ИУ5-32Б,

Колесников Никита

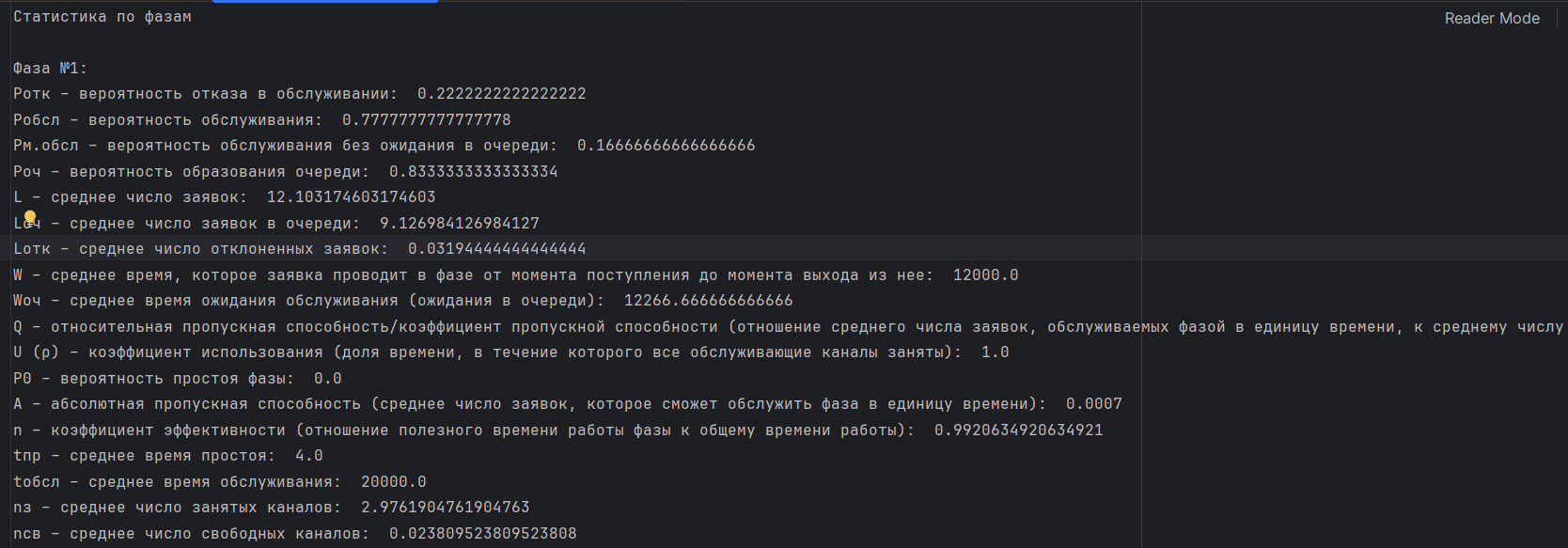
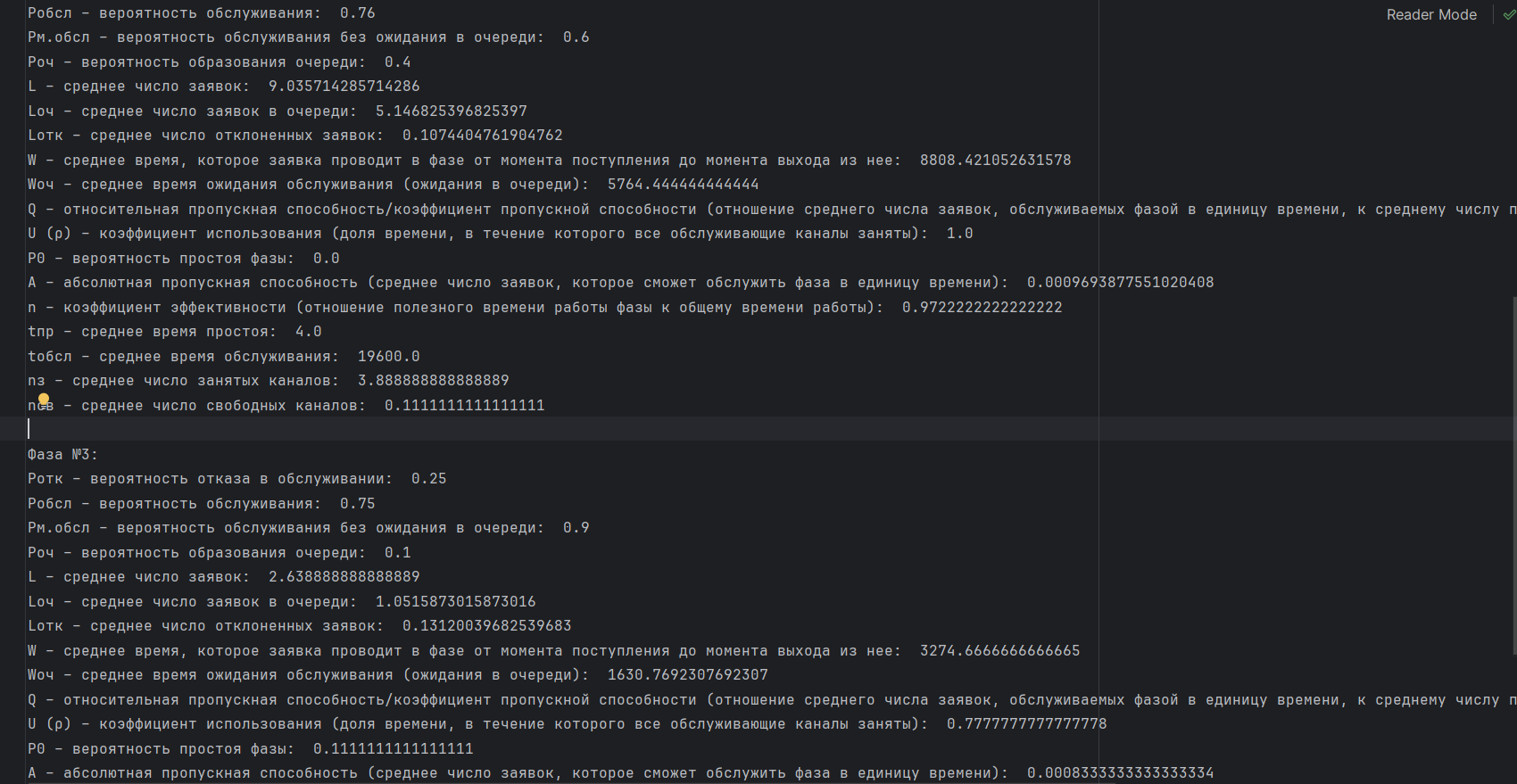
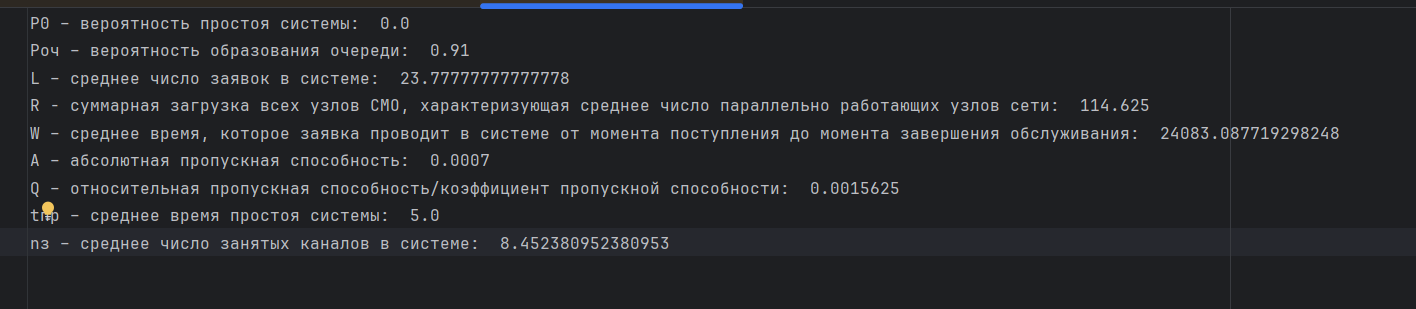
**Отчет по домашнему заданию**

**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 рапсределения 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].channels[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("Pоч – 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("Pотк – 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("Pоч – 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=file)  
 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("А – 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\_interval) / 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\_interval) / 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\_channels[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**

**** ****