НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ  
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ  
імені ІГОРЯ СІКОРСЬКОГО»  
Факультет інформатики та обчислювальної техніки

Звіт з дисципліни  
**«Моделювання систем»**

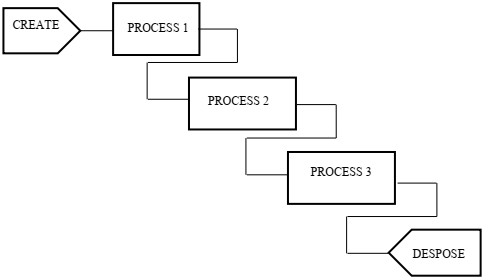
|  |  |
| --- | --- |
| Виконав:  Студент групи ЗПІ-зп01  Кононов М. А. | Перевірила  д.т.н., проф. кафедри ІПІ  Стеценко І. В. |

Київ 2023

**КОМП’ЮТЕРНИЙ ПРАКТИКУМ 2**

Завдання до практичної роботи:

1. Реалізувати алгоритм імітації простої моделі обслуговування одним пристроєм з використанням об’єктно-орієнтованого підходу.
2. Модифікувати алгоритм, додавши обчислення середнього завантаження пристрою.
3. Створити модель за схемою, представленою на рисунку:



1. Виконати верифікацію моделі, змінюючи значення вхідних змінних та параметрів моделі. Навести результати верифікації у таблиці.

5. Модифікувати клас PROCESS, щоб можна було його використовувати для моделювання процесу обслуговування кількома ідентичними пристроями.

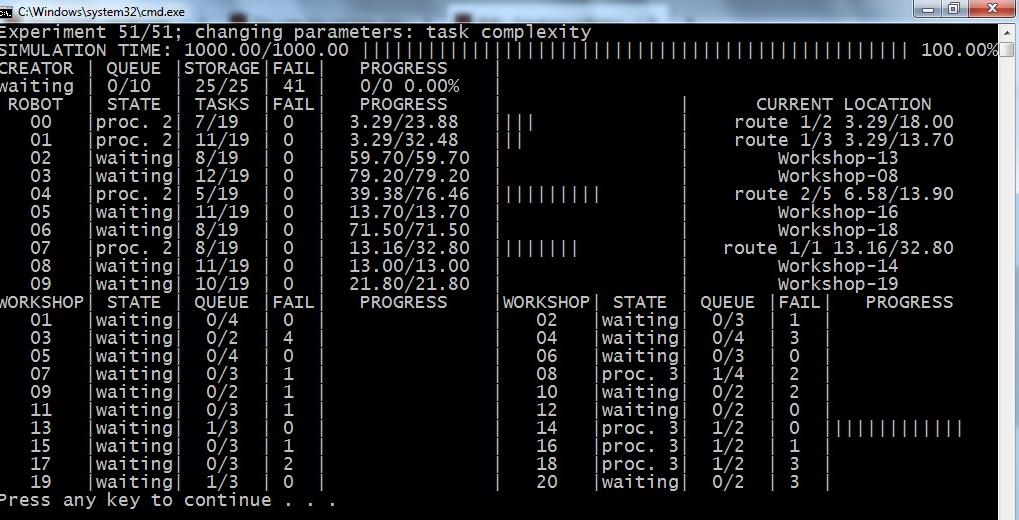
6. Модифікувати клас PROCESS, щоб можна було організовувати вихід в два і більше наступних блоків, в тому числі з поверненням у попередні блоки.

**ОПИС РОЗРОБЛЕНИХ РІШЕНЬ**

Щоб вирішити всі вищенаведені завдання нами був розроблений консольний додаток на мові C++ з використанням об’єктно-орієнтованих засобів. Він представляє собою середовище для імітації мережі масового обслуговування, учасниками якої є creator, robot та workshop. Усі вони належать класу device, успадковуючи спільні властивості. Програма ілюструє роботу системи масового обслуговування, виводячи на екран стан кожного з «акторів» та перелік подій, які відбуваються в певний період модельного часу, що розраховується за принципом найближчої події. Головна ідея розробленого нами рішення полягає в тому, що creator створює завдання та передає їх об’єкту класу robot, який переміщується між виробничими приміщеннями (workshop), де ці завдання виконуються. Маршрути руху задані графом у вигляді матриці суміжності (зчитується з файлу), відстані та порядок слідування обчислюються за допомогою алгоритму Дейкстри. Усі «дійові особи», окрім creator, мають параметр «надійність», згідно з яким вони переходять в поламаний стан та відновлюють роботу через проміжки часу відповідного розміру. Максимальна кількість «аварій» також встановлюється, «актори» не приймають та не продовжують виконання завдань до «ремонтування». Таким чином об’єктом класу robot досягається необхідність переходу в попередні блоки.

Усі параметри роботи моделі задаються в програмному коді, зміни можна побачити лише після успішної компіляції. Для зручності роботи код було поділено на cpp-файли, назви методів та змінних знаходяться у header-файлах. *main.cpp* містить ініціалізуючі інструкції та запускає тести. *model.cpp* описує реалізацію моделі, а також зберігає результати у вигляді csv-файлів. *device.cpp* містить команди, які спільно виконуються об’єктами класів creator, robot та workshop (їх методи написані у відповідних файлах). *SharedFunctions.cpp* зберігає допоміжні підпрограми, які потрібні в будь-якій частині проекту.

Для коректного відображення даних у системних налаштуваннях консолі має бути вказана ширина 91 символ:

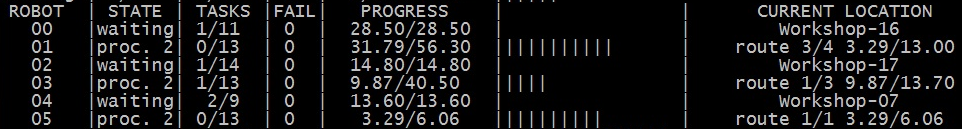


**Creator** генерує завдання у вигляді пари значень «номер цеху – тривалість виконання». Він має «відділи» для роботів, сховище для «макетів», а також перелік робочих приміщень. У першу чергу до роботів завантажуються раніше створені завдання, які вже знаходяться «на складі», лише у другу чергу створюються нові. Разом із новими завданнями об’єкти класу robot отримують значення інтервалу, після якого вони мають розпочати свої дії. Зрозуміло, що нові задачі потрапляють у сховище обмеженого розміру лише у випадку відсутності наявних роботів. При спробі додати завдання у переповнене сховище виникає помилка та значення змінної *CountOfFailures* збільшується. Процес генерації завдань запускається у заданий проміжок часу, який розраховується методом *GetDelay*. В залежності від заданих параметрів, він повертає момент часу наступної події за одним із трьох законів розподілу (рівномірного, нормального або експоненціального). Якщо закон розподілу заданий «некоректним» символом, інтервал наступної події встановлюється значенням вхідної змінної*.* Інші «актори» також використовують цей метод для обчислення моменту часу початку подальших дій. Робота creator візуалізується відповідним індикатором виконання (progress bar) у вигляді символів вертикальної риски «|» і має наступний вигляд:



Розмір часу, необхідний для генерації одного завдання, є випадковою величиною, яка знаходиться в заданому діапазоні. Складність завдання (максимальна кількість workshop, які має відвідати robot) також регулюється відповідними змінними. Вона не може бути меншою за 2, оскільки останнім пунктом завжди має бути сам creator (robot повертається до нього після виконання всіх завдань). Зрозуміло, що максимальне значення не може перевищувати кількість робочих приміщень. Отже, creator виконує перший етап «CREATE», зазначений на схемі, та вмикає подальші процеси.

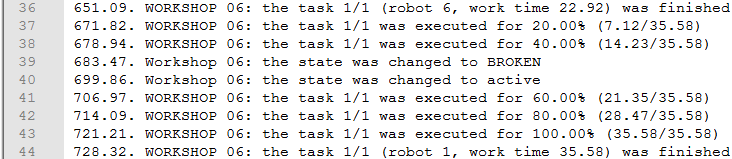
Після отримання завдання у вигляді масиву структур «номер цеху – тривалість роботи» **robot** розпочинає «PROCESS 1». За нашим задумом, на реальному виробництві він є аналогом завантаженням каркасу майбутнього виробу на конвеєр, який переміщує деталі, поступово виконуючи їх збірку. У програмному коді цей процес реалізований шляхом відліку кількості цехів, які потрібно відвідати, із заданим інтервалом (змінна *TaskTime*). По його завершенню після певного проміжку часу, який може бути заданий за одним із законів розподілу, robot починає «PROCESS 2», який полягає у переміщенні макету до вказаного workshop. Цей процес включає в себе обчислення найкоротшої відстані, а також детальний вивід маршруту руху. Робота цього «актора» візуалізується належним чином:



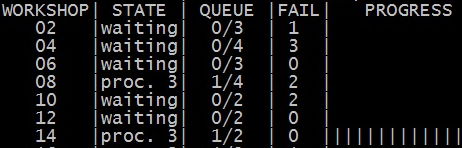
Алгоритм robot також передбачає коректну поведінку при неможливості потрапити у цех (через відсутність вільного місця в черзі або поламаний стан workshop): він рухається до наступного виробничого приміщення, порушивши заданий технологічний процес. У цьому випадку після першого обходу здійснюється перехід у попередні блоки, тобто robot намагається повторити маршрут, відвідавши не досягнуті workshop. Якщо це йому не вдається 3 рази, кількість аварій (*CountOfFailures*) збільшується, та цех видаляється з черги завдань. У випадку наявності лише одного «проблемного» приміщення роботом здійснюється циклічний рух між creator та цим workshop, до тих пір, поки він або не потрапить до нього або не видалиться.

У випадку успішного прийняття цехом, robot переходить у режим очікування сигналу про завершення обробки макету (час наступної події встановлюється максимальним). Після відвідування всіх виробничих приміщень або видалення недосяжних robot повертається у creator за новими завданнями, виконуючи процес «DISPOSE». Завершивши його, він не призначає час запуску creator, а просто очікує в черзі на нові завдання, які мають завантажитись зі сховища або згенеруватись згідно з розкладом. Кількість роботів також задається в програмному коді.

«PROCESS 3» виконується об’єктом класу **workshop**. У нашому випадку він полягає у відліку часу, отриманого від robot. Щоб уникнути надмірно велику кількість однотипних подій, ми ділимо цей час на 5 рівних частин, тобто інтерфейс користувача сповіщає про прогрес лише через інтервал *StepSize=WorkTime/5.0*. У випадку виникнення аварії робота продовжується лише після ремонтування, що можна помітити у відповідному log-файлі:

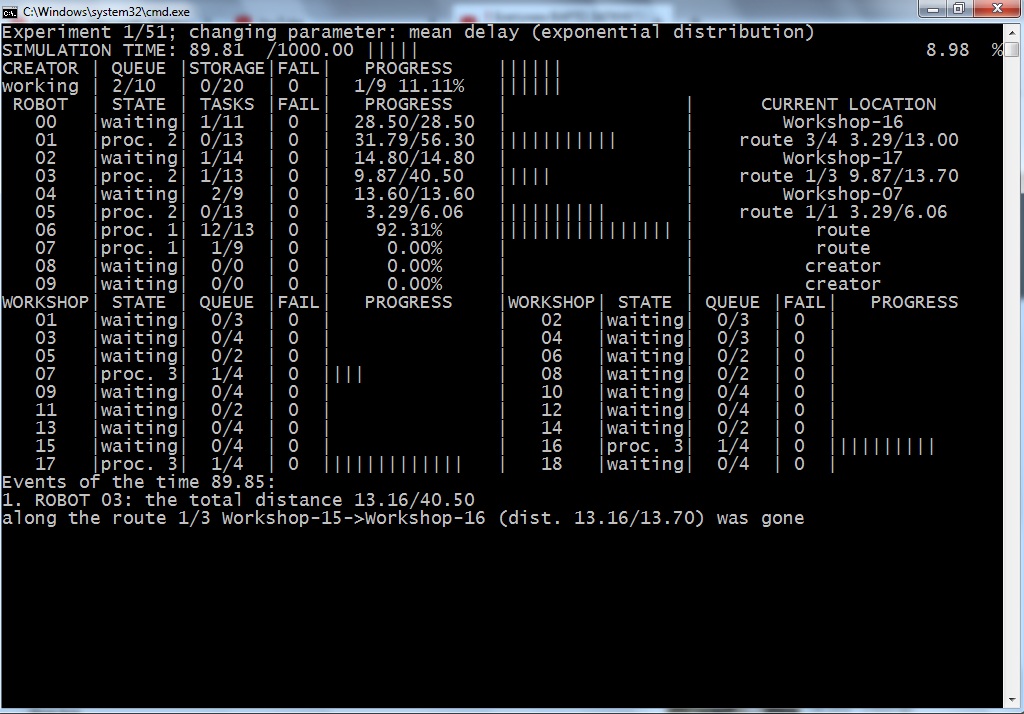


Тут ми бачимо, що у час «678.94» було виконано 40% роботи, але у «683.47» workshop 06 зламався, робота відновилась лише в «699.86», після цього було заново відраховано наступний проміжок часу. З частини цього log-файлу ми також бачимо, що на повне виконання одного завдання workshop витрачає 120% модельного часу, оскільки останні 20% йому необхідні для переходу в попередній стан. Цей процес також включає в себе оновлення індикаторів, які мають наступний вигляд:

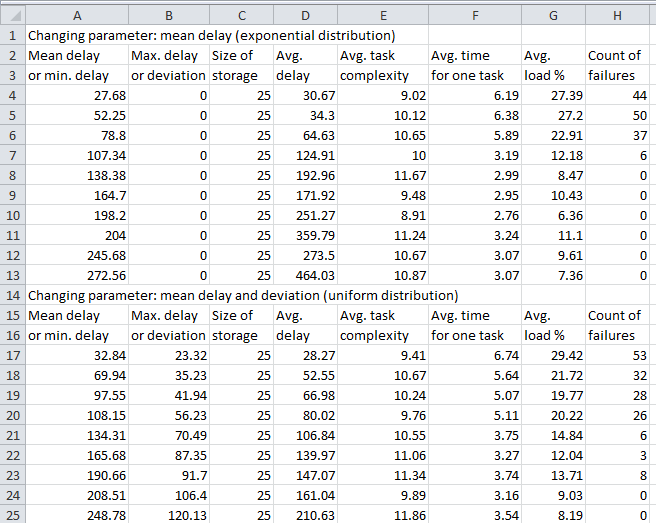


Значення стовпця «FAIL» збільшується, якщо robot не зміг потрапити до нього через відсутність місця в черзі або через неробочий стан. Після обслуговування всіх robot він переходить в режим очікування, встановивши час наступної події рівним часу симуляції (змінна *SimulationTime*).

Розроблений нами додаток може працювати і в режимі покрокового управління, тобто користувач може бачити перелік подій, які виконуються в певний період часу, а після натискання клавіші «Enter» здійснюється оновлення всіх індикаторів. Для того щоб перейти в цей режим, необхідно активізувати рядки коду, скриті символами коментаря у файлі *model.cpp*:



Як було помічено вище, перелік усіх подій зберігається у відповідних файлах, тобто кожен об’єкт класу *device* «веде» власний «щоденник», крім них є і *AllEvents.txt*, де перелічені абсолютно всі події, які відбулись у системі. Окрім log-файлів система створює і csv-файли, наповнені статистичними даними. Кожен з «акторів» має свої власні стовпці таблиць у відповідному файлі. Так для creator у вікні Excel таблиця виглядає таким чином:



У ній ми «реєструємо» середню або мінімальну затримку, максимальну затримку або відхилення (необхідна лише для генерації значень за рівномірним або нормальним законом розподілу), максимальний розмір сховища. У процесі роботи обчислюються середня затримка, середня складність створених завдань, середній час на генерацію одного завдання, середнє навантаження (процент активного часу), кількість «аварій» через відсутність місця для нових макетів. Таблиці будуються відповідно до характеру експериментів один з яких містить в собі 10 або 11 «ітерацій», протягом яких фіксуються зміни стандартно заданих значень і отримані результати. Для creator перелік стандартних змінних наведений у наступній таблиці:

|  |  |
| --- | --- |
| Creator\_DelayDistribution; | символ назви закону розподілу часу |
| Creator\_MeanDelayOrMinDelay | середня або мінімальна затримка часу |
| Creator\_MaxDelayOrDeviation | максимальна затримка або відхилення |
| Creator\_MinTaskTime | мінімальний час на генерацію одного завдання |
| Creator\_MaxTaskTime | максимальний час на генерацію одного завдання |
| Creator\_MaxQueue | максимальний розмір черги завдань (сховище) |
| MinTaskComplexity | мінімальна складність завдання |
| MaxTaskComplexity | максимальна складність завдання |

Результати роботи robot залежать від значень наступних параметрів:

|  |  |
| --- | --- |
| Robot\_DelayDistribution | символ назви закону розподілу часу |
| Robot\_MeanDelayOrMinDelay | середня або мінімальна затримка часу |
| Robot\_MaxDelayOrDeviation | максимальна затримка або відхилення |
| Robot\_MinMovementTime | мінімальний час на «подолання» певної відстані |
| Robot\_MaxMovementTime | максимальний час на «подолання» певної відстані |
| Robot\_MinTaskTime | мінімальний час на додавання одного завдання |
| Robot\_MaxTaskTime | максимальний час на додавання одного завдання |
| Robot\_MinReliability | мінімальна надійність пристрою (не більше 1.0) |
| Robot\_MaxReliability | максимальна надійність пристрою (не більше 1.0) |
| Robot\_MinCountOfBreakdowns | мінімальна кількість «аварій» |
| Robot\_MaxCountOfBreakdowns | максимальна кількість «аварій» |

Таблиця змінних workshop:

|  |  |
| --- | --- |
| Workshop\_MaxCountOfIdDigits | максимальна кількість цифр у номері (необхідно для естетичного вигляду) |
| Workshop\_MinSizeOfQueueOfRobots | мінімальна кількість robot у черзі |
| Workshop\_MaxSizeOfQueueOfRobots | максимальна кількість robot у черзі |
| Workshop\_DelayDistribution | символ назви закону розподілу часу |
| Workshop\_MeanDelayOrMinDelay | середня або мінімальна затримка часу |
| Workshop\_MaxDelayOrDeviation | максимальна затримка або відхилення |
| Workshop\_MinReliability | мінімальна надійність пристрою (не більше 1.0) |
| Workshop\_MaxReliability | максимальна надійність пристрою (не більше 1.0) |
| Workshop\_MinCountOfBreakdowns | мінімальна кількість «аварій» |
| Workshop\_MaxCountOfBreakdowns | максимальна кількість «аварій» |

При створенні об’єктів вищезазначених класів всі ці параметри отримують випадкові значення у вказаних діапазонах, при необхідності їх можна зробити рівними, забезпечивши максимальну ідентичність пристроїв.

У рамках повного тесту нашої системи ми провели 5 експериментів, отримавши 5 таблиць з результатами роботи кожного «учасника» мережі. Перший з них присвячений відслідковуванню впливу середньої затримки, яка обчислюється за експоненціальним законом розподілу. Для creator графік має наступний вигляд:

Ми бачимо, що збільшення заданої користувачем затримки призводить до збільшення часу реальної затримки, але з певними перепадами, середнє навантаження (процент активного часу) при цьому зменшується. Ми робимо висновок, що creator коректно реагує на зміну вказаних параметрів, відхилення від норми можна пояснити постійним використанням випадкових значень.

Побудуємо подібний графік для robot:

Тут спостерігається подібна тенденція, що може свідчити про правильність реалізованих алгоритмів. Графік для workshop:

Тут, окрім вищезазначених параметрів ми проаналізували процент виконаних завдань. Було помічено, що він не завжди залежить від заданого параметру затримки, оскільки на нього впливають кілька факторів, включаючи надійність та швидкість robot.

Подібні експерименти ми провели, розрахувавши затримки за іншими вищезазначеними законами розподілів, отримавши ідентичні результати. При фіксованих значеннях графіки мають схожий вигляд:

|  |  |
| --- | --- |
| Creator: | Robot |
|  |  |
| Workshop | |
|  | |

Також нами було проаналізовано вплив однакової складності завдань на продуктивність та активність компонентів системи. Графік для creator:

Ми бачимо, що збільшення даного параметру не завжди відповідним чином впливає на завантаженість пристрою, однак загальна тенденція має зростаючий характер. Графік для robot:

Тут видно, що разом зі зростаючою складністю завдання у більшості випадків зростає і розмір активного часу, а процент виконаних не завжди залежить від значення змінюваного параметра, хоча загальна тенденція є низхідною. Для workshop графік виглядає наступним чином:

На ньому видно, що середнє завантаження та процент виконаних завдань не залежить від їх складності. Тобто workshop у найменшій мірі підвладний впливу creator, що не можна стверджувати про robot.

Також нами було проаналізовано середній та максимальний розміри черги для workshop. Було встановлено, що при заданих параметрах системи черга майже не використовуються, після виконання всіх експериментів значення цього параметру не перевищувало одиницю.

Проаналізувавши отримані результати ми дійшли висновку, що зміна вказаних параметрів коректно впливає на інші показники системи. Проте розроблена нами програма не позбавлена недоліків і мало помітних «багів». Наприклад, «актори» переходять у режим «аварії» навіть якщо знаходяться в стані спокою: у реальному світі таке дуже рідко трапляється. Тільки в деяких виключних випадках, зазвичай, під впливом зовнішнього середовища, пристрій може зламатись, знаходячись у неактивному стані. Для усунення цього недоліку нам слід змінювати стан пристрою лише в активний час, а не за весь період моделювання. Тільки в цьому випадку ми зможемо визначити точний вплив надійності пристрою на його продуктивність. Під час проведення експериментів ми дуже рідко відслідковували перехід robot у попередні блоки, оскільки «колізії» рідко траплялись при наведених параметрах. Тому для доказів того, що дана функція коректно відпрацьовує в необхідних випадках, ми залишили рядку коду для її детального тестування. Їх можна активізувати видаливши символи коментаря у файлі *main.cpp* та скривши або видаливши код для виконання тестів. На даному етапі розвитку цього проекту ми вважаємо, що наша програма повністю виконує поставлені задачі.