**Міністерство освіти і науки України**

**Національний університет «Запорізька Політехніка»**

Кафедра програмних засобів

**ЗВІТ**

з лабораторної роботи №3

з дисципліни «Моделювання та Аналіз Програмного Забезпечення» на тему:

«Моделювання системи масового обслуговування зі зворотнім зв'язком. Моделювання багатоканальної системи масового обслуговування»

**Виконав:**

Студент групи КНТ-122 О. А. Онищенко

**Прийняли:**

Викладач: Ж. К. Камінська

2024

Моделювання системи масового обслуговування зі зворотнім зв'язком. Моделювання багатоканальної системи масового обслуговування

Мета роботи

Метою роботи є вивчення методів моделювання СМО зі зворотнім зв'язком та багатоканальних СМО на основі використання SIMC

Результати виконання

Код програми

#include "../simc/simc.h"

#include <iostream>

using namespace std;

void one() {

  auto Modeling\_Hours=40;

  auto Total\_Modeling\_Time=Modeling\_Hours\*60;

  auto Assembly\_Delay=30;

  auto Firing\_Delay=8;

  auto Worker\_Hourly\_Salary=50;

  auto Firing\_Daily\_Price=200;

  auto Material\_Price=2;

  auto Product\_Price=7;

  auto Workers\_Min=4;

  auto Workers\_Max=6;

  auto Best\_Result=0;

  auto Best\_Count=0;

  pqueue Assembly\_Queue;

  pstorage Assembly\_Facility;

  pqueue Firing\_Queue;

  pfacility Firing\_Facility;

  initlist(Total\_Modeling\_Time);

  initcreate(1, 0);

  newqueue(Assembly\_Queue, "\"Assembly Queue\"");

  newqueue(Firing\_Queue, "\"Firing Queue\"");

  newfac(Firing\_Facility, "\"Firing Facility\"");

  for (auto Workers\_Count=Workers\_Min;Workers\_Count<=Workers\_Max;Workers\_Count++) {

    for (auto j=Workers\_Min;j<=Workers\_Count;j++) initcreate(1,0);

    newstorage(Assembly\_Facility, "\"Assembly Facility\"", 3);

    auto Parts\_Assembled=0;

    while (systime<Total\_Modeling\_Time) {

      plan();

      switch (sysevent) {

        case 1: inqueue(Assembly\_Queue); break;

        case 2: enter(Assembly\_Facility, 1); break;

        case 3: outqueue(Assembly\_Queue); break;

        case 4: delayt(Assembly\_Delay); break;

        case 5: leave(Assembly\_Facility, 1); break;

        case 6: inqueue(Firing\_Queue); break;

        case 7: seize(Firing\_Facility); break;

        case 8: outqueue(Firing\_Queue); break;

        case 9: delayt(Firing\_Delay); break;

        case 10: outfac(Firing\_Facility); Parts\_Assembled+=1; break;

        case 11: next(1); break;

      }

    }

    auto Workers\_Salary=Worker\_Hourly\_Salary\*Modeling\_Hours\*Workers\_Count;

    auto Firing\_Facility\_Cost=Firing\_Daily\_Price/8\*Modeling\_Hours;

    auto Materials\_Cost=Parts\_Assembled\*Material\_Price;

    auto Parts\_Cost=Parts\_Assembled\*Product\_Price;

    auto Total\_Expenses=Workers\_Salary+Firing\_Facility\_Cost+Materials\_Cost;

    auto Profit=Parts\_Cost-Total\_Expenses;

    if (abs(Profit)>abs(Best\_Result)) { Best\_Result=Profit; Best\_Count=Workers\_Count; }

    cout << "Workers: " << Workers\_Count << " Profit: " << Profit << endl;

  }

  cout << "\nBest Count: " << Best\_Count << " Best Result: " << Best\_Result << endl << endl;

  printall();

}

void two() {

  auto Interval=115;

  auto First\_Delay=335;

  auto Second\_Delay=110;

  auto Modeling\_Hours=1;

  auto Total\_Modeling\_Time=Modeling\_Hours\*60\*60;

  pqueue First\_Queue;

  pqueue Second\_Queue;

  pfacility First\_Facility;

  pfacility Second\_Facility;

  initlist(Total\_Modeling\_Time);

  initcreate(1, 0);

  newqueue(First\_Queue, "\"First Queue\"");

  newqueue(Second\_Queue, "\"Second Queue\"");

  newfac(First\_Facility, "\"First Facility\"");

  newfac(Second\_Facility, "\"Second Facility\"");

  while (systime<Total\_Modeling\_Time) {

    plan();

    switch (sysevent) {

      case 1: create(Interval); break;

      case 2: cout << "First Queue Length: " << First\_Queue->lq << endl; if (Second\_Queue->status == queue::empty) next(8); else inqueue(First\_Queue); break;

      case 3: seize(First\_Facility); break;

      case 4: outqueue(First\_Queue); break;

      case 5: delayt(First\_Delay); break;

      case 6: outfac(First\_Facility); break;

      case 7: destroy(); break;

      case 8: create(Interval); break;

      case 9: cout << "Second Queue Length: " << Second\_Queue->lq << endl; Second\_Queue->mq=1; inqueue(Second\_Queue); break;

      case 10: seize(Second\_Facility); break;

      case 11: outqueue(Second\_Queue); break;

      case 12: delayt(Second\_Delay); break;

      case 13: outfac(Second\_Facility); break;

      case 14: destroy(); break;

    }

  }

  cout << "\nFirst Queue Max Length: " << First\_Queue->mq << " Average Length: " << First\_Queue->lm << endl << endl;

  printall();

}

void three() {

  auto First\_Interval=1;

  auto First\_Delay=1;

  auto Second\_Interval=4;

  auto Second\_Delay=5;

  auto Total\_Modeling\_Time=300;

  // first t can be served only if second facility is seized

  pfacility First\_Facility;

  pfacility Second\_Facility;

  initlist(Total\_Modeling\_Time);

  initcreate(1, 0);

  newfac(First\_Facility, "\"First Facility\"");

  newfac(Second\_Facility, "\"Second Facility\"");

  while (systime<Total\_Modeling\_Time) {

    plan();

    switch (sysevent) {

      case 1: create(First\_Interval); break;

      case 2: if (Second\_Facility->status == facility::seized) seize(First\_Facility); else next(7); break;

      case 3: delayt(First\_Delay); break;

      case 4: outfac(First\_Facility); break;

      case 6: destroy(); break;

      case 7: create(Second\_Interval); break;

      case 8: seize(Second\_Facility); break;

      case 9: delayt(Second\_Delay); break;

      case 10: outfac(Second\_Facility); break;

      case 11: destroy(); break;

    }

  }

  cout << "Modeling finished, praise Jesus Christ our Holy Lord GOD Almighty" << endl << endl;

  printall();

}

int main()

{

  one();

  two();

  three();

  return 0;

}

Виконання програми

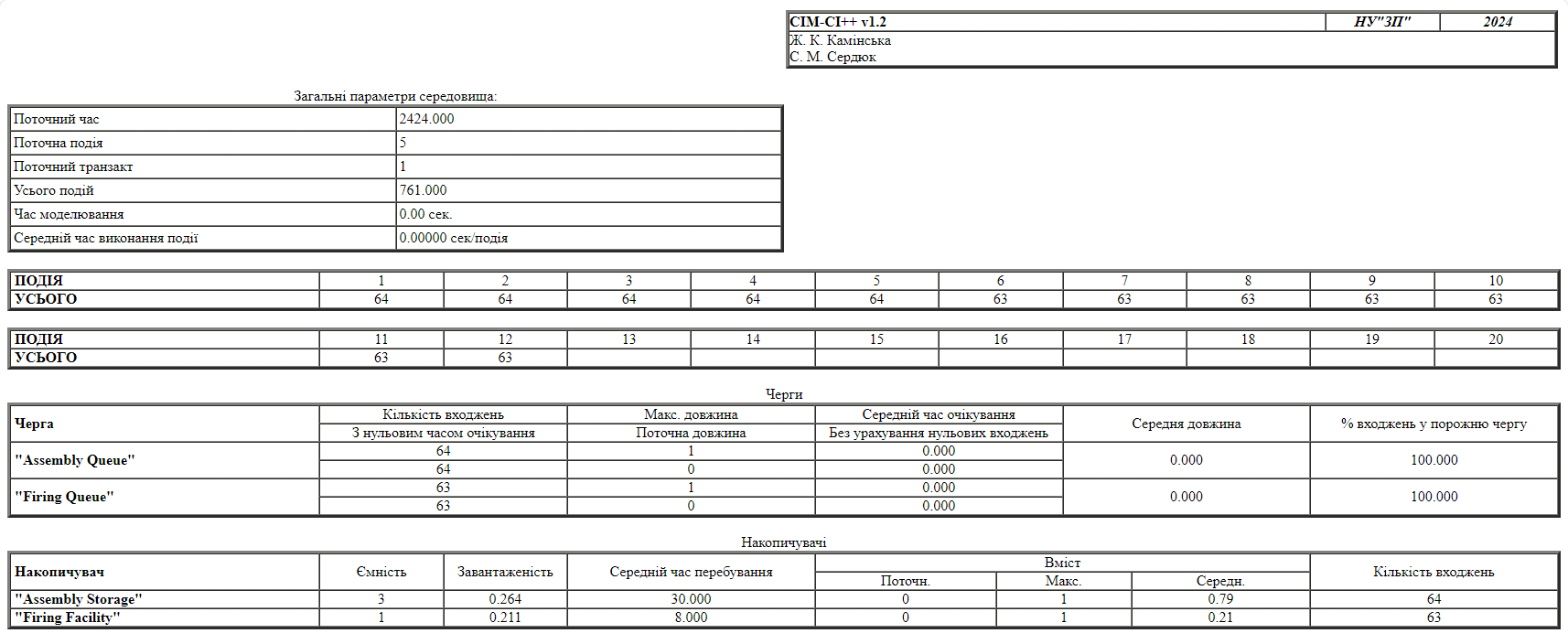


Рисунок 1.1 – Загальне завдання – браузер

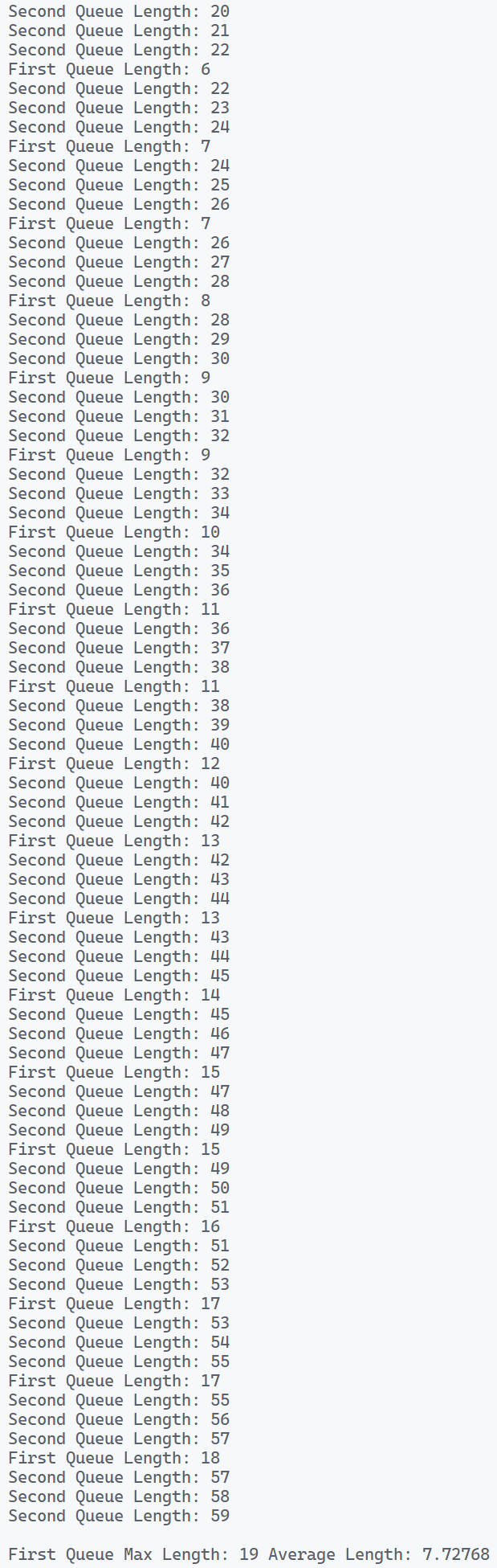
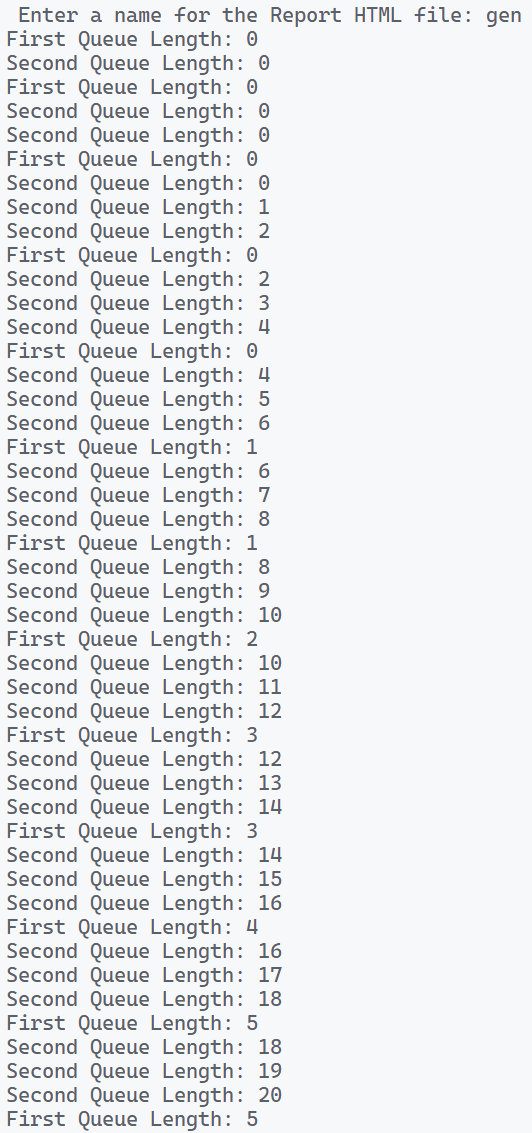
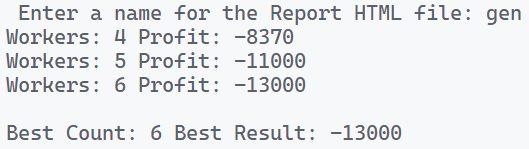




Рисунок 1.2 – Загальне завдання – консолька ♰

Висновки

Таким чином, ми вивчили методи моделювання СМО зі зворотнім зв'язком, а також багатоканальні СМО на основі використання SIMC

Контрольні питання

Багатоканальні прилади. Множинні типи даних "НАКОПИЧУВАЧ" (багатоканальний прилад)

Накопичувач то є багатоканальний прилад де сам накопичувач може бути звільнений не тим транзактом, яким був зайнятий. То є динамічний об'єкт для моделювання декілької пристроїв.

Процедури створення – знищення накопичувача

Створення накопичувачів виконується функцією newstorage, а знищення виконується функцією destrs.

Блокування транзактів

Блокування транзактів здійснюється функцією void wait(event e) де e то є номер очікуваної події.