

**Національний Технічний Університет України КПІ**

Факультет Інформатики та Обчислювальної Техніки  
Кафедра інформатики та програмної інженерії

**Практична робота №2**

З дисципліни «Моделювання систем»

ОБ’ЄКТНО-ОРІЄНТОВАНИЙ ПІДХІД ДО ПОБУДОВИ ІМІТАЦІЙНИХ

МОДЕЛЕЙ ДИСКРЕТНО-ПОДІЙНИХ СИСТЕМ

**Перевірила:**

Асистент

Бернатович Анатолій Олександрович

Оцінка:\_\_\_\_\_\_\_\_\_\_\_**Виконав:**

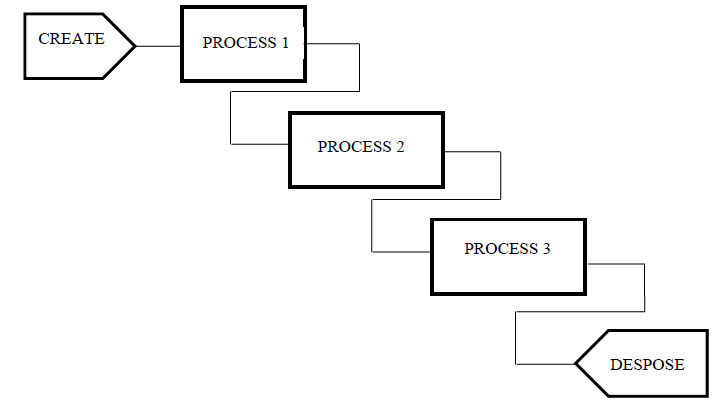
Студент групи ІТ-92

Бондаренко Д.С.

Дата: \_\_\_\_\_\_\_\_\_\_\_\_\_

Завдання до практичної роботи

1. Реалізувати алгоритм імітації простої моделі обслуговування одним пристроєм з використанням об’єктно-орієнтованого підходу. **5 балів.**
2. Модифікувати алгоритм, додавши обчислення середнього завантаження пристрою. **5 балів.**
3. Створити модель за схемою, представленою на рисунку нище. **30 балів.**

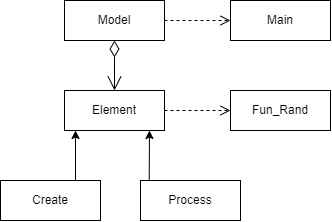


1. Виконати верифікацію моделі, змінюючи значення вхідних змінних та параметрів моделі. Навести результати верифікації у таблиці. **10 балів.**
2. Модифікувати клас PROCESS, щоб можна було його використовувати для моделювання процесу обслуговування кількома ідентичними пристроями. **20 балів.**
3. Модифікувати клас PROCESS, щоб можна було організовувати вихід в два і більше наступних блоків, в тому числі з поверненням у попередні блоки. **30 балів.**

Теоретичні відомості

Для даної лабораторної роботи була обрана універсальна мова програмування Python. Даний вибір був здійснений через те, що дана мова надає безліч різноманітних бібліотек для роботи із графіками, гістограмами та математичними функціями, що робить її дуже зручною для виконання завдань лабораторної роботи.

Приблизну структуру об’єктно-орієнтованої програми представимо діаграмою класів:



Модель складається з елементів, які є нащадками одного універсального типу Element. Цей клас містить основні поля та методи елементу моделі такі, як tcurr (поточний момент часу), tnext (момент часу наступної події), delayMean (середеє значення часової затримки), delayDev(середнє квадратичне відхилення часової затримки), getDelay() (розрахунок часової затримками), inAct() (вхід в елемент), outAct() (вихід з елементу). Використання універсального класу надає можливість уніфікувати використання різних елементів в імітаційній програмі.

Важливим для з’єднання елементів в єдину модель є поле nextElement, що вказує на наступний (в маршруті слідування вимоги) елемент моделі.

Клас Model містить метод simulate(double time), що здійснює імітацію на інтервалі часу time. Імітація здійснюється за відомим з попередньої теми принципом: визначається момент найближчої події, просувається час в момент найближчої події та здійснюється відповідна подія. Щоб зменшити обсяг обчислень, введемо також здійснення відповідної події для всіх елементів, час наступної події яких співпадає з поточним моментом часу. Оскільки ми не розглядаємо можливість присвоєння пріоритету елементам моделі, то ця дія цілком допустима.

Виконання лабораторної роботи

Побудуємо алгоритм імітації простої моделі обслуговування одним пристроєм з використанням об’єктно-орієнтованого підходу, використовуючи приклад, який було наведено у лекції, та розрахуємо середнє завантаження пристрою. Наш алгоритм буде виглядати наступним чином:

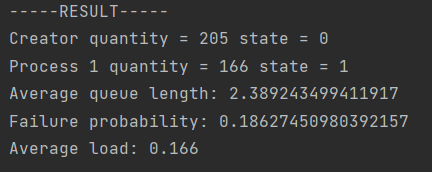


Емуляцію будемо виконувати на проміжку часу рівним у 1000 одиниць. Затримка (delay) для CREATE та PROCESS буде рівним 5. Для розрахунку коли відбудиться наступна подія ми будемо використовувати експоненційний розподіл.

Не вдаючись у подробиці внутрішньої реалізації коду, (детальний опис коду можна знайти у мене на сторінці у GitHub <https://github.com/Agupnik/system_modeling_kpi_fict>) побудова даної моделі буде виглядати так:

c = Create(5)  
  
p1 = Process(5)  
  
p1.max\_queue = 5  
  
c.distribution = 'exp'  
p1.distribution = 'exp'  
  
c.name = 'Creator'  
p1.name = 'Process 1'  
  
c.next\_element = p1  
p1.is\_next\_dispose = True  
  
elements = [c, p1]  
model = Model(elements)  
res = model.simulate(1000)

Виконаємо програму та отримаємо наступні результати:



Де “Creator quantity” – кількість опрацьованих елементів процесом CREATE; “Process 1 quantity” – кількість опрацьованих елементів процесом PROCESS; “Average queue length” – середня довжина черги для PROCESS; “Failure probability” – ймовірність не опрацювання завдання для PROCESS; “Average load” – середнє завантаження PROCESS.

Побудуємо модель за схемою наведеною у завданні до практичної роботи. Для цього нам потрібно модифікувати клас PROCESS, щоб можна було його використовувати для моделювання процесу обслуговування кількома ідентичними пристроями. Разом із цим ми також модифікуємо клас PROCESS, щоб можна було організовувати вихід в два і більше наступних блоків, в тому числі з поверненням у попередні блоки. Ці дві дії було вирішено об’єднати в одну, оскільки у нас критично зміниться метод out\_act() класу Process.

Для виконання наступних дій додаймо до властивостей масив next\_processes а також змінимо метод out\_act() наступним чином:

def out\_act(self):  
 super().out\_act()  
 self.set\_t\_next(float('inf'))  
 self.set\_state(0)  
  
 if self.queue > 0:  
 self.queue -= 1  
 self.set\_state(1)  
 self.t\_next = self.t\_curr + self.get\_delay()  
  
 if len(self.next\_processes) > 0:  
 if not self.is\_next\_dispose:  
 index = rnd.randint(0, len(self.next\_processes) - 1)  
 else:  
 index = rnd.randint(0, len(self.next\_processes))  
 if index != len(self.next\_processes):  
 self.next\_processes[index].in\_act()

Тепер у нас якщо масив не пустий, то вбудованими у Python генератором випадкових чисел ми будемо обирати наступну дію і на цю наступну дію ми будемо виконувати метод in\_act(). При цьому, якщо наступна дія DISPOSE, то ми також будемо обирати наступну дію враховуючи це.

Побудуємо модель разом із верифікацією моделі. Модель для всіх її процесів буде використовувати експоненційний розподіл. Для більш детального опису коду варто подивитися на GitHub: <https://github.com/Agupnik/system_modeling_kpi_fict>:

n\_param = 20  
  
delay\_create\_list = [4, 10, 4, 4, 4, 4, 4, 4, 4, 1, 4, 4, 4, 0.5, 4, 4, 4, 4, 4, 4]  
delay\_process1\_list = [4, 4, 10, 4, 4, 4, 4, 4, 4, 4, 1, 4, 4, 4, 0.5, 4, 4, 4, 4, 4]  
delay\_process2\_list = [4, 4, 4, 10, 4, 4, 4, 4, 4, 4, 4, 1, 4, 4, 4, 0.5, 4, 4, 4, 4]  
delay\_process3\_list = [4, 4, 4, 4, 10, 4, 4, 4, 4, 4, 4, 4, 1, 4, 4, 4, 0.5, 4, 4, 4]  
maxQ\_list1 = [5, 5, 5, 5, 5, 10, 5, 5, 1, 1, 1, 1, 1, 5, 5, 5, 5, 1, 5, 5]  
maxQ\_list2 = [5, 5, 5, 5, 5, 5, 10, 5, 1, 1, 1, 1, 1, 5, 5, 5, 5, 5, 1, 5]  
maxQ\_list3 = [5, 5, 5, 5, 5, 5, 5, 10, 1, 1, 1, 1, 1, 5, 5, 5, 5, 5, 5, 1]  
distribution = ['exp', 'exp', 'exp', 'exp', 'exp', 'exp', 'exp', 'exp', 'exp', 'exp', 'exp', 'exp', 'exp', 'exp',  
 'exp', 'exp', 'exp', 'exp', 'exp', 'exp', 'exp']  
  
df = pd.DataFrame()  
rows = []  
  
for i in range(n\_param):  
 c = Create(delay\_create\_list[i])  
  
 p1 = Process(delay\_process1\_list[i])  
 p2 = Process(delay\_process2\_list[i])  
 p3 = Process(delay\_process3\_list[i])  
  
 p1.max\_queue = maxQ\_list1[i]  
 p2.max\_queue = maxQ\_list2[i]  
 p3.max\_queue = maxQ\_list3[i]  
  
 c.distribution = distribution[i]  
 p1.distribution = distribution[i]  
 p2.distribution = distribution[i]  
 p3.distribution = distribution[i]  
  
 c.name = 'Creator'  
 p1.name = 'Process 1'  
 p2.name = 'Process 2'  
 p3.name = 'Process 3'  
  
 c.next\_element = p1  
 p1.next\_processes.append(p2)  
 p2.next\_processes.append(p3)  
 p3.is\_next\_dispose = True  
  
 elements = [c, p1, p2, p3]  
 model = Model(elements)  
 res = model.simulate(1000)  
  
 param = {'delay\_create': delay\_create\_list[i],  
 'delay\_process1': delay\_process1\_list[i],  
 'delay\_process2': delay\_process2\_list[i],  
 'delay\_process3': delay\_process3\_list[i],  
 'max\_queue1': maxQ\_list1[i],  
 'max\_queue2': maxQ\_list2[i],  
 'max\_queue3': maxQ\_list3[i],  
 'process1\_processed': p1.quantity,  
 'process1\_failed': p1.failure,  
 'process2\_processed': p2.quantity,  
 'process2\_failed': p2.failure,  
 'process3\_processed': p3.quantity,  
 'process3\_failed': p3.failure,  
 'distribution': distribution[i]}  
  
 rows.append({\*\*param, \*\*res})  
  
# назва файлу xlsx  
file\_name = 'ModelData.xlsx'  
  
# імпорт в Excel  
df = df.append(rows)  
df.to\_excel(file\_name)  
print(tabulate(df, headers='keys', tablefmt='fancy\_grid', numalign="center"))

Отримаємо наступні результати, які ми заповнимо до таблиці:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Номер прогону | Затримка CREATE | Затримка PROCESS 1 | Затримка PROCESS 2 | Затримка PROCESS 3 | Черга PROCESS 1 | Черга PROCESS 2 | Черга PROCESS 3 | Опрацьовано PROCESS 1 | Не опрацьовано PROCESS 1 | Опрацьовано PROCESS 2 | Не опрацьовано PROCESS 2 | Опрацьовано PROCESS 3 | Не опрацьовано PROCESS 3 | Максимальна черга на модель | Середня черга на модель | Ймовірність невдачі на модель | Середня завантаженість моделі |
| 1 | 4 | 4 | 4 | 4 | 5 | 5 | 5 | 223 | 50 | 206 | 13 | 201 | 1 | 5 | 1,63 | 0,08 | 0,21 |
| 2 | **10** | 4 | 4 | 4 | 5 | 5 | 5 | 117 | 0 | 118 | 0 | 117 | 0 | 4 | 0,29 | 0 | 0,11 |
| 3 | 4 | **10** | 4 | 4 | 5 | 5 | 5 | 95 | 146 | 96 | 0 | 96 | 0 | 5 | 1,63 | 0,20 | 0,09 |
| 4 | 4 | 4 | **10** | 4 | 5 | 5 | 5 | 211 | 58 | 108 | 101 | 108 | 0 | 5 | 2,26 | 0,23 | 0,14 |
| 5 | 4 | 4 | 4 | **10** | 5 | 5 | 5 | 224 | 30 | 210 | 14 | 98 | 107 | 5 | 2,69 | 0,23 | 0,17 |
| 6 | 4 | 4 | 4 | 4 | **10** | 5 | 5 | 239 | 16 | 216 | 24 | 199 | 13 | 10 | 2,97 | 0,07 | 0,21 |
| 7 | 4 | 4 | 4 | 4 | 5 | **10** | 5 | 202 | 64 | 194 | 9 | 178 | 10 | 10 | 1,91 | 0,11 | 0,19 |
| 8 | 4 | 4 | 4 | 4 | 5 | 5 | **10** | 214 | 21 | 203 | 12 | 196 | 0 | 8 | 1,80 | 0,04 | 0,20 |
| 9 | 4 | 4 | 4 | 4 | **1** | **1** | **1** | 207 | 102 | 157 | 50 | 139 | 17 | 1 | 0,24 | 0,22 | 0,16 |
| 10 | **1** | 4 | 4 | 4 | 1 | 1 | 1 | 243 | 760 | 179 | 65 | 139 | 38 | 1 | 0,44 | 0,41 | 0,18 |
| 11 | 4 | **1** | 4 | 4 | 1 | 1 | 1 | 219 | 9 | 156 | 63 | 134 | 21 | 1 | 0,16 | 0,15 | 0,16 |
| 12 | 4 | 4 | **1** | 4 | 1 | 1 | 1 | 173 | 64 | 172 | 2 | 138 | 34 | 1 | 0,21 | 0,16 | 0,16 |
| 13 | 4 | 4 | 4 | **1** | 1 | 1 | 1 | 158 | 64 | 134 | 25 | 134 | 00 | 1 | 0,18 | 0,14 | 0,14 |
| 14 | **0,5** | 4 | 4 | 4 | 5 | 5 | 5 | 235 | 1820 | 205 | 29 | 186 | 19 | 5 | 2,92 | 0,36 | 0,2 |
| 15 | 4 | **0,5** | 4 | 4 | 5 | 5 | 5 | 246 | 0 | 209 | 34 | 186 | 22 | 5 | 1,28 | 0,08 | 0,21 |
| 16 | 4 | 4 | **0,5** | 4 | 5 | 5 | 5 | 202 | 35 | 203 | 0 | 191 | 10 | 5 | 1,03 | 0,06 | 0,19 |
| 17 | 4 | 4 | 4 | **0,5** | 5 | 5 | 5 | 213 | 18 | 203 | 10 | 202 | 0 | 5 | 1,00 | 0,04 | 0,20 |
| 18 | 4 | 4 | 4 | 4 | **1** | 5 | 5 | 171 | 93 | 159 | 13 | 156 | 2 | 5 | 0,79 | 0,14 | 0,26 |
| 19 | 4 | 4 | 4 | 4 | 5 | **1** | 5 | 208 | 51 | 155 | 53 | 154 | 0 | 5 | 1,21 | 0,15 | 0,17 |
| 20 | 4 | 4 | 4 | 4 | 5 | 5 | **1** | 228 | 17 | 217 | 8 | 154 | 62 | 5 | 1,19 | 0,13 | 0,20 |

Так, із таблиці ми можемо побачити, що при зміні параметрів моделі відповідають очікувані зміни у результатах. Наприклад, якщо ми піднімемо значення черг, значення не опрацьованих елементів буде падати і навпаки, якщо ми знизимо це значення, кількість не опрацьованих елементів буде рости. Або якщо ми піднімемо затримки, то також кількість не опрацьованих елементів на наступних пристроях буде дуже низькою. Тобто, ми можемо зробити висновок, що модель працює вірно.

Висновок

У даній лабораторній роботі ми успішно побудували алгоритм імітації простої моделі обслуговування одним пристроєм з використанням об’єктно-орієнтованого підходу. Модифікували його таким чином, щоб ми могли будувати більш складні моделі. Провели верифікацію моделі, змінюючи значення вхідних змінних та параметрів моделі та зрозуміли, що модель побудована та працює вірно.