ИУ5-32Б,

Колесников Никита

Отчет по домашнему заданию

Цель: моделирования СМО и сбор статистики с использованием библиотеки Simpy

Текст программы

```
import simpy
import random
import sys
#Мы не работаем с моменттом поступления заявок до входа в систему, а только
по факту. Момент создания заявки

max_queue_size = 1000000  # размер очереди
max_wait_time = 100000000  # максимальное время ожидания в очереди
prob_remork = 0.05  # вероятность ухода на доп обслуживания
sim_time = 20160  # время симуляции
count_request = 30  # количество запросов
count_phases = 3  # Количество фаз
count_prioritys = 3  # Количество приоритетов
processing_time = [4000, 3920, 2000]  # Время обработки запроса на каждой
фазе
count_channels = [3, 4, 2]  # Количество каналов в каждой фазе
prob_start_phase = [0.5, 0.4, 0.1]  # Начальная вероятность попадания на фазу
prob_priority = [0.2, 0.5, 0.3]  # Вероятность рапспределения приоритетов

l=count_request/sim_time
l_f=l*(1+prob_remork)
#Класс запросов
class Request(object):
    def __init__(self, env, name, number):
        self.env=env
        self.name=name
        self.number=number
        self.phase=self.set_phase()
        self.priority=self.set_priority()
        self.succes=False # Флаг, указывающий, что запрос был выполнен
        #self.time_success=0
        self.abandoned = False  # Флаг, указывающий, что запрос покинул
очередь из-за таймаута
        self.queue=False
        self.waiting=0
        self.reentries=0#Число доработок

    def set_phase(self):
        return random.choices(range(1, count_phases + 1),
weights=prob_start_phase)[0]
    def set_priority(self):
        return random.choices(range(1, count_phases + 1),
weights=prob_priority)[0]

class Channel(object):
    def __init__(self, env):
        self.env=env
        self.work time=0
        self.state = False
        self.in_work = {i: 0 for i in range(sim_time + 1)}  # всех запросов
        self.req=[]
        self.downtime_interval=[]#Интервалы простоя
```

```
        self.worktime_interval = []  # Интервалы работы
        self.using=0

class Phase(simpy.Resource):
    def __init__(self, env,service_time,count_channels, *args, **kwargs):
        super().__init__(env, *args, **kwargs)
        self.env = env
        self.service_times =service_time  # Времена обслуживания в этой фазе
        self.count_channels = count_channels

        self.work_time=0 #время работы
        self.count_requests = 0  # количество полученных запросов
        self.count_ser_req=0 # Количесвто обработанных запросов
        self.count_rej_req = 0  # количесвто отклоненных запросов
        self.count_que_req=0 #  количество запросов попавших в очередь

        self.P0=1#вероятность простоя фазы
        self.P_och=0#вероятность обрахования очереди
        self.W=0#среднее время, которое заявка проводит в фазе
        self.A=float('inf')#абсолютная пропускная способность
        self.t_pr=0#tпр – среднее время простоя каналов
        self.n_z=0#среднее число занятых каналов;
        self.L_rej=0
        self.L_req=0
        self.L_time=0
        self.L_que=0
        self.p=0
        # Словари моментов времени для:
        self.channels={i: Channel(env) for i in range(1,count_channels+1)}

        self.requests_in_time = {i:0 for i in range(sim_time+1)}#всех
запросов
        self.requests_in_que= {i:0 for i in range(sim_time+1)}#запросов в
очереди
        self.requests_in_rej = {i:0 for i in range(sim_time+1)}#отклоненных
запросов

        self.work_time_proc={i:0 for i in range(1,count_request+1)} # время
работы фазы
        self.que_time_proc = {i: 0 for i in range(1, count_request + 1)}

        self.all_time_work = {i: 0 for i in range(sim_time + 1)}  # занятость
по времени всех каналов
        self.any_time_work= {} # занятость по времени хотя бы 1 канала

        self.downtime_interval = []  # Интервалы простоя общий
        self.worktime_interval = []  # Интервалы работы общий
        self.count_work_channels={i:0 for i in range(sim_time + 1)}
        self.count_down_channels = {i: 0 for i in range(sim_time + 1)}

def get_all_time_proc():
    for i in range(count_phases):
        for j in range(1, (phases[i].count_channels)+1):
            x=0
            y=0
            for z in range(sim time):
                phases[i].all_time_work[z]+=phases[i].channels[j].in_work[z]
                if phases[i].channels[j].in_work[z]==0:
                    x+=1
                elif x!=0:
                    phases[i].channels[j].downtime_interval.append(x)
                    x=0

                if phases[i].channels[j].in_work[z]==1:
```

```
                    y+=1
                elif y!=0:
                    phases[i].channels[j].worktime_interval.append(y)
                    y=0

                if phases[i].channels[j].in_work[z]==1:
                    phases[i].count_work_channels[z]+=1
                else:
                    phases[i].count_down_channels[z] += 1
                for x in range(len(phases[i].channels[j].req)):

phases[i].channels[j].using=(phases[i].channels[j].req[x][0]*phases[i].channe
ls[j].req[x][1])
            if x!=0:
                phases[i].channels[j].downtime_interval.append(x)
            if y!=0:
                phases[i].channels[j].worktime_interval.append(y)

def print_result():
    print("")
    if len(abandoned_requests) > 0:
        print("Отброшенные запросы:")
        for req in abandoned_requests:
            print(req.name)
    if len(success_requests) > 0:
        print("Обработанные запросы: ")
        for req in success_requests.keys():
            if success_requests[req]!=0:
                print("Процесс",req)
    else:
        print("Обработанных запросов нет")

def change_requests_in_rej():
    for i in range(count_phases):
        #print("проверка фазы ", i)
        if phases[i].count_rej_req > 0:
            #print(phases[i].requests_in_rej.items())
            for j in range(1, sim_time + 1):
                #print(phases[i].requests_in_rej[j-1])
                if phases[i].requests_in_rej[j - 1] > 0:
                    if phases[i].requests_in_rej[j] == 0:
                        phases[i].requests_in_rej[j] =
phases[i].requests_in_rej[j - 1]
                    else:
                        #print(phases[i].requests_in_rej[j])
                        phases[i].requests_in_rej[j] +=
phases[i].requests_in_rej[j - 1]
            #print(phases[i].requests_in_rej.items())
            if phases[i].requests_in_rej[sim_time] !=
phases[i].count_rej_req:
                print("В изменении несоответствия!!!!!!!!!")

print(phases[i].requests_in_rej[sim_time],phases[i].requests_in_rej.items())

def copy_dict(dict):
    dictionari={i:j for i,j in dict.items()}
    return dictionari

def del_keys(dict):
    del_keys = [i for i in dict.keys() if dict[i] == 0]
    for i in del_keys:
        del dict[i]
    return dict
```

```
def dict_for_L(first_dict):
    dictionary=copy_dict(first_dict)
    dict=del_keys(dictionary)
    array_L = []
    if len(dict)!=0:
        list_val=(dict.values())
        set_list_L=set(list_val)
        for i in set_list_L:
            time=0
            for x in dict.values():
                if x==i:
                    time+=1
            array=[i,time]
            array_L.append(array)
    return array_L

def L(array_L):
    L = 0
    for i in array_L:
        L += (i[0] * i[1])
    L/=sim_time
    return L

def statistics_system(file=sys.stdout):
    if count_request==0:
        print("Система не получила ни одного запроса", file=file)
    else:
        P0=1
        P_och1=1
        L_req = 0
        L_que = 0
        R=0
        W=0
        A=float('inf')
        t_pr=0
        n_z=0
        del_phase=0
        for i in range(count_phases):
            if phases[i].count_requests==0:
                del_phase+=1
            else:
                #if phases[i].P0!=0:
                P0*=phases[i].P0
                P_och1*=(1-phases[i].P_och)
                L_req+=phases[i].L_req
                L_que+=phases[i].L_que
                R+=phases[i].p
                W+=phases[i].W
                A=min(A,phases[i].A)
                t_pr+=phases[i].t_pr
                n_z+=phases[i].n_z
        L=L_req+L_que
        t_pr/=(count_phases-del_phase)
        print("P0 - вероятность простоя системы: ",P0,file=file)
        print("Роч - вероятность образования очереди: ",1-P_och1, file=file)
        print("L - среднее число заявок в системе: ",L, file=file)
        print("R - суммарная загрузка всех узлов СМО, характеризующая среднее
число параллельно работающих узлов сети: ",R,file=file)
        print("W - среднее время, которое заявка проводит в системе от
момента поступления до момента завершения обслуживания: ",W,file=file)
        print("A - абсолютная пропускная способность: ",A,file=file)
        print("Q - относительная пропускная способность/коэффициент
пропускной способности: ",l_f,file=file)
        print("tпр - среднее время простоя системы: ", t_pr, file=file)
```

```
        print("nз - среднее число занятых каналов в системе: ", n_z,
file=file)

def statistics_phase(phases,file=sys.stdout):
    print("Статистика по фазам\n", file=file)
    for i in range(count_phases):
        print("Фаза №%d:"%(i+1) ,file=file)
        if phases[i].count_requests==0:
            print("Не было получено запросов на обработку", file=file)
        else:
            P_rej=phases[i].count_rej_req/phases[i].count_requests
            print("Ротк - вероятность отказа в обслуживании: ", P_rej,
file=file)
            print("Робсл - вероятность обслуживания: ", 1-P_rej, file=file)
            print("Рм.обсл - вероятность обслуживания без ожидания в очереди:
", (phases[i].count_requests - phases[i].count_que_req) /
phases[i].count_requests, file=file)
            phases[i].P_och=phases[i].count_que_req /
phases[i].count_requests
            print("Роч - вероятность образования очереди: ", phases[i].P_och,
file=file)

            list_L_time = dict_for_L(phases[i].requests_in_time)
            phases[i].L_time=L(list_L_time)
            print("L - среднее число заявок: ", phases[i].L_time,file=file)

            list_L_que = dict_for_L(phases[i].requests_in_que)
            phases[i].L_que=L(list_L_que)
            print("Lоч - среднее число заявок в очереди: ", phases[i].L_que,
file=file)

            list_L_rej = dict_for_L(phases[i].requests_in_rej)
            phases[i].L_rej=L(list_L_rej)
            print("Lотк - среднее число отклоненных заявок: ",
phases[i].L_rej, file=file)
            phases[i].L_req=phases[i].L_time-phases[i].L_que

            phases[i].work_time_proc=del_keys(phases[i].work_time_proc)
            phases[i].W=
(sum(phases[i].work_time_proc.values())/len(phases[i].work_time_proc))
            print("W - среднее время, которое заявка проводит в фазе от
момента поступления до момента выхода из нее: ",phases[i].W,file=file)
            phases[i].que_time_proc = del_keys(phases[i].que_time_proc)
            if len(phases[i].que_time_proc)==0:
                print("Wоч - среднее время ожидания обслуживания (ожидания в
очереди): 0",file=file)
            else:
                print("Wоч - среднее время ожидания обслуживания (ожидания в
очереди): ",
(sum(phases[i].que_time_proc.values())/len(phases[i].que_time_proc)),file=fil
e)
            print("Q - относительная пропускная способность/коэффициент
пропускной способности (отношение среднего числа заявок, обслуживаемых фазой
в единицу времени, к среднему числу поступивших за это же время заявок): ",
                  phases[i].count_ser_req/phases[i].count_requests,
file=file)

            phases[i].any_time_work = copy_dict(phases[i].all_time_work)
            del_keys(phases[i].any_time_work)
            phases[i].work_time=len(phases[i].any_time_work)
            for j in range(len(phases[i].all_time_work)):
                #print(phases[i].all_time_work[j])
                if phases[i].all_time_work[j]!=phases[i].count_channels:
                    del phases[i].all_time_work[j]
```

```
            print("U (ρ) - коэффициент использования (доля времени, в течение
которого все обслуживающие каналы заняты): ",
len(phases[i].all_time_work)/phases[i].work_time, file=file)
            X=0
            for j in range(1, phases[i].count_channels+1):
                X+=((phases[i].work_time-
phases[i].channels[j].work_time)/phases[i].work_time)
            phases[i].P0=X/phases[i].count_channels
            print("P0 - вероятность простоя фазы: ",phases[i].P0,file=file)
            phases[i].A=phases[i].count_ser_req/len(phases[i].any_time_work)
            print("A - абсолютная пропускная способность (среднее число
заявок, которое сможет обслужить фаза в единицу времени):
",phases[i].A,file=file)
            print("η - коэффициент эффективности (отношение полезного времени
работы фазы к общему времени работы): ",
len(phases[i].any_time_work)/sim_time,file=file)

            for j in range(1,phases[i].count_channels+1):
                if len(phases[i].channels[j].downtime_interval)!=0:

phases[i].downtime_interval.append(sum(phases[i].channels[j].downtime_interva
l) / len(phases[i].channels[j].downtime_interval))
                else:
                    phases[i].downtime_interval.append(0)
                if len(phases[i].channels[j].worktime_interval)!=0:

phases[i].worktime_interval.append(sum(phases[i].channels[j].worktime_interva
l) / len(phases[i].channels[j].worktime_interval))
                else:
                    phases[i].worktime_interval.append(0)
            phases[i].t_pr=sum(phases[i].downtime_interval) /
len(phases[i].downtime_interval)
            print("tпр - среднее время простоя: ",phases[i].t_pr,file=file)
            print("tобсл - среднее время обслуживания:
",sum(phases[i].worktime_interval) / len(phases[i].worktime_interval),
file=file)

            list_n_work=dict_for_L(phases[i].count_work_channels)
            phases[i].n_z=L(list_n_work)
            print("nз - среднее число занятых каналов: ",phases[i].n_z,
file=file)
            list_n_down=dict_for_L(phases[i].count_down_channels)
            print("nсв - среднее число свободных каналов: ", L(list_n_down),
file=file)
            print("",file=file)
            p1=0
            for j in range(1,phases[i].count_channels+1):
                p1+=phases[i].channels[j].using
            phases[i].p=l_f*p1

def check_completion(env, end_event, abandoned_requests):
    yield env.timeout(sim_time + 1)
    if len(success_requests) != count_request:
        print("Не все процессы были обработанны.")
    end_event.succeed()

#Моделирование запроса
def request_generator(env, count_requests,phases, abandoned_requests):
    for i in range(1, count_request+1):
        req=Request(env, 'Процесс %d' % i,i)
        requests.append(req)
    requests.sort(key=lambda obj: obj.priority)
    for req in requests:
        env.process(process_request(env, req, phases, abandoned_requests))
```

```
#Моделирование системы
def process_request(env, req, phases, abandoned_requests):
        while not(req.succes) or req.abandoned:
            print('%s поступил на обработку на фазу %s в
%s'%(req.name,req.phase ,env.now))
            # if env.now>0 and phases[req.phase-
1].requests_in_rej[((env.now)-1)]!=0:
            #     print(env.now, phases[req.phase -
1].requests_in_rej.items())
            #     phases[req.phase -
1].requests_in_rej[env.now]=phases[req.phase - 1].requests_in_rej[env.now -
1]#отклоненные заявки

            phases[req.phase-1].count_requests+=1
            if env.now+phases[req.phase-1].service_times>sim_time:
                print('%s не принят в обработку в %s, так как не успеет
выполниться' % (req.name, env.now))
                #phases[req.phase - 1].requests_in_time[env.now] -= 1
проверить
                phases[req.phase - 1].count_rej_req+=1
                #print(env.now, phases[req.phase -
1].requests_in_rej.items())
                phases[req.phase-1].requests_in_rej[env.now]+=1
                #print(env.now,phases[req.phase-1].requests_in_rej.items())
                req.abandoned=True
                break
            else:
                queue_length = len(phases[req.phase - 1].queue)
                if queue_length >= max_queue_size and req.reentries==0:  #
Проверка длины очереди
                    print("%s отклонен: очередь на фазе %s заполнена в
%s"%(req.name, req.phase, env.now))
                    #phases[req.phase - 1].requests_in_time[env.now] -= 1
                    phases[req.phase - 1].requests_in_rej[env.now] += 1
                    phases[req.phase - 1].count_rej_req += 1
                    req.abandoned = True
                    break
                with phases[req.phase-1].request() as req_proc:# работа с
запросом
                    timeout_event = env.timeout(max_wait_time)  # Создать
событие таймаута ожидания
                    print('%s запрашивает доступ к ресурсу на фазе %s в %s' %
(req.name, req.phase, env.now))
                    time_come=env.now
                    phases[req.phase - 1].requests_in_time[env.now] += 1
                    yield req_proc | timeout_event  # Ждать ресурс ИЛИ
таймаут
                    time_waiting=env.now-time_come
                    if time_waiting>0:
                        req.waiting+=time_waiting
                        phases[req.phase-
1].que_time_proc[req.number]+=time_waiting
                        for i in range(time_come+1, env.now+1):
                            phases[req.phase - 1].requests_in_time[i]+=1
                            phases[req.phase - 1].requests_in_que[i] += 1
                        if not(req.queue):#????
                            phases[req.phase-1].count_que_req+=1
                            req.queue=True

                    if env.now + phases[req.phase-1].service_times >
sim_time:
                        print('%s не принят в обработку в %s, так как не
успеет выполнится из за ожидания в очереди' % (req.name, env.now))
```

```
                        req.abandoned=True
                        phases[req.phase-1].requests_in_time[sim_time] -= 1
                        phases[req.phase - 1].requests_in_rej[env.now] += 1
                        phases[req.phase - 1].count_rej_req += 1
                       #print(env.now, phases[req.phase -
1].requests_in_rej.items())
                        req.abandoned = True
                        break

                elif req_proc.triggered:  # Ресурс получен
                        print('%s начал обработку на фазе %s в
%s'%(req.name,req.phase ,env.now))
                        time_start=env.now
                        # if not(any(phases[req.phase-1].channels[i].in_work
for i in range(1,(len(phases[req.phase-1].channels))+1))):
                        #     phases[req.phase-
1].work_time+=(time_start+phases[req.phase-1].service_times)
                        for i in range(1,(len(phases[req.phase-
1].channels))+1):
                            #print(phases[req.phase-1].channels[i].state)
                            if not(phases[req.phase-1].channels[i].state):
                                phases[req.phase - 1].channels[i].state=True
                                yield env.timeout(phases[req.phase-
1].service_times)#передача запроса в канал
                                phases[req.phase-
1].channels[i].work_time+=(env.now-time_start)
                                list_rej=[]
                                list_rej.append(req.priority)
                                list_rej.append(env.now-time_start)
                                phases[req.phase-
1].channels[i].req.append(list_rej)
                                #print(phases[req.phase-
1].channels[i].work_time)
                                phases[req.phase-1].channels[i].state=False
                                for j in range(time_start, env.now):
                                    phases[req.phase -
1].channels[i].in_work[j] += 1
                                break
                        for i in range(time_start+1, env.now):
                            phases[req.phase - 1].requests_in_time[i] += 1
                        print('%s конец работы на фазе %s в %s' %
(req.name,req.phase, env.now))
                        phases[req.phase-1].count_ser_req+=1
                        phases[req.phase - 1].work_time_proc[req.number] +=
(env.now - time_come)

                elif timeout_event.triggered:  # Сработало событие
таймаута
                        print("%s покидает систему, недождавшись очереди на
фазе %s в %s" %(req.name,req.phase,env.now))
                        #phases.requests_in_time[sim_time] -= 1 Надо
проверить
                        phases[req.phase - 1].requests_in_rej[env.now] += 1
                        phases[req.phase - 1].count_rej_req += 1
                        req.abandoned = True
                        break  # Прервать цикл по фазам

            #Анализ обработанного запроса
            if not(req.abandoned) and random.random() < prob_remork:
                print("%s уходит на дообслуживание на фазе %s в %s"
%(req.name,req.phase,env.now))#Должен освободить канала
                req.phase -= 1
                phases[req.phase - 1].count_requests -= 1
                phases[req.phase - 1].count_ser_req -= 1
```

```
                    req.reentries += 1  # Увеличиваем счетчик повторных
входов
                if req.phase!=count_phases:
                    req.phase += 1  # Перейти к следующей фазе
                else:
                    req.succes = True

        if req.queue:
            que_requests[req.number]=req.waiting# Добавляем запрос который
попал в очередь

        if req.abandoned:
            abandoned_requests.append(req)  # Добавляем отброшенный запрос в
список

        if not(req.abandoned):
            print('%s полностью завершил обработку в %s' % (req.name,
env.now))
            success_requests[req.number]=env.now
            #req.time_success=env.now

        if all(success_requests.values()) != 0:
            print("Все процессы обработаны. Время обработки %d" % (env.now))

env=simpy.Environment()
end_event = env.event()
abandoned_requests= [] # Отслеживаем отброшенные запросы

requests = [] # Список активных запросов
success_requests={i:0 for i in range(1,count_request+1)} # Словарь
обработанных запросов
que_requests={i:0 for i in range(1,count_request+1)} #Словарь запросов
попавшие в ожидание

phases=[Phase(env,processing_time[i],count_channels[i],capacity=count_channel
s[i]) for i in range(count_phases)]
request_generator(env,count_request, phases, abandoned_requests)

env.process(check_completion(env, end_event, abandoned_requests))
env.run(until=end_event)

print_result()

change_requests_in_rej()

get_all_time_proc()

with open('phases_statistics.txt', 'w', encoding='utf-8') as f:
    statistics_phase(phases, file=f)

with open('simulation_statistics.txt', 'w', encoding='utf-8') as f:
    statistics_system(file=f)
```

Анализ результатов