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

Кафедра обчислювальної техніки
(повна назва кафедри, циклової комісії)

РОЗРАХУНКОВО-ГРАФІЧНА РОБОТА

з дисципліни «Програмування вдубованих систем»
(назва дисципліни)

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

Студент 3 курсу групи ІП-84
спеціальності
121 «Інженерія програмного
забезпечення»

Печена М.В.
(прізвище та ініціали)

Керівник доцент Волокіта А.М.

Київ – 2021 рік

Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”

Факультет (інститут) інформатики та обчислювальної техніки
(повна назва)

Кафедра обчислювальної техніки
(повна назва)

Освітньо-кваліфікаційний рівень бакалавр

Напрям підготовки 121 «Інженерія програмного забезпечення»
(шифр і назва)

З *А В Д А Н Н Я*

НА РОЗРАХУНКОВО-ГРАФІЧНУ РОБОТУ

Голубову Івану Олеговичу
(прізвище, ім’я, по батькові)

1. Тема роботи «Дослідження роботи планувальників роботи систем реального часу»
керівник роботи Волокіта Артем Миколайович к.т.н., доцент
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)

2. Строк подання студентом роботи 8 червня 2021 р.
3. Вхідні дані до роботи
- гармонічний сигнал x = A*sin(wt+f)
- Алгоритм, кількість заявок, мінімальна вага заявки, максимальна вага заявки, інтенсивність
- мови (бібліотеки) програмування: Python
- середовище розробки: PyCharm

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
- огляд алгоритмів планувальника
- розробка і тестування програми

5. Перелік графічного матеріалу
- скріншоти програми

7. Дата видачі завдання 15 04 2021 .

ЗМІСТ

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 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 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) – To; математичним очікуванням часу виконання є середнє значення часу виконання відповідних обчислень в попередніх лабораторних роботах;

3) крайній строк завершення (дедлайн) – Td –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 $1/\lambda$, 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 FIFO (first in, first out - перший прийшов, перший вийшов) є одним із найпростіших алгоритмів планування і полягає у виборі заявок у мірі їх надходження. Під час надходження заявок алгоритм ігнорує всі параметри, крім часу надходження. Щоразу вибирається та заявка, яка прийшла раніше. Цим по суті своїй FIFO аналогічний поведінці людей, що стоять у черзі. Через надмірну простоту та ігнорування особливостей заявок, FIFO мало використовується. Іноді використовується у ОС для планування процесів, тоді час надається кожному процесу в порядку їхнього надходження на обслуговування. Належить до статичних алгоритмів планування

Дисципліна RM

Алгоритм RM (Rate Monotonic) належить до динамічних алгоритмів планування із фіксованими пріоритетами. Планувальник визначає частоту кожного типу заявок та виконує спочатку ті заявки, що мають більшу частоту. Таким чином досягається зменшення накопиченої черги заявок, однак можливе порушення дедлайнів більш довгих заявок, якщо черга надто забивається малими заявками. [2]

Дисципліна EDF

Алгоритм EDF (Earliest Deadline First - найранніший дедлайн перший) належить до динамічних алгоритмів планування із динамічним пріоритетом.

Планувальник надає обчислювальні потужності тій заявці, чий дедлайн найближче. Таким чином досягається максимальне виконання дедлайнів, але може збільшуватися час очікування заявки в системі (менш термінові заявки посуваються). [3]

Розробка програми

Мова програмування: Python
Клас SMO

```
class
SMO:
            currentTime = 0
            Q = []  # Task queue
            Qready = []
            Tw = [0.0 for _ in range(10000)]
            Tn = [0 for _ in range(10000)]
            faults = [0 for _ in range(10000)]
            currentTask = None

            def __init__(self, Q):
                self.Q = Q

            def GetEDTask(self, removeFromQ):
                timeBuf = 9999999
                taskwED = None
                for i in range(len(self.Qready)):
                    newTime = self.Qready[i].GetDeadline()
                    if timeBuf > newTime:
                        timeBuf = newTime
                        taskwED = self.Qready[i]
                if (removeFromQ):
                    self.Qready.remove(taskwED)
                return taskwED

            def ToReadyQueue(self):
                for i in range(len(self.Q)):
                    if self.Q[i].GetCreationTime() == self.currentTime:
                        self.Qready.append(self.Q[i])

            def CheckForDeadlines(self):
                flt = 0
                flti = []
                for i in range(len(self.Qready)):
                    if self.Qready[i].GetDeadline() < self.currentTime:
                        flt += 1
                        flti.append(i)
                if self.currentTime == 0:
                    self.faults[self.currentTime] = flt
                else:
                    self.faults[self.currentTime] = self.faults[self.currentTime - 1] + flt
                for i in range(len(flti)):
                    del self.Qready[flti[i]]
```

```
            for j in range(i, len(flti)):
                flti[j] -= 1

    def Work(self,isRM, isEDF):
        self.currentTime = 0

        for self.currentTime in range(10000):
            if self.currentTime != 0:
                self.Tn[self.currentTime] = self.Tn[self.currentTime - 1]
            timewait = 0
            self.CheckForDeadlines()
            self.ToReadyQueue()
            if self.currentTask is not None and self.currentTask.GetExecutionTime() ==
0:
                self.currentTask = None
            elif self.currentTask is not None and self.GetEDTask(False) is not None
and isRM:
                if self.GetEDTask(False).GetExecutionTime() <
self.currentTask.GetExecutionTime():
                    self.Qready.append(self.currentTask)
                    self.currentTask = self.GetEDTask(True)
            elif self.currentTask is not None and self.GetEDTask(False) is not None
and isEDF:
                if self.GetEDTask(False).GetDeadline() <
self.currentTask.GetDeadline():
                    self.Qready.append(self.currentTask)
                    self.currentTask = self.GetEDTask(True)
            elif self.currentTask is not None:
                self.currentTask.WorkedOn()
            if self.GetEDTask(False) is None and self.currentTask is None:
                self.Tn[self.currentTime] += 1
                continue
            elif self.currentTask is None:
                self.currentTask = self.GetEDTask(True)
            for task in self.Qready:
                task.Wait()
                #timewait += 1
            self.Tw[self.currentTime] += 1# timewait

    def GetWaitTimes(self):
        return self.Tw

    def GetFaults(self):
        return self.faults

    def GetProcessorFreeTime(self):
        return self.Tn
```

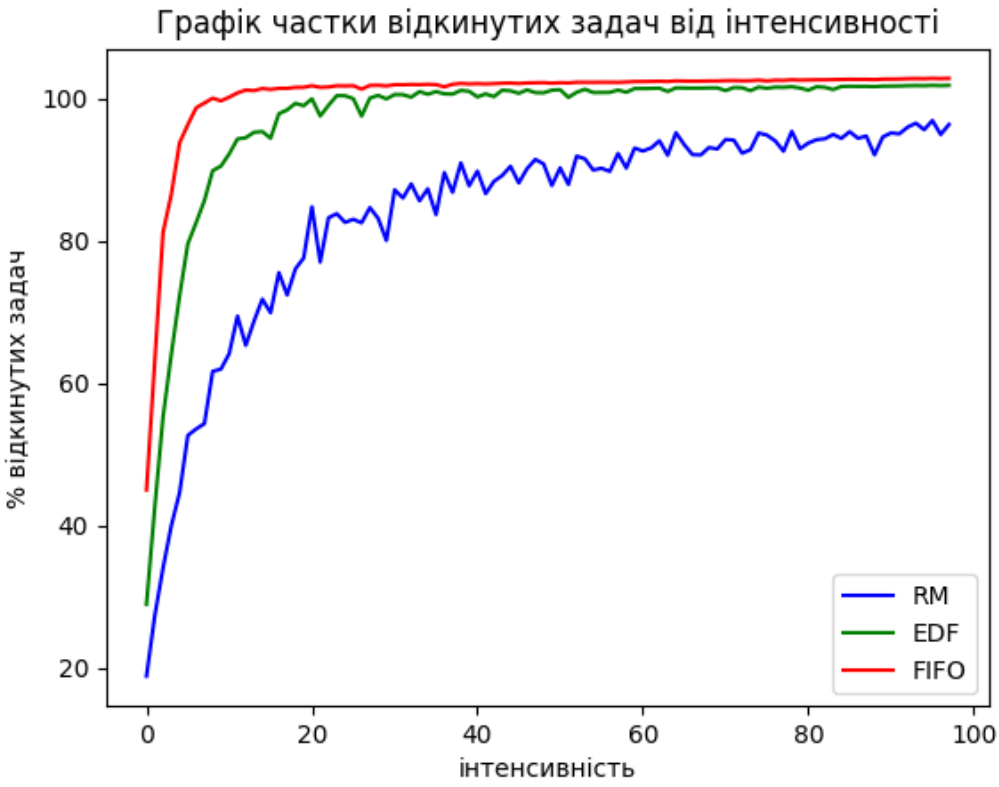
```
import random
import math


class ErlangDistribution:
    k = 0
    lam = 0

    def __init__(self, k, lam):
        self.lam = lam
        self.k = k
        if (k == 0):
            raise Exception("Order parameter can't be less than 1!")
        if (lam <= 0):
            raise Exception("Streaming rate can't be less or equal 0!")

    def GenerateNext(self):
        res = 0
        for n in range(self.k - 1):
            if (res != 0):
                res = random.random() * res
            else:
                res = random.random()
        res = 0 - (math.log(res) / self.lam)
        return res

    def GenerateNextInternal(self):
        return random.gammavariate(self.k, self.lam)

    def ChangeLambda(self, lam):
        self.lam = lam
 class
 Task:
             deadline = 0
             creationTime = 0
             executionTime = 0
             deadlineMultiplier = 0
             waitTime = 0

             def __init__(self, creationTime, executionTime):
                 self.executionTime = int(executionTime)
                 self.creationTime = int(creationTime)
                 self.deadline = int(creationTime + executionTime *
         self.deadlineMultiplier)

             def GetTimeLimit(self):
                 return self.deadline + self.creationTime
```

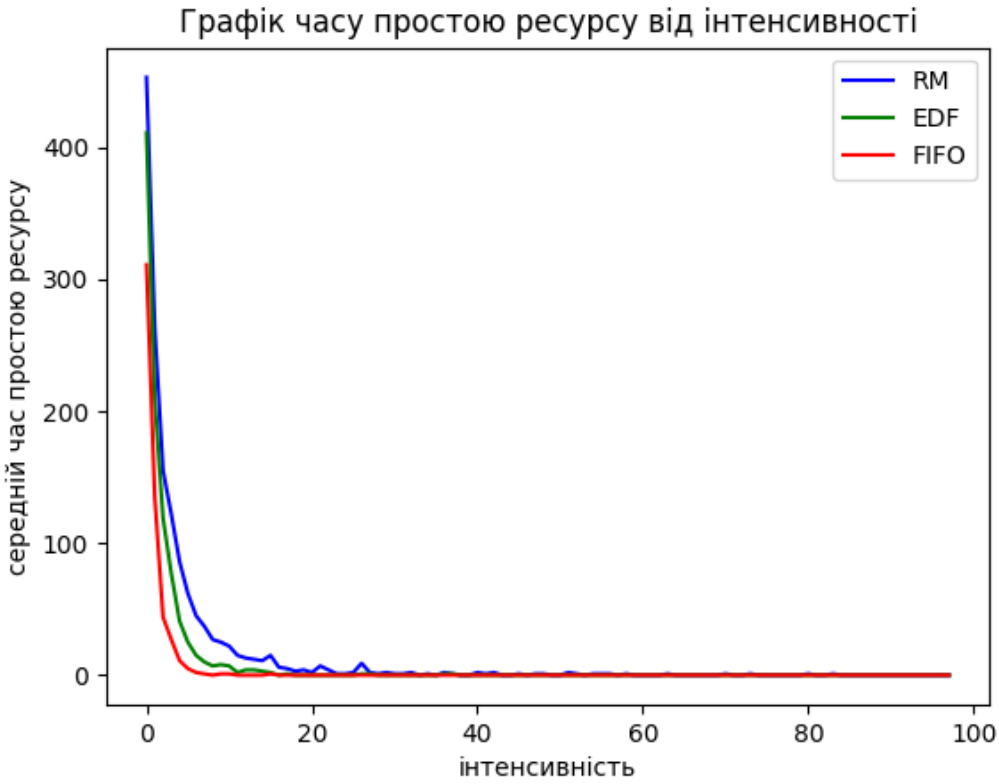
```
            def GetDeadline(self):
                return self.deadline

            def WorkedOn(self):
                self.executionTime = self.executionTime - 1

            def Wait(self):
                self.waitTime = self.waitTime + 1

            def GetExecutionTime(self):
                return self.executionTime

            def GetCreationTime(self):
                return self.creationTime
import matplotlib.pyplot as plt
import random
import Erlang as el
import Task
import SMO
import numpy

a = []
lam = 1
k = 2
E = el.ErlangDistribution(k, lam)


def GenerateQ():
    Q = []
    i = 0
    time = GenerateTime()
    while (i < 10000):
        i += E.GenerateNextInternal() * 8
        t = Task.Task(i, time)
        Q.append(t)
    return Q


def GenerateTime():
    rnd = random.random()
    if rnd < 0.3:
        return random.randrange(7) + 40  # Rxy
    elif 0.3 <= rnd < 0.6:
        return random.randrange(5) + 40  # Rxx
    elif 0.6 <= rnd < 0.8:
        return random.randrange(3) + 40  # Dx
    else:
        return random.randrange(2) + 40  # Mx
```

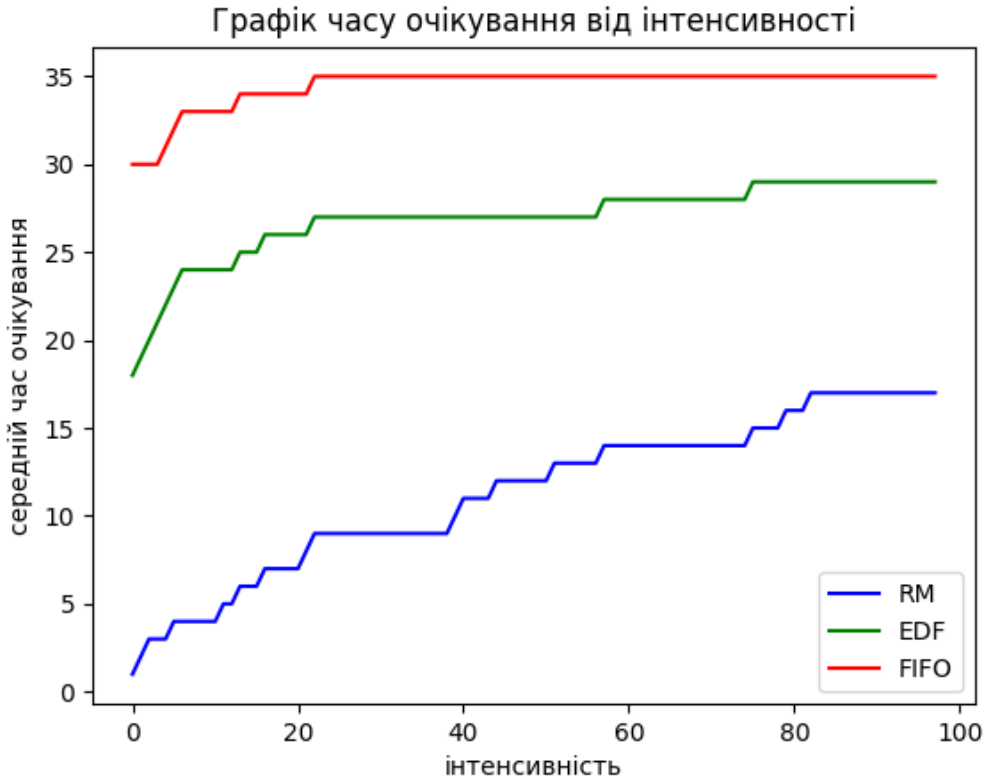
```
def findAnswers(isRM, isEDF):
    SMOs = []
    Tw = []
    Tww = []
    Tn = []
    faults = []
    faultschange = []
    Tnchange = []
    Twchange = []
    FullWaitTime = []
    for i in range(len(Queue)):
        SMOs.append(SMO.SMO(Queue[i]))
    for i in SMOs:
        i.Work(isRM, isEDF)
        faults.append(i.GetFaults()[:])
        Tw.append(i.GetWaitTimes()[:])
        Tn.append(i.GetProcessorFreeTime()[:])
    for i in range(len(SMOs)):
        faultschange.append(faults[i][9999])
        Tnchange.append(Tn[i][9999])
        Twchange.append(Tw[i][9999])
    # for o in range(len(SMOs)):
    #     time = 0.0
    #     for i in range(10000):
    #         time += Tw[o][i]
    #     Tww.append(time)
    # for i in range(len(SMOs)):
    #     Tww.append(FullWaitTime[i])
    # res = [[x/100 for x in faultschange[:]], Tnchange[:], Tww[:]]
    res = [[x / 100 for x in Tnchange[:]], faultschange[:], Twchange[:]]
    return res


if __name__ == "__main__":
    Queue = []
    for lam in numpy.arange(1, 50, 0.5):
        E.ChangeLambda(lam)
        temp = GenerateQ()
        Queue.append(temp)
    resRM = findAnswers(True, False)
    resEDF = findAnswers(False, True)
    resFIFO = findAnswers(False, False)
    l = len(resRM[0])
    t = [x for x in range(l)]

    plt.plot(t, resRM[0], 'b', t, resEDF[0], 'g', t, resFIFO[0], 'r')
```

```
plt.legend(['RM', 'EDF', 'FIFO'])
plt.title("Графік частки відкинутих задач від інтенсивності")
plt.ylabel('% відкинутих задач')
plt.xlabel('інтенсивність')
plt.show()

plt.plot(t, resRM[1], 'b', t, resEDF[1], 'g', t, resFIFO[1], 'r')
plt.legend(['RM', 'EDF', 'FIFO'])
plt.title("Графік часу простою ресурсу від інтенсивності")
plt.xlabel('інтенсивність')
plt.ylabel('середній час простою ресурсу')
plt.show()

plt.plot(t, resRM[2], 'b', t, resEDF[2], 'g', t, resFIFO[2], 'r')
plt.legend(['RM', 'EDF', 'FIFO'])
plt.title("Графік часу очікування від інтенсивності")
plt.xlabel('інтенсивність')
plt.ylabel('середній час очікування')
plt.show()
```

ІЛЮСТРАЦІЇ




Графік часу простою ресурсу від інтенсивності
RM
EDF
FIFO
середній час простою ресурсу
0
100
200
300
400
0
20
40
60
80
100
інтенсивність


Графік часу очікування від інтенсивності
середній час очікування
0
5
10
15
20
25
30
35
0
20
40
60
80
100
інтенсивність
RM
EDF
FIFO

ВИКОРИСТАНА ЛІТЕРАТУРА

1. Asynchronous Flow [Електронний ресурс] – Режим доступу до ресурсу: https://kotlinlang.org/docs/flow.html.

2. Rate-monotonic scheduling [Електронний ресурс] // Wikipedia, The Free Encyclopedia – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Rate-monotonic_scheduling.

3. Earliest deadline first scheduling [Електронний ресурс] // Wikipedia, The Free Encyclopedia – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Earliest_deadline_first_scheduling