Міністерство освіти і науки України

Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

Розрахунково-графічна робота

з дисципліни

«Інтелектуальні вбудовані системи»

на тему

«Дослідження роботи планувальників систем реального часу»

|  |  |  |
| --- | --- | --- |
|  |  |  |
| Виконала: |  | Перевірив: |
| студентка групи ІП-843 |  | викладач |
| Леут Марія Віталіївна |  | Регіда Павло Геннадійович |

номер залікової книжки: 8313

Київ 2021

# Основні теоретичні відомості

# Планування виконання завдань (англ. Scheduling) 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 нитки.

# 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 надійшли вимог, при цьому очікують 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. Основні поняття СМО:

• Вимога (заявка) — запит на обслуговування.

• Вхідний потік вимог — сукупність вимог, що надходять у СМО.

• Час обслуговування - період часу, протягом якого обслуговується вимогу.

**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 (інтенсивність потоків та їх порядок задаються варіантом);

2) час виконання (обробки) – То; математичним очікуванням часу виконання є середнє значення часу виконання відповідних обчислень в попередніх лабораторних роботах;

3) крайній строк завершення (дедлайн) – Тd –задається (випадково?); 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/λ , де λ – інтенсивність потоку.

Потоком Ерланга k-го порядку називається потік, який отримується з потоку Пуассона шляхом збереження кожної (k+1)-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.

***Пристрій обслуговування***

Пристрій обслуговування складається з P незалежних рівноправних обслуговуючих приладів - обчислювальних ресурсів (процесорів). Кожен ресурс обробляє заявки, які йому надає планувальник та може перебувати у двох станах – вільний та зайнятий. Обробка заявок може виконуватися повністю (заявка перебуває на обчислювальному ресурсі доти, доки не обробиться повністю) або поквантово (ресурс обробляє заявку лише протягом певного часу – кванту обробки – і переходить до обробки наступної заявки).

***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.

***Дисципліна 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.

**Дисципліна FIFO**

Алгоритм планування First In, First Out (FIFO), - це найпростіший алгоритм планування. FIFO просто виконує процеси в тому порядку, в якому вони надходять у готову чергу. При цьому процес, який приходить першим, буде виконано першим, а наступний процес розпочнеться лише після повного виконання попереднього.

**Дисципліна EDF**

Алгоритм планування Earliest Deadline First (по найближчому строку завершення) використовується для встановлення черги заявок в операційних системах реального часу. При наставанні події планування (завершився квант часу, прибула нова заявка, завершилася обробка заявки, заявка прострочена) відбувається пошук найближчої до крайнього часу виконання (дедлайну) заявки і призначення її виконання на перший вільний ресурс або на той, який звільниться найшвидше.

**Дисципліна RM**

Алгоритм планування Rate Monotonic - це пріоритетний алгоритм, який належить до алгоритмів планування операційних систем реального часу зі статичними пріоритетами. Пріоритет визначається відповідно до часу виконання процесів, які беруть участь. Якщо процес має невелику тривалість роботи, то він має найвищий пріоритет. Таким чином, якщо процес з найвищим пріоритетом розпочинає виконання, він буде випереджати інші запущені процеси. Пріоритет процесу обернено пропорційний періоду, протягом якого він триватиме.

**Завдання:**

1. Змоделювати планувальник роботи системи реального часу. Три дисципліни планування: перша – FIFO, друга – EDF, третя – RM.

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 numpy as np

from operator import attrgetter

import matplotlib.pyplot as plt

class Task:

def \_\_init\_\_(self, Tp, To, Td):

self.Tp = Tp

self.To = To

self.Td = Td

self.progress = 0

def \_\_lt\_\_(self, other):

return self.Tp < other.Tp

def \_\_repr\_\_(self):

return f'arrive\_time: {self.Tp}, exec\_time: {self.To}, deadline: {self.Td}\n'

def Stream(tact\_count, frequency, exec\_time, k):

tasks = []

start\_time = 0

while start\_time + exec\_time < tact\_count:

deadline = start\_time + exec\_time \* k

task = Task(start\_time, exec\_time, deadline)

tasks.append(task)

interval = np.random.poisson(frequency)

start\_time += interval

return tasks

class Scheduler:

def \_\_init\_\_(self, tact\_count, processor\_count):

self.tact\_count = tact\_count

self.processor\_count = processor\_count

self.downtime = 0

self.waiting\_times = []

self.queue\_sizes = [0] \* self.tact\_count

self.task\_count = 0

self.expired\_task\_count = 0

def get\_avg\_queue\_size(self):

return round(sum(self.queue\_sizes) / self.tact\_count)

def get\_avg\_waiting\_time(self):

if not self.waiting\_times:

return 0

return round(sum(self.waiting\_times) / len(self.waiting\_times))

def get\_downtime\_percent(self):

return round(self.downtime / self.tact\_count \* 100, 2)

def get\_expired\_task\_percent(self):

return round(self.expired\_task\_count / self.task\_count \* 100)

def printCharacteristics(self):

print("Average queue size: ", self.get\_avg\_queue\_size())

print("Average waiting time: ", self.get\_avg\_waiting\_time())

print("Downtime percent: ", self.get\_downtime\_percent())

print("Expired tasks count: ", self.expired\_task\_count)

print("Expired tasks percent: ", self.get\_expired\_task\_percent())

def clearCharacteristics(self):

self.downtime = 0

self.waiting\_times = []

self.queue\_sizes = [0] \* self.tact\_count

self.task\_count = 0

self.expired\_task\_count = 0

def schedule(self, stream, sorting\_par = None):

stream.sort()

currentTasks = [None] \* self.processor\_count

self.clearCharacteristics()

queue = []

for tact in range(self.tact\_count):

for task in stream:

if task.Tp > tact:

break

queue.append(stream.pop(0))

self.queue\_sizes[tact] = len(queue)

if sorting\_par:

queue.sort(key = attrgetter(sorting\_par))

for p in range(self.processor\_count):

if not currentTasks[p]:

if not queue:

self.downtime += 1 / self.processor\_count

continue

else:

currentTasks[p] = queue.pop(0)

self.waiting\_times.append(tact - currentTasks[p].Tp)

while currentTasks[p].Td < tact:

self.waiting\_times.append(tact - currentTasks[p].Tp)

print(f'Tact: {tact}, processor: {p}')

print("Task expired: ", currentTasks[p])

self.task\_count += 1

self.expired\_task\_count += 1

if not queue:

break

currentTasks[p] = queue.pop(0)

if sorting\_par:

if queue:

if getattr(queue[0], sorting\_par) < getattr(currentTasks[p], sorting\_par):

stream.insert(0, currentTasks[p])

currentTasks[p] = queue.pop(0)

currentTasks[p].progress += 1

if currentTasks[p].progress >= currentTasks[p].To:

print(f'Tact: {tact}, processor: {p}')

print("Task completed: ", currentTasks[p])

self.task\_count += 1

currentTasks[p] = None

def fifo(self, stream):

print('-----FIFO---------')

self.schedule(stream)

def rm(self, stream):

print('-----EDF---------')

self.schedule(stream, 'To')

def edf(self, stream):

print('-----RM---------')

self.schedule(stream, 'Td')

def plot\_waiting\_time(schedule\_func):

avg\_waiting\_times = [0] \* 50

for λ in range(1,50):

t1 = Stream(100, λ, 10, 2)

t2 = Stream(100, λ, 5, 3)

t3 = Stream(100, λ, 20, 5)

t = t1 + t2 + t3

sch = Scheduler(100, 3)

getattr(sch, schedule\_func)(t)

avg\_waiting\_times[λ] = round(sum(sch.waiting\_times) / len(sch.waiting\_times))

plt.plot(avg\_waiting\_times)

plt.xlim(1, 50)

plt.xlabel("λ")

plt.ylabel("Average waiting time")

plt.title("Залежність середнього часу очікування від інтенсивності вхідного потоку заявок")

plt.legend()

plt.show()

def plot\_downtime\_percent(schedule\_func):

downtime\_percents = [0] \* 50

for λ in range(1,50):

t1 = Stream(100, λ, 10, 2)

t2 = Stream(100, λ, 5, 3)

t3 = Stream(100, λ, 20, 5)

t = t1 + t2 + t3

sch = Scheduler(100, 3)

getattr(sch, schedule\_func)(t)

downtime\_percents[λ] = round(sch.downtime / sch.tact\_count \* 100, 2)

plt.plot(downtime\_percents)

plt.xlim(1, 50)

plt.xlabel("λ")

plt.ylabel("Downtime percent")

plt.title("Залежність проценту простою ресурсу від інтенсивності вхідного потоку заявок")

plt.legend()

plt.show()

def plot\_task\_count(schedule\_func):

λ = 10

t1 = Stream(100, λ, 10, 2)

t2 = Stream(100, λ, 5, 3)

t3 = Stream(100, λ, 20, 5)

t = t1 + t2 + t3

sch = Scheduler(100, 3)

getattr(sch, schedule\_func)(t)

waiting\_times = dict((x,sch.waiting\_times.count(x)) for x in set(sch.waiting\_times))

plt.bar(range(len(waiting\_times)), waiting\_times.values())

plt.xlabel("λ")

plt.ylabel("Average waiting time")

plt.title("Залежність кількості заявок від часу очікування")

plt.show()

sch.printCharacteristics()

def main():

plot\_task\_count('fifo')

plot\_task\_count('edf')

plot\_task\_count('rm')

plot\_waiting\_time('fifo')

plot\_waiting\_time('edf')

plot\_waiting\_time('rm')

plot\_downtime\_percent('fifo')

plot\_downtime\_percent('edf')

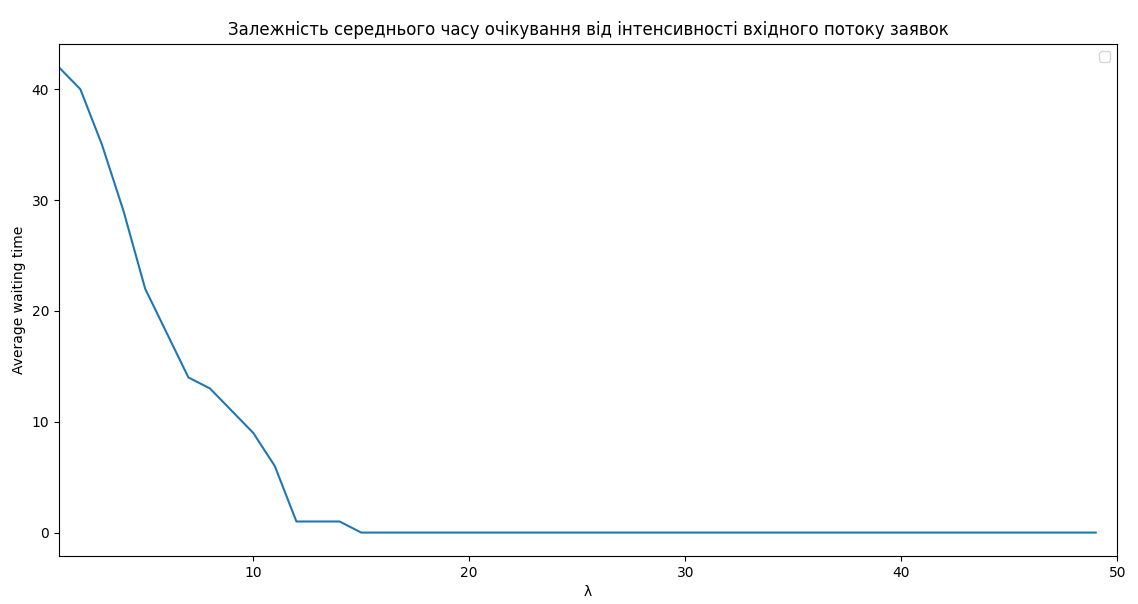
plot\_downtime\_percent('rm')

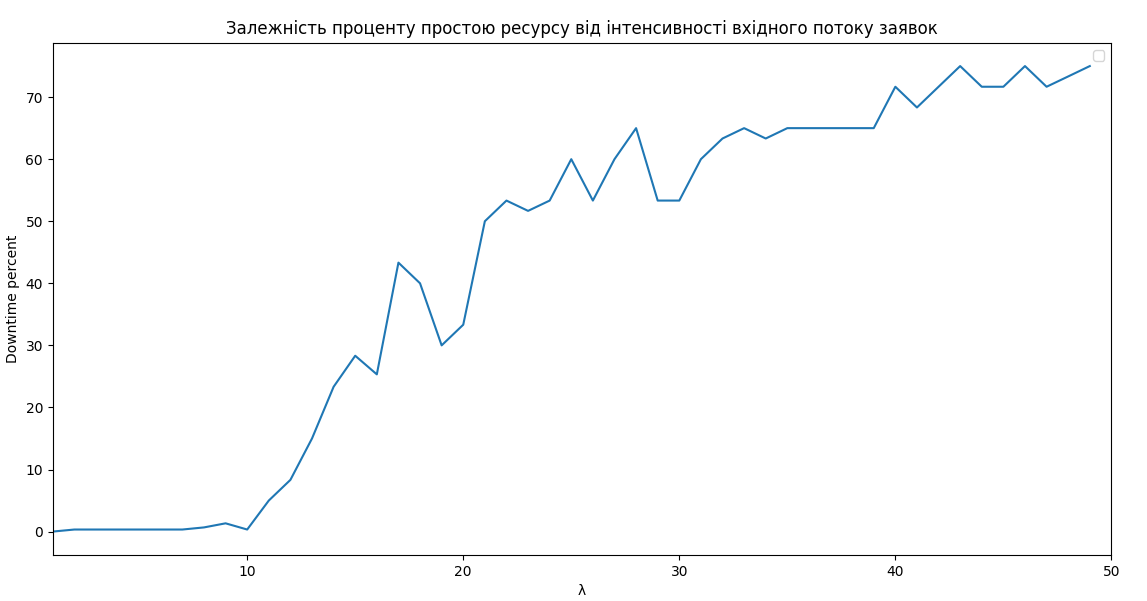
main()

**Результати виконання програми**

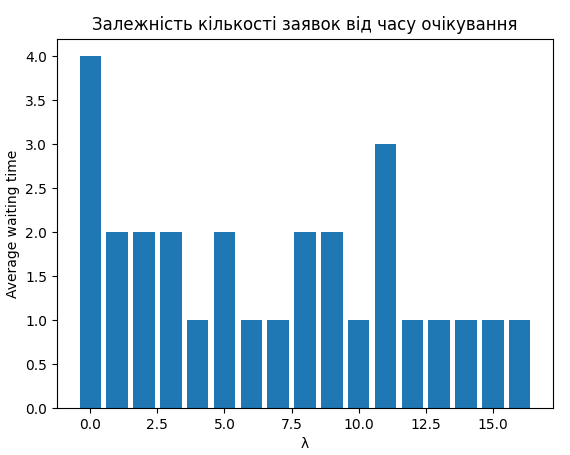
**FIFO**

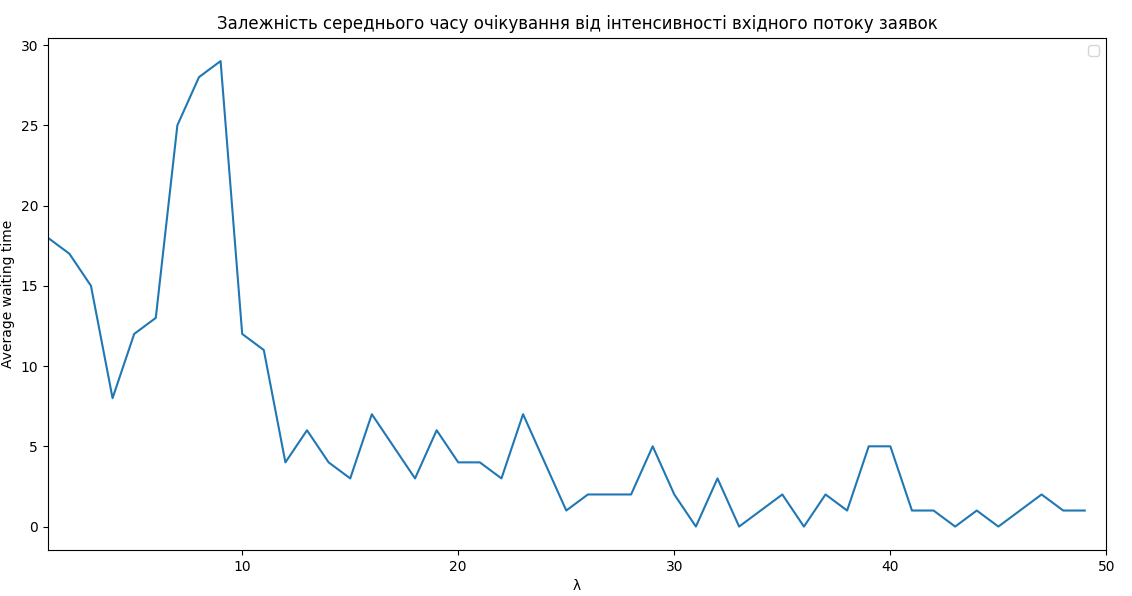
****

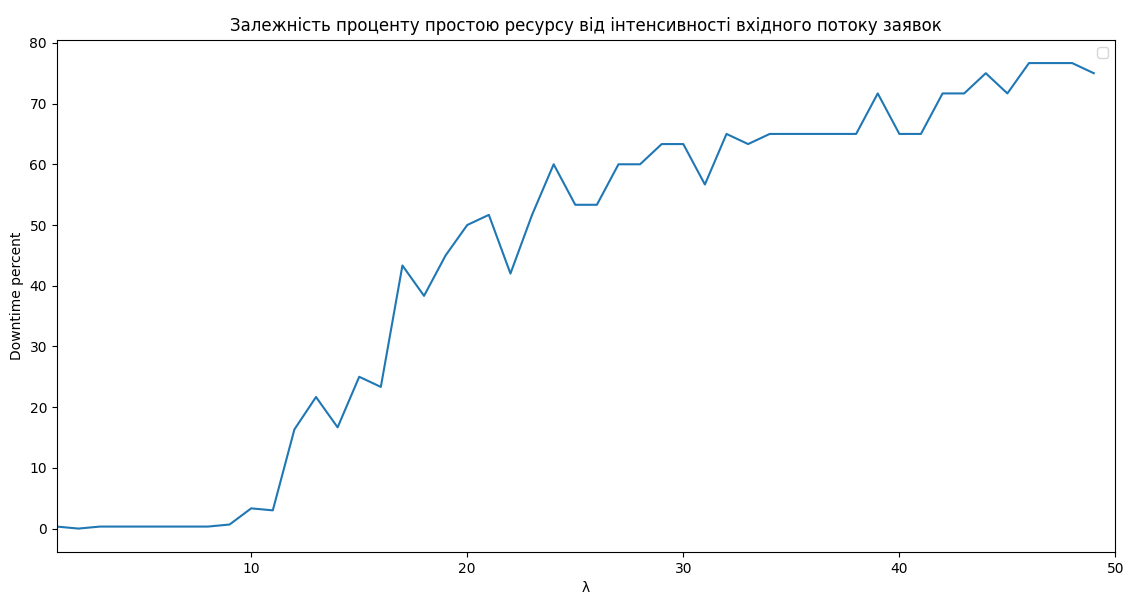
****

****

**EDF**

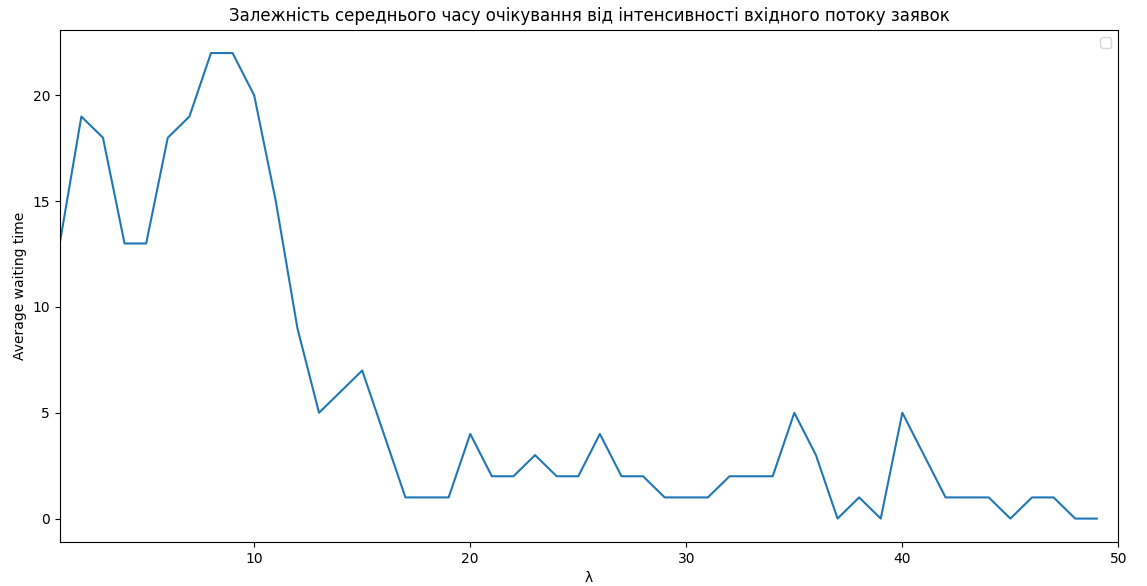
****

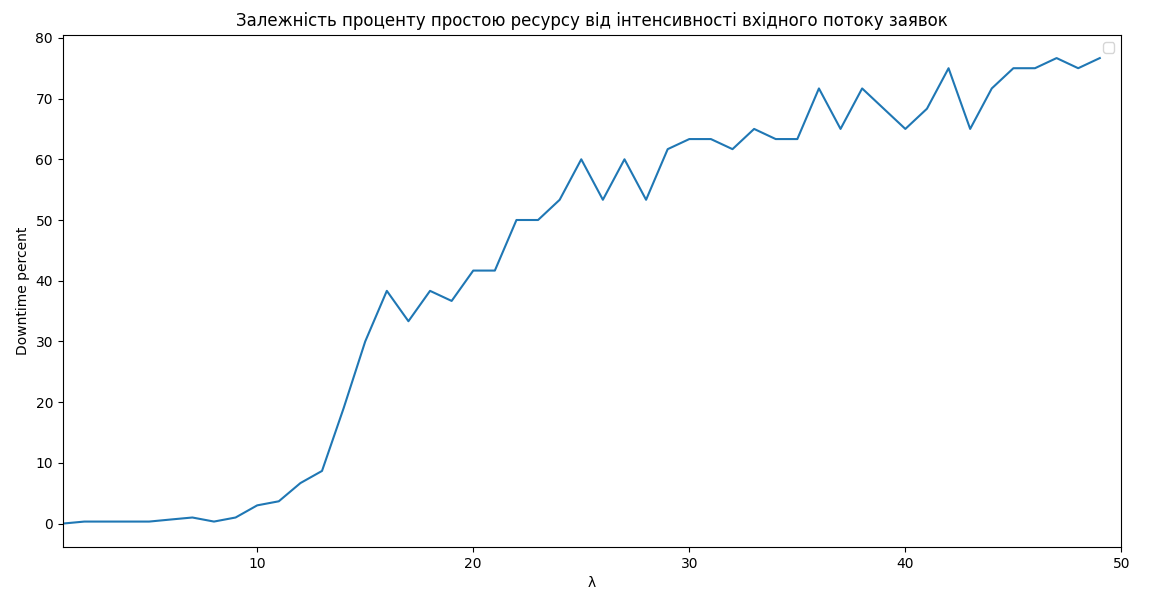
****

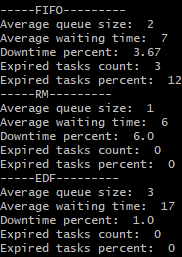
****

**RM**

****

****

****

****

# Висновки

При виконанні цієї лабораторної роботи ми дослідили роботу трьох алгоритмів планування задач у системі реального часу: FIFO, RM та EDF. Змоделювали роботу планувальника з використанням кожного з трьох алгоритмів та порівняли їхні характеристики. Побудували відповідні графіки. Мету роботи виконано.