Національний Технічиний Університет України

«Київський Політехнічний Інститут імені Ігоря Сікорського»

Інститут Прикладного Системного Аналізу

Кафедра Системного Проектування

Паралельні обчислення

Лабораторна робота №2

Роботу виконала:

Желєзнова В.С.

Група: ДА­-81

Перевірив: Яременко В. С.

Київ – 2021

ЗМІСТ

[Мета роботи 3](#_Toc67578172)

[Завдання 3](#_Toc67578173)

[1. Лістинг програми мовою Python 4](#_Toc67578174)

[2. Порівняння реалізацій 7](#_Toc67578175)

[Висновки 10](#_Toc67578176)

Мета роботи

Метою роботи є набуття навичок роботи з потоками при програмуванні на мові Java або будь-якій іншій.

Завдання

Напишіть програму моделювання з використанням на мові Java по одному із наведених нижче варіантів.

**Варіант 10.**

Програма моделює обслуговування одного потоку процесів двома центральними процесорами комп’ютера з двома чергами. Якщо черговий процес генерується в момент, коли будь-який із процесорів вільний, процес надходить на обробку у вільний процесор, інакше процес поміщається в ту з черг, розмір якої менший. Визначити максимальний розмір для кожної із черг для двох однакових процесорів.

1. Лістинг програми мовою Python

import threading  
import random  
from time import sleep  
from time import time  
from queue import Queue  
  
  
class CPUQueue:  
  
 def \_\_init\_\_(self, queue\_id):  
 self.queue = Queue()  
 self.sync = threading.Condition()  
 self.max\_len = 0  
 self.queue\_id = queue\_id  
  
 def is\_empty(self):  
 return self.queue.empty()  
  
 def put(self, element):  
  
 try:  
 self.sync.acquire()  
  
 self.queue.put(element)  
 print(f'Element {element} is added to the queue{self.queue\_id}.')  
  
 self.print\_info()  
 self.max\_len = self.queue.qsize() if self.queue.qsize() > self.max\_len else self.max\_len  
 finally:  
 self.sync.notify()  
 self.sync.release()  
  
 def get(self):  
  
 try:  
 self.sync.acquire()  
 if self.is\_empty():  
 print('Queue is EMPTY. Cannot get elements. Waiting...')  
 self.sync.wait()  
  
 temp = self.queue.get()  
 print(f'Element {temp} is removed from the queue{self.queue\_id}.')  
  
 self.print\_info()  
 return temp  
 finally:  
 self.sync.notify()  
 self.sync.release()  
  
 def print\_info(self):  
 print(f'QUEUE{self.queue\_id}: {self.queue.queue}')  
  
 def get\_size(self):  
 return self.queue.qsize()  
  
  
class ProcessGenerator(threading.Thread):  
  
 def \_\_init\_\_(self, queue1, queue2, num, C1, C2):  
 threading.Thread.\_\_init\_\_(self)  
 self.queue1 = queue1  
 self.queue2 = queue2  
 self.num = num  
 self.C1 = C1  
 self.C2 = C2  
 self.i = 1  
  
 def run(self):  
 for i in range(1, self.num + 1):  
 delay = random.randint(1, 20) / 100  
 sleep(delay)  
 print(f'GENERATOR: Process {i} generated with delay {delay}\n')  
 if not self.C1.is\_busy():  
 print(f'TO PROCESSOR {self.C1.cpu\_id} ->')  
 self.C1.process(i)  
 elif not self.C2.is\_busy():  
 print(f'TO PROCESSOR {self.C2.cpu\_id} ->')  
 self.C2.process(i)  
 elif self.C1.is\_busy and self.C2.is\_busy:  
 if queue1.get\_size() <= queue2.get\_size():  
 self.queue1.put(i)  
 else:  
 self.queue2.put(i)  
  
 # if not self.C1.is\_busy():  
 # print(f'TO PROCESSOR {self.C1.cpu\_id} ->')  
 # self.C1.process(i)  
 # elif not self.C2.is\_busy():  
 # print(f'TO PROCESSOR {self.C2.cpu\_id}->')  
 # self.C2.process(i)  
 # elif queue1.get\_size() <= queue2.get\_size():  
 # self.queue1.put(i)  
 # else:  
 # self.queue2.put(i)  
  
  
class CPU(threading.Thread):  
  
 def \_\_init\_\_(self, queue, cpu\_id):  
 threading.Thread.\_\_init\_\_(self)  
 self.queue = queue  
 self.busy = False  
 self.flow\_time = 0  
 self.id = 0  
 self.cpu\_id = cpu\_id  
 self.end\_time = 0  
  
 def is\_busy(self):  
 return self.busy  
  
 def run(self):  
 end\_time = time()  
 start\_time = time()  
 while True:  
 if self.busy:  
 end\_time = time()  
 print(f'Is processing without queue...')  
 processing\_time = random.randint(10, 40) / 100  
 sleep(processing\_time)  
 print(f'Without QUEUE: Process {self.id} processed in time {processing\_time}.\n')  
 self.busy = False  
 end\_time = time()  
  
 if self.busy == False and self.queue.is\_empty() == False:  
 end\_time = time()  
 self.busy = True  
 processing\_time = random.randint(10, 40) / 100  
 self.id = self.queue.get()  
 sleep(processing\_time)  
 print(f'CPU{self.cpu\_id}: Process {self.id} processed in time {processing\_time}.\n')  
 self.busy = False  
 end\_time = time()  
  
 if time() - end\_time > 1:  
 break  
  
 def process(self, id):  
 self.busy = True  
 self.id = id  
  
  
num\_of\_processes = 50  
  
queue1 = CPUQueue(1)  
queue2 = CPUQueue(2)  
C1 = CPU(queue1, 1)  
C2 = CPU(queue2, 2)  
generator = ProcessGenerator(queue1, queue2, num\_of\_processes, C1, C2)  
  
print(f'Will be generated {num\_of\_processes} processes\n')  
generator.start()  
C1.start()  
C2.start()  
  
generator.join()  
C1.join()  
C2.join()  
  
print(f'Max queue1 size = {queue1.max\_len}.')  
print(f'Max queue2 size = {queue2.max\_len}.')

Лістинг 1. Main.py

1. Порівняння реалізацій

При значеннях:

Num of processes = 10

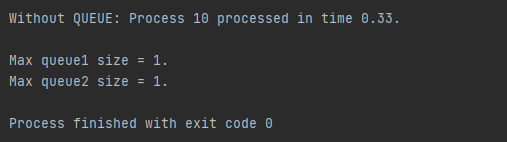


Рисунок 2.1 Максимальна довжина черги при 10 процесах

При значеннях:

Num of processes = 50

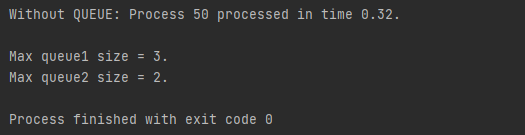


Рисунок 2.2 Максимальна довжина черги при 50 процесах

При значеннях:

Num of processes = 100

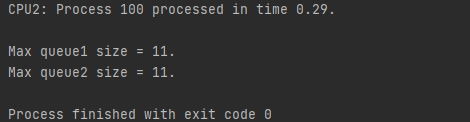


Рисунок 2.3 Максимальна довжина черги при 100 процесах

Висновки

У ході виконання даної лабораторної роботи було розроблено програму, що реалізує умову завдання.

Можу сказати, що створена програма працює, було реалізовано синхронізацію та, насамперед, використання даних трьома потоками.