vârf.

Alocarea fiecărui client unui tip de prioritate: normal, VIP, sau urgent, pe baza unei distribuții pseudo-aleatorii.

Implementarea unui mecanism de prioritizare

Selectarea clientului următor pentru fiecare ghișeu pe baza celui mai ridicat nivel de prioritate disponibil în acel moment.

Tratarea corectă a simultaneităților și a cazurilor de egalitate printr-un sistem clar de selecție (primul venit, primul servit în cadrul aceleiași priorități).

Simularea funcționării sistemului într-un interval de timp determinat

Posibilitatea de a configura durata totală a simulării (în minute).

Repartizarea automată a clienților în funcție de disponibilitatea ghișeelelor și timpul curent de simulare.

Calculul și afișarea statisticilor relevante

Determinarea numărului total de clienți generați și deserviți.

Calcularea timpului mediu și maxim de așteptare al clienților.

Identificarea minutului cel mai aglomerat din timpul simulării.

Vizualizarea dinamicii clienților

Afișarea unui bar chart textual care să reprezinte vizual numărul de clienți generați în fiecare minut.

Exportarea rezultatelor într-un fișier extern (statistici.txt) pentru analiză ulterioară.

Oferirea de flexibilitate utilizatorului

Permisiunea utilizatorului de a alege numărul de ghișee (între 1 și 10).

Validarea intrărilor și oferirea unor valori implicite în cazul în care datele introduse sunt invalide.

Asigurarea scalabilității codului

Utilizarea de constante (MAX_CLIENTI, MAX_GHISEE, MAX_MINUTE) pentru a permite ajustarea rapidă a limitelor sistemului.

Organizarea codului pe funcții modulare pentru a facilita extinderea proiectului cu funcționalități noi (ex: interfață grafică, salvare în baze de date, generare automată de rapoarte etc.).

TEHNOLOGII UTILIZATE

Pentru implementarea proiectului de simulare a unei cozi la ghișee, au fost utilizate o serie de tehnologii și instrumente software care oferă eficiență, flexibilitate și un control detaliat asupra execuției programului. Alegerea acestor tehnologii s-a făcut ținând cont de complexitatea algoritmului propus, de cerințele educaționale ale proiectului și de dorința de a obține un cod clar, performant și portabil.

Limbajul de programare C

Limbajul C este unul dintre cele mai vechi și mai utilizate limbaje de programare din lume, fiind baza pentru multe alte limbaje moderne. Alegerea sa pentru acest proiect a fost indicată la începutul anului dar mai jos avem câteva proprietăți interesante.

1. **Eficiență și performanță ridicată:** C permite scrierea unui cod rapid și eficient, cu acces direct la nivelul hardware, ceea ce este esențial pentru simulări intensive sau cu resurse limitate.
2. **Control total asupra memoriei:** Prin utilizarea pointerilor și a funcțiilor de alocare dinamică, programatorul are un control foarte precis asupra modului în care memoria este folosită.
3. **Ușurință în simulări secvențiale și logice:** Limbajul C se pretează foarte bine pentru scrierea de simulări pas-cu-pas, unde fiecare unitate de timp este procesată în ordine și fiecare eveniment este tratat explicit.
4. **Portabilitate:** Codul scris în C poate fi compilat și rulat pe o gamă largă de sisteme de operare fără modificări semnificative.
5. **Ideal pentru formarea gândirii algoritmice:** Dat fiind că limbajul nu oferă funcționalități de nivel înalt (precum colecții dinamice sau clase), studentul este forțat să înțeleagă și să construiască explicit structuri fundamentale precum cozi, stive, tabele sau simulări de evenimente.

Biblioteci standard C

În cadrul proiectului au fost utilizate următoarele biblioteci standard din C, esențiale pentru funcționalitățile de bază:

- **stdio.h** – Bibliotecă pentru input/output, folosită pentru:
 - Afișarea informațiilor către utilizator în timpul simulării.
 - Scrierea rezultatelor și a statisticilor într-un fișier text.
- **stdlib.h** – Oferă funcții pentru:
 - Generarea de numere aleatoare cu `rand()`, necesare pentru simularea apariției aleatoare a clienților.
 - Alocare și eliberare dinamică a memoriei (dacă este nevoie).
 - Conversii de tipuri și sortări.
- **time.h** – Utilizată pentru:
 - Inițializarea generatorului aleatoriu cu `srand(time(NULL))`, asigurând variația rezultatelor la fiecare rulare a simulării.
 - Obținerea timpului curent pentru raportare sau măsurarea performanței.

Structuri de date personalizate (struct)

Structurile în C au fost folosite pentru a modela entitățile principale ale proiectului:

1. **struct Client** – conține atributele esențiale ale unui client, cum ar fi timpul de sosire, durata serviciului, prioritatea, timpul de așteptare și starea curentă.
2. **struct Ghiseu** – conține starea fiecărui ghișeu, în special timpul rămas până când acesta devine disponibil pentru un nou client.

Prin utilizarea acestor structuri, codul a devenit mai clar și mai ușor de extins.

Algoritmi și tehnici de simulare

Algoritmii folosiți în proiect includ:

- **Simularea pas-cu-pas (minute):** Sistemul avansează în pași discreți de timp (de la minutul 0 la minutul maxim specificat), iar la fiecare pas se:
 - Generează aleatoriu noi clienți.
 - Actualizează starea ghișeelor (libere/ocupate).
 - Alocă clienți pe ghișee în funcție de disponibilitate și prioritate.
- **Selecție cu prioritate:** Clienții pot avea priorități diferite, fiind selectați preferențial în funcție de tipul lor (normal, prioritar, urgență).
- **Calcul de statistici:** La finalul simulării se calculează automat:
 - Timpul mediu și maxim de așteptare.
 - Numărul total de clienți serviți.
 - Gradul de ocupare al ghișeelor.
 - Minutul cu cea mai mare aglomerație.

Mediu de dezvoltare și compilare

Pentru scrierea și testarea codului s-au utilizat:

- **Editor de cod:** Code::Blocks sau Visual Studio Code – cu suport pentru limbajul C, completare automată, evidențiere sintactică și rulare directă.
- **Compiler GCC:** Utilizat pentru compilarea programului, oferă suport complet pentru standardele limbajului C și este disponibil pe toate platformele majore (Windows, Linux, macOS).
- **Terminal/Consolă:** Rularea programului se face în linie de comandă, permițând observarea clară a procesului de simulare și a rezultatelor generate.

Fișiere de ieșire și stocare

Rezultatele obținute în timpul simulării sunt salvate într-un fișier text:

- **statistici.txt** – Conține:
 - Numărul total de clienți generați și deserviți.
 - Timpul maxim și mediu de așteptare.
 - Minutele de vârf și informații utile pentru analiză ulterioară.

Fișierul poate fi deschis cu orice editor text sau importat într-un program de tip Excel pentru vizualizare grafică.

Modularitate și claritate

Programul a fost organizat modular, cu funcții separate pentru:

- Generarea clienților.
- Alocarea la ghișeu.
- Procesarea fiecărui minut.
- Calculul statisticilor.

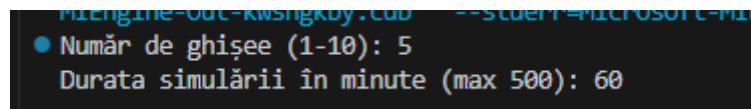
Această abordare asigură o mai bună întreținere, testare și extindere a codului.

STUDIU DE CAZ

În primul rând, voi prezenta ce se întâmplă când codul rulează și apoi voi explica mai jos ce am programat în C cu explicații.

Rulare cod

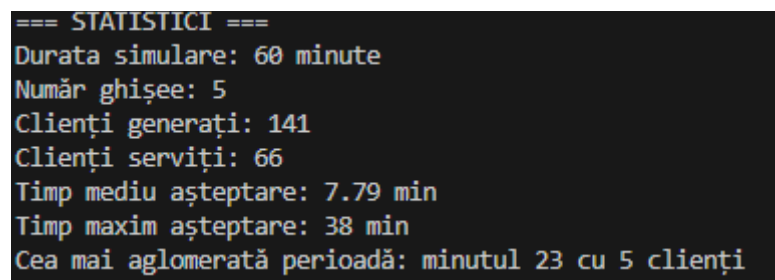
Când dăm pe Run pentru a porni programul, vom fi nevoiți să introducem niste date în terminalul Visual Studio Code.



```
MIEngine-001-kwsngkby.cub --STUENT=MICROSOFT-MIL  
● Număr de ghișee (1-10): 5  
Durata simulării în minute (max 500): 60
```

Figura 3.1 – Terminal cu date introduse de utilizator

Programul preia datele introduse de utilizator și le va folosi pentru a genera statistici



```
=== STATISTICI ===  
Durata simulare: 60 minute  
Număr ghișee: 5  
Clienți generați: 141  
Clienți serviți: 66  
Timp mediu așteptare: 7.79 min  
Timp maxim așteptare: 38 min  
Cea mai aglomerată perioadă: minutul 23 cu 5 clienți
```

Figura 3.2 – Poza Statistici

Aceste statistici sunt salvate și într-un fișier numit statistici.txt pe care îl vom găsi în folder-ul programului cu datele afișate în terminal în caz ca utilizator are nevoie de el.

Apoi avem un fel de interfata grafica facuta in terminal pentru a vedea in fiecare minut cati oameni se afla la coada.

```
=== GRAFIC NUMĂR CLIENȚI / MINUT ===  
Minut 0: ## (2)  
Minut 1: # (1)  
Minut 2: ## (2)  
Minut 3: ## (2)  
Minut 4: # (1)  
Minut 5: # (1)  
Minut 6: ## (2)  
Minut 7: # (1)  
Minut 8: # (1)  
Minut 9: ## (2)
```

Figura 3.3 – Poza Grafic Terminal

In poza se poate vedea clar, oamenii sunt reprezentati cu simbolul „#” si numarul de oameni ce se afla la coada este scris in paranteza.

Implementare pas cu pas

- Definirea Structurilor de Date

Pentru o simulare realistă și bine organizată, s-au definit două structuri esențiale:

```
12 typedef struct {  
13     int id;  
14     int ora_sosire;  
15     int timp_servire;  
16     int timp_inceput_servire;  
17     int timp_finalizare;  
18     int tip_prioritate;  
19     int servit_de_ghiseu;  
20 } Client;  
21  
22 typedef struct {  
23     int timp_liber;  
24 } Ghiseu;
```

Figura 3.4 – Structura Client & Ghiseu

Explicații:

- Client conține toate datele esențiale pentru fiecare client simulat.

- Ghiseu reține informații despre timpul până când ghișeul este disponibil și clientul deservit în prezent.

Inițializarea simulării

Programul începe prin a cere de la utilizator datele de intrare esențiale:

```
int main() {
    srand(time(NULL));

    int nr_ghisee, durata_simulare;

    printf("Număr de ghișee (1-10): ");
    scanf("%d", &nr_ghisee);
    if (nr_ghisee <= 0 || nr_ghisee > MAX_GHISEE) nr_ghisee = 3;

    printf("Durata simulării în minute (max %d): ", MAX_MINUTE);
    scanf("%d", &durata_simulare);
    if (durata_simulare <= 0 || durata_simulare > MAX_MINUTE) durata_simulare = 60;

    simuleaza(durata_simulare, nr_ghisee);
    afiseaza_statistici(durata_simulare, nr_ghisee);
    afiseaza_bar_chart(durata_simulare);

    printf("\nRezultatele au fost salvate în fișierul 'statistici.txt'.\n");

    return 0;
}
```

Figura 3.5 – Cod Main

Generarea aleatoare a clienților

La fiecare minut se simulează sosirea aleatoare a unuia sau mai multor clienți.

Aceasta se face folosind funcția rand():

```
void genereaza_clienti(int minut_curent, int durata_simulare) {
    int clienti_pe_tura = (minut_curent >= durata_simulare/3 && minut_curent <= 2*durata_simulare/3) ? 3 + rand() % 3 : 1 + rand() % 2;
    clienti_pe_minut[minut_curent] = clienti_pe_tura;
    for (int i = 0; i < clienti_pe_tura && nr_clienti < MAX_CLIENTI; i++) {
        Client c;
        c.id = nr_clienti + 1;
        c.ora_sosire = minut_curent;
        c.tip_prioritate = random_tip_prioritate();
        c.timp_servire = random_timp_servire(c.tip_prioritate);
        c.timp_inceput_servire = -1;
        c.timp_finalizare = -1;
        clienti[nr_clienti++] = c;
    }
}
```

Figura 3.6 – Cod funcție generare clienti

Alocarea clienților la ghișee

În fiecare minut, sistemul verifică dacă ghișeele sunt libere și, dacă există clienți în coadă, încearcă să-i aloc

```
int alege_client(int minut, int *servit) {
    int max_prioritate = -1;
    int index = -1;
    for (int i = 0; i < nr_clienti; i++) {
        if (!servit[i] && clienti[i].ora_sosire <= minut) {
            if (clienti[i].tip_prioritate > max_prioritate) {
                max_prioritate = clienti[i].tip_prioritate;
                index = i;
            }
        }
    }
    return index;
}
```

Figura 3.7 – Cod alege_client

```
void simuleaza(int durata_simulare, int nr_ghisee) {
    int servit[MAX_CLIENTI] = {0};
    for (int minut = 0; minut < durata_simulare; minut++) {
        genereaza_clienti(minut, durata_simulare);
        for (int g = 0; g < nr_ghisee; g++) {
            if (ghisee[g].timp_liber <= minut) {
                int idx = alege_client(minut, servit);
                if (idx != -1) {
                    clienti[idx].timp_inceput_servire = minut;
                    clienti[idx].timp_finalizare = minut + clienti[idx].timp_servire;
                    clienti[idx].servit_de_ghiseu = g + 1;
                    ghisee[g].timp_liber = clienti[idx].timp_finalizare;
                    servit[idx] = 1;

                    int astept = minut - clienti[idx].ora_sosire;
                    timpul_total_asteptare += astept;
                    if (astept > timpul_maxim_asteptare)
                        timpul_maxim_asteptare = astept;
                    clienti_serviti++;
                }
            }
        }
    }
}
```

Figura 3.8 – Cod simuleaza

Calculul statisticilor finale

La finalul simulării se calculează

```
void afiseaza_statistici(int durata_simulare, int nr_ghisee) {
    FILE *f = fopen("statistici.txt", "w");

    printf("\n=== STATISTICI ===\n");
    fprintf(f, "=== STATISTICI ===\n");
    printf("Durata simulare: %d minute\n", durata_simulare);
    printf("Număr ghisee: %d\n", nr_ghisee);
    printf("Clienți generați: %d\n", nr_clienti);
    printf("Clienți serviți: %d\n", clienti_serviti);
    printf("Timp mediu așteptare: %.2f min\n", clienti_serviti > 0 ? (float)timpul_total_asteptare / clienti_serviti : 0);
    printf("Timp maxim așteptare: %d min\n", timpul_maxim_asteptare);

    fprintf(f, "Clienți generați: %d\n", nr_clienti);
    fprintf(f, "Clienți serviți: %d\n", clienti_serviti);
    fprintf(f, "Timp mediu: %.2f min\n", clienti_serviti > 0 ? (float)timpul_total_asteptare / clienti_serviti : 0);
    fprintf(f, "Timp maxim: %d min\n", timpul_maxim_asteptare);

    int max_min = 0;
    for (int i = 1; i < durata_simulare; i++) {
        if (clienti_pe_minut[i] > clienti_pe_minut[max_min])
            max_min = i;
    }

    printf("Cea mai aglomerată perioadă: minutul %d cu %d clienți\n", max_min, clienti_pe_minut[max_min]);
    fprintf(f, "Minutul de vârf: %d (%d clienți)\n", max_min, clienti_pe_minut[max_min]);
    fclose(f);
}
```

Figura 3.9 – Cod subprogram afiseaza_statistici

```
void afiseaza_bar_chart(int durata_simulare) {
    printf("\n=== GRAFIC NUMĂR CLIEȚI / MINUT ===\n");
    for (int i = 0; i < durata_simulare; i++) {
        printf("Minut %2d: ", i);
        for (int j = 0; j < clienti_pe_minut[i]; j++)
            printf("#");
        printf(" (%d)\n", clienti_pe_minut[i]);
    }
}
```

Figura 3.10 – Cod subprogram afisare grafica terminal

CONCLUZIE

Pentru a extinde și a îmbunătăți acest proiect, se pot implementa următoarele funcționalități și optimizări:

- 1. Introducerea unei interfețe grafice (GUI):**
O interfață vizuală realizată în C (folosind librării precum GTK sau Qt) sau în alte limbaje ar facilita interacțiunea cu simularea, afișând în timp real starea ghișeelor, coada de așteptare și statistici grafice.
- 2. Simularea unui număr mai mare de ghișee și clienți:**
Optimizarea algoritmilor pentru a gestiona volume mari de date și simulări pe durate extinse, cu posibilitatea de a analiza efectele unor politici diferite de alocare.
- 3. Implementarea unor strategii avansate de prioritizare:**
De exemplu, introducerea unor reguli dinamice de prioritizare în funcție de timpul petrecut în coadă, sau clienți care pot schimba prioritatea în timp (de exemplu, cazuri de urgență ce devin mai presante).
- 4. Includerea factorului uman și alatoriu mai complex:**
Introducerea unor variabile precum timpul mediu de servire dependent de operatorul ghișeului, erori de procesare, sau întreruperi neașteptate.
- 5. Generarea de rapoarte detaliate și export în formate diverse:**
De exemplu, export în format Excel sau CSV pentru analize ulterioare, grafice interactive și comparații între diferite setări de simulare.
- 6. Simularea scenariilor multi-ghișeu cu cooperare și redistribuire:**
În care ghișeele se pot ajuta reciproc, redirectionând clienții pentru a reduce timpii de așteptare globali.
- 7. Aplicarea tehnicilor de optimizare și învățare automată:**
Folosind datele generate, se pot aplica algoritmi care să optimizeze programul ghișeelor, numărul de personal necesar în funcție de orele din zi, sau să prezică perioadele de aglomerație.

În concluzie, proiectul actual reprezintă un punct de plecare solid în domeniul simulărilor de cozi și al managementului resurselor, oferind un cadru clar pentru dezvoltări viitoare ce pot contribui la eficientizarea proceselor și la creșterea satisfacției clienților în sisteme reale.

BIBLIOGRAFIE

1 - Sedgewick, Robert, și Kevin Wayne. *Algorithms*. 4th Edition. Addison-Wesley, 2011.

– Referință importantă pentru algoritmi și structuri de date, inclusiv cozi și gestionarea priorităților.

2 – Surse online

- Tutorialspoint. “C Programming Tutorial.”
<https://www.tutorialspoint.com/cprogramming/index.htm>
- GeeksforGeeks. “Data Structures and Algorithms.”
<https://www.geeksforgeeks.org/data-structures/>
- Stack Overflow. <https://stackoverflow.com/> — pentru soluții și explicații legate de probleme de programare.

ANEXE

Link unde se afla programul incarcat pe GitHub:

<https://github.com/dincovercristian08/dincovercristian/tree/main/proiect%20coada>

Mai jos voi pune codul complet in C

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

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

```
#include <time.h>
```

```
#define MAX_CLIENTI 1000
```

```
#define MAX_GHISEE 10
```

```
#define MAX_MINUTE 500
```

```
#define MAX_PRIORITATE 3 // 0 - NORMAL, 1 - VIP, 2 - URGENT
```

```
const char* tipuri[] = {"NORMAL", "VIP", "URGENT"};
```

```
typedef struct {
```

```
    int id;
```

```
    int ora_sosire;
```

```
    int timp_servire;
```

```
    int timp_inceput_servire;
```

```
    int timp_finalizare;
```

```
    int tip_prioritate;
```

```
    int servit_de_ghiseu;
```

```
} Client;
```

```
typedef struct {
```

```
    int timp_liber;
```

```
} Ghiseu;
```

```
Client clienti[MAX_CLIENTI];
```

```
Ghiseu ghisee[MAX_GHISEE];
```

```
int nr_clienti = 0;
```

```
int clienti_serviti = 0;
```

```
int timpul_total_asteptare = 0;
```

```
int timpul_maxim_asteptare = 0;
```

```
int clienti_pe_minut[MAX_MINUTE];
```

```
int random_tip_prioritate() {
```

```
    int r = rand() % 100;
```

```
    if (r < 10) return 2;
```

```
    else if (r < 30) return 1;
```

```
    else return 0;
```

```
}
```

```
int random_timp_servire(int tip) {
```

```
    switch (tip) {
```

```

        case 2: return 2 + rand() % 3;

        case 1: return 3 + rand() % 4;

        default: return 4 + rand() % 5;

    }

}

```

```

void genereaza_clienti(int minut_curent, int durata_simulare) {

    int clienti_pe_tura = (minut_curent >= durata_simulare/3 && minut_curent <=
2*durata_simulare/3) ? 3 + rand() % 3 : 1 + rand() % 2;

    clienti_pe_minut[minut_curent] = clienti_pe_tura;

    for (int i = 0; i < clienti_pe_tura && nr_clienti < MAX_CLIENTI; i++) {

        Client c;

        c.id = nr_clienti + 1;

        c.ora_sosire = minut_curent;

        c.tip_prioritate = random_tip_prioritate();

        c.timp_servire = random_timp_servire(c.tip_prioritate);

        c.timp_inceput_servire = -1;

        c.timp_finalizare = -1;

        clienti[nr_clienti++] = c;

    }

}

```

```

int alege_client(int minut, int *servit) {

    int max_prioritate = -1;

    int index = -1;

```

```

for (int i = 0; i < nr_clienti; i++) {

    if (!servit[i] && clienti[i].ora_sosire <= minut) {

        if (clienti[i].tip_prioritate > max_prioritate) {

            max_prioritate = clienti[i].tip_prioritate;

            index = i;

        }

    }

}

return index;

}

```

```

void simuleaza(int durata_simulare, int nr_ghisee) {

    int servit[MAX_CLIENTI] = {0};

    for (int minut = 0; minut < durata_simulare; minut++) {

        genereaza_clienti(minut, durata_simulare);

        for (int g = 0; g < nr_ghisee; g++) {

            if (ghisee[g].timp_liber <= minut) {

                int idx = alege_client(minut, servit);

                if (idx != -1) {

                    clienti[idx].timp_inceput_servire = minut;

                    clienti[idx].timp_finalizare = minut + clienti[idx].timp_servire;

                    clienti[idx].servit_de_ghiseu = g + 1;

                    ghisee[g].timp_liber = clienti[idx].timp_finalizare;

                    servit[idx] = 1;

```

```

        int astept = minut - clienti[idx].ora_sosire;

        timpul_total_asteptare += astept;

        if (astept > timpul_maxim_asteptare)

            timpul_maxim_asteptare = astept;

        clienti_serviti++;

    }

}

}

}

```

```

void afiseaza_statistici(int durata_simulare, int nr_ghisee) {

    FILE *f = fopen("statistici.txt", "w");

    printf("\n=== STATISTICI ===\n");

    fprintf(f, "=== STATISTICI ===\n");

    printf("Durata simulare: %d minute\n", durata_simulare);

    printf("Număr ghișee: %d\n", nr_ghisee);

    printf("Clienți generați: %d\n", nr_clienti);

    printf("Clienți serviți: %d\n", clienti_serviti);

    printf("Timp mediu așteptare: %.2f min\n", clienti_serviti > 0 ?
(float)timpul_total_asteptare / clienti_serviti : 0);

    printf("Timp maxim așteptare: %d min\n", timpul_maxim_asteptare);

```

```

    fprintf(f, "Clienți generați: %d\n", nr_clienti);

    fprintf(f, "Clienți serviți: %d\n", clienti_serviti);

    fprintf(f, "Timp mediu: %.2f min\n", clienti_serviti > 0 ?
(float)timpul_total_asteptare / clienti_serviti : 0);

    fprintf(f, "Timp maxim: %d min\n", timpul_maxim_asteptare);


    int max_min = 0;

    for (int i = 1; i < durata_simulare; i++) {

        if (clienti_pe_minut[i] > clienti_pe_minut[max_min])

            max_min = i;

    }


    printf("Cea mai aglomerată perioadă: minutul %d cu %d clienți\n", max_min,
clienti_pe_minut[max_min]);

    fprintf(f, "Minutul de vârf: %d (%d clienți)\n", max_min,
clienti_pe_minut[max_min]);

    fclose(f);

}


void afiseaza_bar_chart(int durata_simulare) {

    printf("\n=== GRAFIC NUMĂR CLIEȚI / MINUT ===\n");

    for (int i = 0; i < durata_simulare; i++) {

        printf("Minut %2d: ", i);

        for (int j = 0; j < clienti_pe_minut[i]; j++)

            printf("#");

    }

```



```

        printf(" (%d)\n", clienti_pe_minut[i]);

    }

}

int main() {

    srand(time(NULL));

    int nr_ghisee, durata_simulare;

    printf("Număr de ghișee (1-10): ");

    scanf("%d", &nr_ghisee);

    if (nr_ghisee <= 0 || nr_ghisee > MAX_GHISEE) nr_ghisee = 3;

    printf("Durata simulării în minute (max %d): ", MAX_MINUTE);

    scanf("%d", &durata_simulare);

    if (durata_simulare <= 0 || durata_simulare > MAX_MINUTE) durata_simulare =
60;

    simuleaza(durata_simulare, nr_ghisee);

    afiseaza_statistici(durata_simulare, nr_ghisee);

    afiseaza_bar_chart(durata_simulare);

    printf("\nRezultatele au fost salvate în fișierul 'statistici.txt'.\n");

    return 0;

```

}

Poze cu codul complet in Visual Studio Code

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <time.h>
4
5 #define MAX_CLIENTI 1000
6 #define MAX_GHISEE 10
7 #define MAX_MINUTE 500
8 #define MAX_PRIORITATE 3 // 0 - NORMAL, 1 - VIP, 2 - URGENT
9
10 const char* tipuri[] = {"NORMAL", "VIP", "URGENT"};
11
12 typedef struct {
13     int id;
14     int ora_sosire;
15     int timp_servire;
16     int timp_inceput_servire;
17     int timp_finalizare;
18     int tip_prioritate;
19     int servit_de_ghiseu;
20 } Client;
21
22 typedef struct {
23     int timp_liber;
24 } Ghiseu;
25
26 Client clienti[MAX_CLIENTI];
27 Ghiseu ghisee[MAX_GHISEE];
28
29 int nr_clienti = 0;
30 int clienti_serviti = 0;
31 int timpul_total_asteptare = 0;
32 int timpul_maxim_asteptare = 0;
33 int clienti_pe_minut[MAX_MINUTE];
34
35 int random_tip_prioritate() {
36     int r = rand() % 100;
37     if (r < 10) return 2;
38     else if (r < 30) return 1;
39     else return 0;
40 }
41
42 int random_timp_servire(int tip) {
43     switch (tip) {
44         case 2: return 2 + rand() % 3;
45         case 1: return 3 + rand() % 4;
46         default: return 4 + rand() % 5;
47     }
48 }
```

Figura 6.1 – Prima parte de cod

```
50 void genereaza_clienti(int minut_curent, int durata_simulare) {
51     int clienti_pe_tura = (minut_curent >= durata_simulare/3 && minut_curent <= 2*durata_simulare/3) ? 3 + rand() % 3 : 1 + rand() % 2;
52     clienti_pe_minut[minut_curent] = clienti_pe_tura;
53     for (int i = 0; i < clienti_pe_tura && nr_clienti < MAX_CLIENTI; i++) {
54         Client c;
55         c.id = nr_clienti + 1;
56         c.ora_sosire = minut_curent;
57         c.tip_prioritate = random_tip_prioritate();
58         c.timp_servire = random_timp_servire(c.tip_prioritate);
59         c.timp_inceput_servire = -1;
60         c.timp_finalizare = -1;
61         clienti[nr_clienti++] = c;
62     }
63 }
64
65 int alege_client(int minut, int *servit) {
66     int max_prioritate = -1;
67     int index = -1;
68     for (int i = 0; i < nr_clienti; i++) {
69         if (!servit[i] && clienti[i].ora_sosire <= minut) {
70             if (clienti[i].tip_prioritate > max_prioritate) {
71                 max_prioritate = clienti[i].tip_prioritate;
72                 index = i;
73             }
74         }
75     }
76     return index;
77 }
78
79 void simuleaza(int durata_simulare, int nr_ghisee) {
80     int servit[MAX_CLIENTI] = {0};
81     for (int minut = 0; minut < durata_simulare; minut++) {
82         genereaza_clienti(minut, durata_simulare);
83         for (int g = 0; g < nr_ghisee; g++) {
84             if (ghisee[g].timp_liber <= minut) {
85                 int idx = alege_client(minut, servit);
86                 if (idx != -1) {
87                     clienti[idx].timp_inceput_servire = minut;
88                     clienti[idx].timp_finalizare = minut + clienti[idx].timp_servire;
89                     clienti[idx].servit_de_ghiseu = g + 1;
90                     ghisee[g].timp_liber = clienti[idx].timp_finalizare;
91                     servit[idx] = 1;
92                 }
93             }
94         }
95     }
96 }
```

Figura 6.2 – Cod ce include subprogramele

```

142 int main() {
143     srand(time(NULL));
144
145     int nr_ghisee, durata_simulare;
146
147     printf("Număr de ghișee (1-10): ");
148     scanf("%d", &nr_ghisee);
149     if (nr_ghisee <= 0 || nr_ghisee > MAX_GHISEE) nr_ghisee = 3;
150
151     printf("Durata simulării în minute (max %d): ", MAX_MINUTE);
152     scanf("%d", &durata_simulare);
153     if (durata_simulare <= 0 || durata_simulare > MAX_MINUTE) durata_simulare = 60;
154
155     simuleaza(durata_simulare, nr_ghisee);
156     afiseaza_statistici(durata_simulare, nr_ghisee);
157     afiseaza_bar_chart(durata_simulare);
158
159     printf("\nRezultatele au fost salvate în fișierul 'statistici.txt'.\n");
160
161     return 0;
162 }
163

```

Figura 6.3 – Cod Main complet