НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»
Кафедра обчислювальної техніки

РОЗРАХУНКОВО-ГРАФІЧНА РОБОТА

з дисципліни «Інтелектуальні вбудовані системи»

на тему: «Дослідження роботи планувальників роботи систем реального часу»

Виконала:
Студентка групи ІП-84
Шахова Поліна Миколаївна
№ залікової книжки: 8424

Перевірив:
доцент Волокіта А.М.

Київ 2021

Дослідження роботи планувальників роботи систем реального часу

Мета роботи - змоделювати роботу планувальника задач у системі реального часу

Основні теоретичні відомості

Планування виконання завдань є однією з ключових концепцій в багатозадачності і багатопроцесорних систем, як в операційних системах загального призначення, так і в операційних системах реального часу. Планування полягає в призначенні пріоритетів процесам в черзі з пріоритетами.

Найважливішою метою планування завдань є якнайповніше завантаження доступних ресурсів. Для забезпечення загальної продуктивності системи планувальник має опиратися на:

- Використання процесора(-ів) — дати завдання процесору, якщо це можливо.
- Пропускна здатність — кількість процесів, що виконуються за одиницю часу.
- Час на завдання — кількість часу, для повного виконання певного процесу.
- Очікування — кількість часу, який процес очікує в черзі готових.
- Час відповіді — час, який проходить від подання запиту до першої відповіді на запит.
- Справедливість — Рівність процесорного часу для кожної нитi.

У середовищах обчислень реального часу, наприклад, на пристроях, призначених для автоматичного управління в промисловості (наприклад, робототехніка), планувальник завдань повинен забезпечити виконання процесів в перебігу заданих часових проміжків (час відгуку); це критично для підтримки коректної роботи системи реального часу.

Система масового обслуговування (СМО) — система, яка виконує обслуговування вимог (заявок), що надходять до неї. Обслуговування вимог у СМО проводиться обслуговуючими приладами. Класична СМО містить від одного до нескінченного числа приладів.

В залежності від наявності можливості очікування вхідними вимогами початку обслуговування СМО(наявності черг) поділяються на:

1. системи з втратами, в яких вимоги, що не знайшли в момент надходження жодного вільного приладу, втрачаються;
2. системи з очікуванням, в яких є накопичувач нескінченної ємності для буферизації вимог, при цьому очікувані вимоги утворюють чергу;
3. системи з накопичувачем кінцевої ємності (чеканням і обмеженнями), в яких довжина черги не може перевищувати ємності накопичувача; при цьому вимога, що надходить в переповнену СМО (відсутні вільні місця для очікування), втрачається.

Вимоги до системи

Вхідні задачі

Вхідними заявками є обчислення, які проводилися в лабораторних роботах 1-3, а саме обчислення математичного очікування, дисперсії, автокореляції, перетворення Фур'є. Вхідні заявки характеризуються наступними параметрами:

1. час приходу в систему – Tp – потік заявок є потоком Пуассона або потоком Ерланга k-го порядку;
2. час виконання (обробки) – To; математичним очікуванням часу виконання є середнє значення часу виконання відповідних обчислень в попередніх лабораторних роботах;
3. крайній строк завершення (дедлайн) – Td, якщо заявка залишається необробленою в момент часу t = Td, то її обробка припиняється і вона покидає систему.

Потік вхідних задач

Потоком Пуассона є послідовність випадкових подій, середнє значення інтервалів між настанням яких є сталою величиною, що дорівнює $1/\lambda$, де λ – інтенсивність потоку.

Потоком Ерланга k-го порядку називається потік, який отримується з потоку Пуассона шляхом збереження кожної (k + i)-ї події (решта відкидаються). Наприклад, якщо зобразити на часовій осі потік Пуассона, поставивши у відповідність кожній події деяку точку, і відкинути з потоку кожну другу подію (точку на осі), то отримаємо потік Ерланга 2-го порядку. Залишивши лише кожну третю точку і відкинувши дві проміжні, отримаємо потік Ерланга 3-го порядку і т.д. Очевидно, що потоком Ерланга 0-го порядку є потік Пуассона.

Пріоритети заявок

Заявки можуть мати пріоритети – явно задані, або обчислені системою (в залежності від алгоритму обслуговування або реалізації це може бути час обслуговування (обчислення), час до дедлайну і т.д.). Заявки в чергах сортуються за пріоритетом.

Є два види обробки пріоритетів заявок:

1. без витіснення – якщо в чергу до ресурсу потрапляє заявка з більшим пріоритетом, ніж та, що в даний момент часу обробляється ним, то вона чекає завершення обробки ресурсом його задачі.
2. з витісненням – якщо в чергу до ресурсу потрапляє заявка з більшим пріоритетом, ніж та, що в даний момент часу обробляється ним, то вона витісняє її з обробки; витіснена задача стає в чергу. В даній роботі алгоритми реалізовані без витіснення за явно заданими пріоритетами. При цьому деякі алгоритми можуть витісняти задачі базуючись на внутрішніх обчислюваних пріоритетах.

Дисципліни обслуговування

Вибір заявки з черги на обслуговування здійснюється за допомогою так званої дисципліни обслуговування. Їх прикладами є FIFO (прийшов першим - обслуговується першим), LIFO (прийшов останнім - обслуговується першим), RANDOM (випадковий вибір). У системах з очікуванням накопичувач в загальному випадку може мати складну структуру.
В даній роботі використовувалися дисципліни обслуговування EDF(Earliest Deadline First) та RM(Rate Monotonic)

Дисципліна EDF

Алгоритм EDF(спочатку найперший термін) є динамічним алгоритмом планування з динамічним пріоритетом. Планувальник надає перевагу задачі, що знаходиться найближче до свого терміну виконання. Цей алгоритм гарантує високу вірогідність виконання задачі в заданий термін, але через це може відбуватись затриманні заявок з довгим терміном виконання в черзі.

Дисципліна RM

Алгоритм RM це динамічний алгоритм з статичними пріоритетами заявок. Алгоритм в першу чергу обробляє ті заявки що надходять частіше. Через це черга заявок при виконанні планування менша ніж в інших алгоритмів, але збільшується ризик порушення дедлайну задач, що приходять рідше.
Цей алгоритм може використовуватися для процесів, які відповідають таким умовам:

1) Кожен періодичний процес має бути завершений за час його періоду.
2) Жоден процес не повинен залежати від будь-якого іншого процесу.
3) Кожному процесу потрібно однаковий процесорний час на кожному інтервалі.
4) У неперіодичних процесів немає жорстких термінів.
5) Переривання процесу відбувається миттєво, без накладних витрат.

Завдання на лабораторну роботу

1. Змоделювати планувальник роботи системи реального часу. Дві дисципліни планування.

2. Знайти наступні значення:

1) середній розмір вхідної черги заявок, та додаткових черг (за їх наявності);

2) середній час очікування заявки в черзі;

3) кількість прострочених заявок та її відношення до загальної кількості заявок

3. Побудувати наступні графіки:

1) Графік залежності кількості заявок від часу очікування при фіксованій інтенсивності вхідного потоку заявок.

2) Графік залежності середнього часу очікування від інтенсивності вхідного потоку заявок.

3) Графік залежності проценту простою ресурсу від інтенсивності вхідного потоку заявок

Лістинг програми

```
import random
import numpy
import matplotlib.pyplot as plt

avg_solution_time1 = (0.0020012855529785156 + 0.052039384841918945)/2
avg_solution_time2 = (0.02399897575378418 + 0.16211581230163574)/2
first_range = (0.0020012855529785156, 0.052039384841918945)
second_range = (0.02399897575378418, 0.16211581230163574)

class Task:
    def __init__(self, t, s_t, d):
        self.time = t
        self.solution_time = s_t
        self.deadline = d

def GenerateQueue(periods, solution_time, range_random, tact_size):
    queue = []
    for i in range(len(periods)):
        tasks_per_tact = []
        for j in range(periods[i]):
            error = random.uniform(range_random[0], range_random[1])/2
            if random.random() < 0.5:
                tasks_per_tact.append(Task(i * tact_size, solution_time +
error, i * tact_size + (solution_time + error) * random.randint(2,10)))
            else:
                tasks_per_tact.append(Task(i * tact_size, solution_time -
error, i * tact_size + (solution_time + error) * random.randint(2,10)))
```

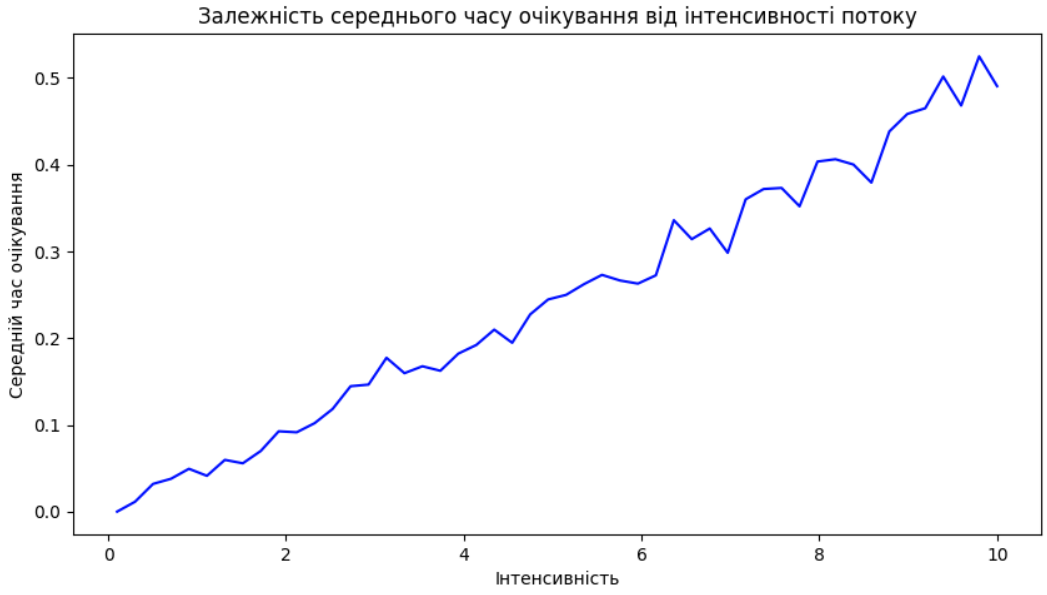
```
            queue.append(tasks_per_tact)
        return queue

    def SMO(queue, tact_size, number_of_tasks, type_smo):
        Statistic = {
        'average_wait_in_system' : 0,
        'solved_tasks' : 0,
        'average_queue_length' : [],
        'free_time' : 0,
        'skiped_tasks' : []}
        real_time_queue = []
        time = 0
        in_processor = 0
        remained_time = 0

        while True:
            #delete overdue tasks
            if in_processor == 0:
                before = len(real_time_queue)
                real_time_queue = [i for i in real_time_queue if i.deadline >
time]
                after = len(real_time_queue)
                # print(before, after)
                diff = before - after
                Statistic['skiped_tasks'].append(diff)

            #if all tasks already prepared
            if len(real_time_queue) == 0 and len(queue) == 0:
                print("Finished.")
                break
            #add tasks if it is new tact
            if len(queue) != 0 and remained_time == 0:
                real_time_queue += queue.pop(0)
                Statistic['average_queue_length'].append(len(real_time_queue))

            #if processor is empty - choose the shortest
            if in_processor == 0:
                if type_smo == 'RM':
                    real_time_queue.sort(key=lambda x: x.solution_time)
                elif type_smo == 'EDF':
                    real_time_queue.sort(key=lambda x: x.deadline)

            if len(real_time_queue) != 0:
                #calculate tasks in the same tact
                if remained_time != 0:
                    if real_time_queue[0].solution_time > remained_time:
                        time += remained_time
                        real_time_queue[0].solution_time -= remained_time
                        in_processor = 1
                        remained_time = 0
                    else:
                        remained_time -= real_time_queue[0].solution_time
                        time += real_time_queue[0].solution_time
                        Statistic['average_wait_in_system'] += time -
real_time_queue[0].time - real_time_queue[0].solution_time
                        Statistic['solved_tasks'] += 1
                        del real_time_queue[0]
                        in_processor = 0
                #calculate in the new tact
                else:
                    if real_time_queue[0].solution_time > tact_size:
                        real_time_queue[0].solution_time -= tact_size
                        in_processor = 1
                        remained_time = 0
```

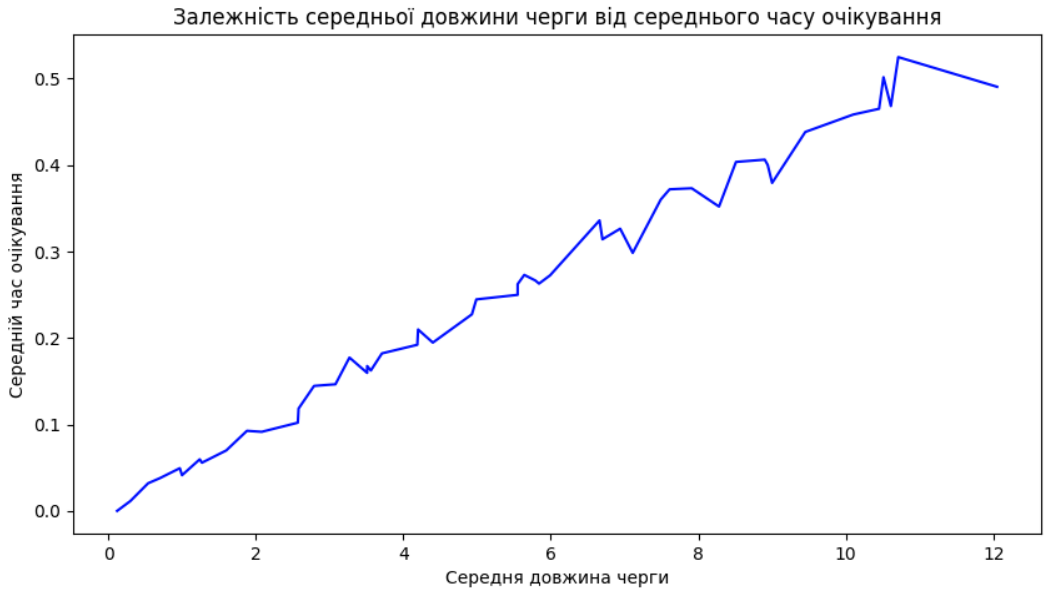
```
                    time += tact_size
                else:
                    time += real_time_queue[0].solution_time
                    Statistic['average_wait_in_system'] += time -
real_time_queue[0].time - real_time_queue[0].solution_time
                    Statistic['solved_tasks'] += 1
                    remained_time = tact_size -
real_time_queue[0].solution_time
                    in_processor = 0
                    del real_time_queue[0]
        #if real queue is empty add remained time and start next tact
        else:
            if remained_time != 0:
                Statistic['free_time'] += remained_time
                time += remained_time
                remained_time = 0
        #add tact time to system time if no tasks in current tact
            else:
                Statistic['free_time'] += tact_size
                time += tact_size

    #print(time, number_of_tasks * tact_size)
    Statistic['average_wait_in_system'] = Statistic['average_wait_in_system']
/ Statistic['solved_tasks']
    Statistic['average_queue_length'] =
sum(Statistic['average_queue_length']) / number_of_tasks
    return Statistic

def Execute(lambda_p, size, avg_sol_time, range1, tact_size, type_smo):
    period = numpy.random.poisson(lambda_p, size)
    period_k= [period[j] for j in range(len(period)) if not j % 3]
    period_k=numpy.array(period_k)
    queue = GenerateQueue(period, avg_sol_time, range1, tact_size)
    Stats  = SMO(queue, tact_size, size, type_smo)
    print("SMO type: ",type_smo)
    print("Intensity: ",lambda_p)
    print("Number of tasks : {} Solved : {}".format(sum(period),
Stats['solved_tasks']))
    return Stats

def CreateGraph(type_smo, tact_size,avg_solution_time,second_range, color):
    n=3
    wait_time, queue_length, free_time = [], [], []
    for i in numpy.linspace(0.1, 10):
        one_smo = Execute(i, 100, avg_solution_time, second_range, tact_size,
type_smo)
        wait_time.append(one_smo['average_wait_in_system'])
        queue_length.append(one_smo['average_queue_length'])
        free_time.append(one_smo['free_time'])


    figure1 = plt.figure(figsize=(10,5))
    figure1.canvas.set_window_title(type_smo)
    t_i = figure1.add_subplot(111)
    t_i.set_title("Залежність середнього часу очікування від інтенсивності
потоку")
    t_i.set_xlabel('Інтенсивність')
    t_i.set_ylabel('Середній час очікування')
    t_i.plot(numpy.linspace(0.1, 10), wait_time, color = color)
    plt.subplots_adjust(wspace=0.1, hspace=1, bottom=0.1, top=0.9)
    plt.show()


    figure2 = plt.figure(figsize=(10,5))
```

```
    figure2.canvas.set_window_title(type_smo)
    w_ql = figure2.add_subplot(111)
    w_ql.set_title("Залежність середньої довжини черги від середнього часу
очікування")
    w_ql.set_xlabel('Середня довжина черги')
    w_ql.set_ylabel('Середній час очікування')
    w_ql.plot(sorted(queue_length), wait_time, color = color)
    plt.subplots_adjust(wspace=0.1, hspace=1, bottom=0.1, top=0.9)
    plt.show()


    figure3 = plt.figure(figsize=(10,5))
    figure3.canvas.set_window_title(type_smo)
    f_p = figure3.add_subplot(111)
    f_p.set_title("Залежність часу простою процесора від інтенсивності
потоку")
    f_p.set_xlabel('Інтенсивність')
    f_p.set_ylabel('Час простою')
    f_p.plot(numpy.linspace(0.1, 10), free_time, color = color)
    plt.subplots_adjust(wspace=0.1, hspace=1, bottom=0.1, top=0.9)
    plt.show()


CreateGraph('RM', 1,avg_solution_time1,first_range, '#00FFFF')
CreateGraph('EDF', 1,avg_solution_time2,second_range, '#000FFF')
```

Результати виконання програми

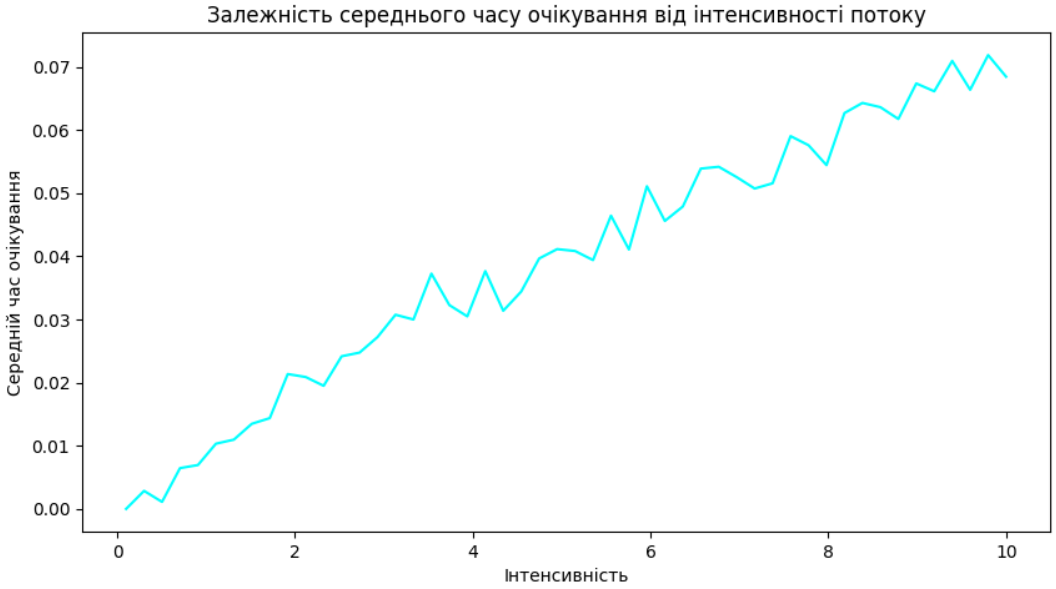
Rate Monotonic

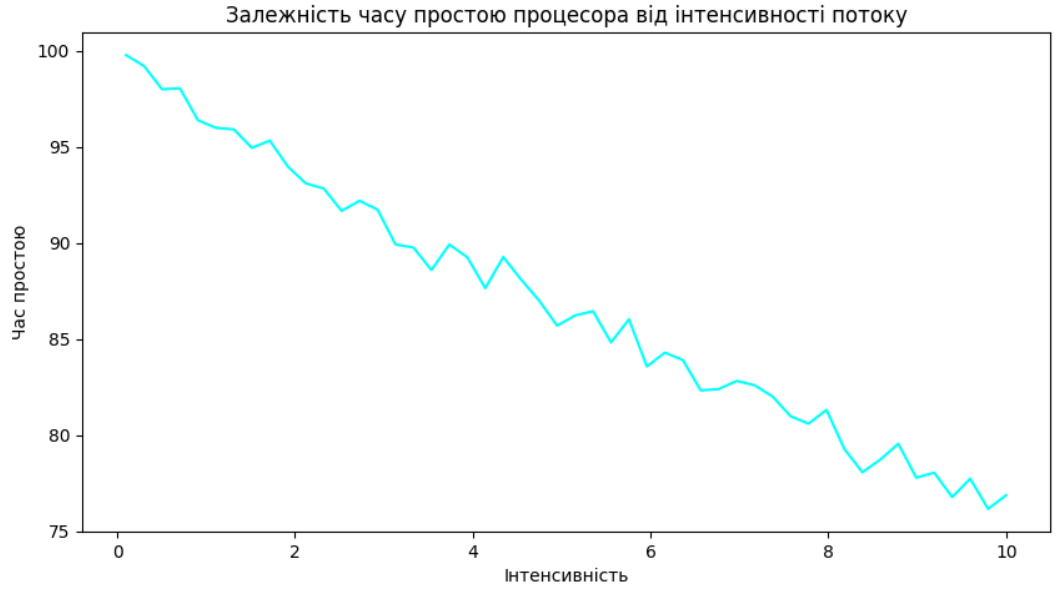



Залежність середньої довжини черги від середнього часу очікування
Середній час очікування
Середня довжина черги
0.0
0.1
0.2
0.3
0.4
0.5
0
2
4
6
8
10
12


Залежність часу простою процесора від інтенсивності потоку
Час простою
Інтенсивність
20
40
60
80
100
0
2
4
6
8
10

Earliest Deadline First





Висновок

У даній роботі була змодельована робота планувальника задач у системі реального часу. Для цієї роботи були використанні дві дисципліни обслуговування: RM та EDF, та потік Ерланга.

У протоколі були описані основні теоретичні відомості для виконання данної роботи. Було розглянуто, що таке планування виконання завдань, система масового обслуговування, потік вхідних задач, пристрій обслуговування, пріоритети заявок. Також, була описана дисципліна EDF та дисципліна RM, які використані для данної роботи.

Було розроблено программу для планування з використання дисциплін RM і EDF. Після виконання роботи программи були отримані наступні графіки:

- Графік залежності середнього часу очікування від інтенсивності вхідного потоку заявок.
- Графік залежності середньої довжини черги від середнього часу очікування
- Графік залежності проценту простою ресурсу від інтенсивності вхідного потоку заявок

Як бачимо з першого графіку, зі збільшенням інтенсивності зростає середній час очікування заявок для обох дисциплін. У дисципліни RM середній час очікування більший за EDF. Можемо зробити висновок, що у цьому плані EDF вигідніший ніж RM. При збільшенні інтенсивності різниця у часі очікування EDF та RM стає досить суттєвою.

Як бачимо з другого графіку, зі збільшенням кількості задач у черзі, середній час очікування також збільшується для обох дисциплін.

Як бачимо з третього графіку, що зі збільшенням інтенсивності вхідного потоку заявок процент простою ресурсу зменшується. Це обумовлено тим,

що планувальник постійно обробляє вхідні заявки, та не простоює. Також, варто додати, що у обох дисциплінах графіки майже однакові, планувальники добре виконують свою роботу.