МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

НИЖЕГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ

УНИВЕРСИТЕТ им. Р.Е.АЛЕКСЕЕВА

Институт радиоэлектроники и информационных технологий

Кафедра вычислительные системы и технологии

Проект

Моделирование работы синхронных схем

Вариант №8

ПРОЕКТ

по дисциплине

Параллельные вычисления

РУКОВОДИТЕЛЬ:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Мартынов Д.С.

СТУДЕНТЫ:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Сапожников В.О.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Аверьянова А.А.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Папанов Р.В.

19-В-1

Работа защищена «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_

С оценкой \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Нижний Новгород 2021

1. **Цель работы**

Построить модель и разработать программную модель для следующей задачи:

8. Моделирование работы распределенного файлового хранилища.

Разработать программу, имитирующую процесс чтения приборных данных и записи логов в несколько «журналов», физически представленных в виде отдельных текстовых файлов. Данные журналов событий читаются в рамках отдельной задачи и отображаются в виде отчета в отдельном окне приложения.

В рамках данной системы рекомендуется выделить подсистемы «писателей» и «читателей». «Писатели» выступают в роли генераторов событий, моменты возникновения которых определяются при помощи настраиваемых генераторов псевдослучайных чисел. «Читатели» связаны с текстовыми файлами, в которые осуществляется запись информации о сгенерированных системных событиях.

1. **Анализ задачи**

Пусть имеется N приборов, которые могу генерировать M событий. Генерация событий будет происходить при помощи генератора псевдослучайных чисел (ГПЧ).

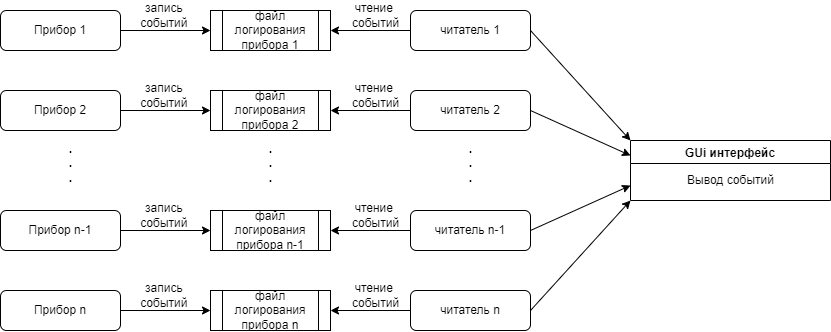
Выделим 2 отдельных события, сообщающих о старте работы прибора и удачном завершении работы. На данные события можно повлиять вручную посредством графического пользовательского интерфейса. Пусть так же будет еще одно специальное событие, которое сообщает о завершении работы прибора с ошибкой, оно так же будет генерироваться ГПЧ, но с меньшей вероятностью.

О каждом событии идет запись в отдельный файл логирования для данного прибора. С этим же файлом связан читатель, который сообщает о новом событии через GUI интерфейс.

При данной модели узким местом программы является файл логирования прибора. Для решения данной, когда писатель или читатель обращается к файлу логирования, файл необходимо блокировать как разделяемый ресурс. Для блокирования потока работы с файлом подойдет обычный монитор, однако, чтобы ограничить максимальное возможное кол-во потоков программы мы используем семафор с одинаковым счетчиков для чтецов и писателей.

Так же узким местом является поток вывода GUI интерфейса, но поскольку операции чтения из файла и вывод производятся друг за другом, то они обе находятся под семафором в данный момент.

1. **Диаграмма потока данных**



1. **Программная реализация**

Программная модель была реализована средствами языка программирования Python. Для распараллеливания потоков используется модуль threading.

При реализации данной модели было решено, что максимальное кол-во приборов и читателей будет ограничено 10.

Для создания GUI использовалась библиотека Tkinter.

При помощи интерфейса мы можем включить/выключить любой из 10 приборов или все сразу. Каждый из приборов и читателей работает в отдельном потоке, GUI обрабатывается в главном Main потоке.

Информация о событиях пишется в отдельный лог файл откуда считывается читателем и выводится внутри текстового поля разработанного интерфейса.

Код программной реализации разделен на 4 файла:

**Device.py** – содержит класс приборов.

**Logger.py** – класс логгер – “писатель”

**Reader.Py** – класс читателей

**main.py** – содержит точку входа в программу, а так же настройку и обработку GUI.

1. **Листинг программы**

**Device.py**

|  |
| --- |
| import random  import threading |
| import time |
| from threading import BoundedSemaphore |
|  |
| from Logger import Logger |
| from Reader import Reader |
|  |
|  |
| class Device: |
| """ |
| Класс приборов, которые будут проводить измерения |
| и регистрировать события в файлах при помощи логгеров |
| :see Logger.py: |
| """ |
|  |
| # Ограничение кол-во параллельно работающих приборов |
| # при помощи семафора |
| devicesPool: BoundedSemaphore = BoundedSemaphore(10) |
|  |
| # словарь событий |
| # {код события: словесное описание} |
| event: dict = { |
| 100: 'Завершение работы', |
| 1: 'Начал измерение', |
| 2: 'Проводит измерение', |
| 3: 'Анализирует данные', |
| 4: 'Фиксирует данные', |
| -1: 'Завершение с ошибкой', |
| -100: 'Завершение вручную' |
| } |
|  |
| def \_\_init\_\_(self, name, outStream): |
| """ |
| Конструтор с параметрами. |
| При создании прибор находится в состоянии покоя |
| Так же за прибором закрепляется логгер |
| :param name: имя прибора |
| """ |
| self.isWork = False |
| self.name = name.split("/")[-1] |
| self.logger = Logger(name) |
| self.text = outStream |
|  |
| def work(self): |
| """ |
| Метод для запуска в отдельном потоке. |
| Прибор переходит в состояние 'работы' и прибывает |
| в нем вплоть до завершения или завершения с ошибкой |
| :return: |
| """ |
| self.isWork = True |
|  |
| # Обращение к файлу - разделяемому ресурсу |
| # ограждается семафорами |
| self.devicesPool.acquire() |
| self.logger.system(self.name.ljust(10) + ": " + self.event[1]) |
| self.devicesPool.release() |
|  |
| # Так же запускаем поток с Читателем файла логгирования |
| # Такой подход позволяет работать с прибором и читать лог |
| # не зависимо от других приборов |
| Reader("Logs/" + self.name + ".log", self.text).run() |
| time.sleep(random.randint(1, 8)) |
|  |
| # Цикл имитации работы прибора |
| while self.isWork: |
| rand = random.randint(-10, 110) |
|  |
| # Обращение к файлу - разделяемому ресурсу |
| # ограждается семафорами |
| self.devicesPool.acquire() |
|  |
| if rand < 0: |
| self.logger.error(self.name.ljust(10) + ": " + self.event[-1]) |
| self.isWork = False |
|  |
| elif 0 < rand <= 33: |
| self.logger.debug(self.name.ljust(10) + ": " + self.event[2]) |
|  |
| elif 33 < rand <= 66: |
| self.logger.debug(self.name.ljust(10) + ": " + self.event[3]) |
|  |
| elif 66 < rand <= 100: |
| self.logger.debug(self.name.ljust(10) + ": " + self.event[4]) |
|  |
| else: |
| self.logger.system(self.name.ljust(10) + ": " + self.event[100]) |
| self.isWork = False |
|  |
| self.devicesPool.release() |
|  |
| time.sleep(random.randint(3, 15)) |
|  |
| def run(self): |
| """Запуск потока""" |
| if not self.isWork: |
| threading.Thread(target=self.work).start() |
|  |
| def turnOff(self): |
| """ |
| Функция выключения прибора |
| :return: |
| """ |
| self.isWork = False |
| self.logger.warn(self.name.ljust(10) + ": " + self.event[-100]) |

**Logger.py**

|  |
| --- |
| import datetime  import os |
|  |
|  |
| class Logger: |
| """ |
| Логер, умеющий делать запись в файл. |
| Использует для записи 'событий' прибора. |
|  |
| :see Device.py |
| """ |
|  |
| def \_\_writeLog(self, message: str): |
| """ |
| 'Приватный' метод записи лога в файл. |
| Вызывается в других функциях класса. |
|  |
| :param message: сообщение для записи |
| :return: |
| """ |
|  |
| # Получение даты и времени |
| time = datetime.datetime.now() |
| timeStr = str(time.date()) + " " + str(time.time()).split(".")[0] |
|  |
| with open(self.path, "a") as file: |
| file.write(timeStr + " " + message + "\n") |
|  |
| def \_\_deleteOldLogs(self): |
| """ |
| 'Приватный' метод очистки 'тяжелых' log файлов. |
| Если вес лога превышает 1Кб, то удаляем данный файл. |
| Для записи будет создан новый файл. |
| :return: |
| """ |
| directory = self.path.split('/')[0] |
| for file in os.listdir(directory): |
| filePath = directory + "/" + file |
| if os.path.getsize(filePath) > 1024: |
| os.remove(filePath) |
|  |
| def \_\_init\_\_(self, path: str): |
| """ |
| Конструктор с параметром |
| :param path: путь для записи лога |
| """ |
| self.path = path + ".log" |
| self.\_\_deleteOldLogs() |
|  |
| def system(self, message: str): |
| """ |
| Специальный уровень для логирования |
| начала работы и её удачного завершения |
| :param message: |
| :return: |
| """ |
| self.\_\_writeLog('SYSTEM'.ljust(6) + " " + message) |
|  |
| # Типичные уровни логирования |
| def info(self, message: str): |
| self.\_\_writeLog('INFO'.ljust(6) + " " + message) |
|  |
| def debug(self, message: str): |
| self.\_\_writeLog('DEBUG'.ljust(6) + " " + message) |
|  |
| def error(self, message: str): |
| self.\_\_writeLog('ERROR'.ljust(6) + " " + message) |
|  |
| def warn(self, message: str): |
| self.\_\_writeLog('WARNING'.ljust(6) + " " + message) |

**Reader.py**

|  |
| --- |
| import threading |
| from threading import BoundedSemaphore |
|  |
|  |
| class Reader: |
| """ |
| Класс читатель |
| Читает из лог файла события воспроизводимые прибором |
| """ |
|  |
| # Ограничение кол-во параллельно работающих чтецов |
| # при помощи семафора |
| readersPool: BoundedSemaphore = BoundedSemaphore(10) |
|  |
| def \_\_init\_\_(self, path: str, text): |
| """ |
| Конструтор с параметром |
| :param path: путь к лог файлу для чтения |
| """ |
| self.path = path |
| self.text = text |
|  |
| def read(self): |
| """ |
| Метод для запуска отедльного потока |
| Читаем последнюю строку из файла и сравниванием ее |
| с предыдущей последней строкой. |
| Если строки не совпадают, значит прибор находится в новом |
| состоянии о чем читатель и сообщает |
| :return: |
| """ |
| lastLine = str() |
|  |
| # Цикл пока в строке не встретится 'Завершение', |
| # вне зависимости от успешного/аварийного прерывания работы приборы |
| # работаем с поледними строками файла |
| while not ("Завершение" in lastLine.split(" ")): |
| # Обращение к файлу - разделяемому ресурсу |
| # ограждается семафорами |
| self.readersPool.acquire() |
| with open(self.path, 'r') as file: |
| line = file.readlines()[-1] |
| if lastLine != line: |
| self.text.insert('end', line) |
| self.text.see('end') |
| lastLine = line |
|  |
| self.readersPool.release() |
|  |
| def run(self): |
| """Запуск потока""" |
| threading.Thread(target=self.read).start() |

**main.py**

|  |
| --- |
| import tkinter as tk  from tkinter import \* |
|  |
| from Device import Device |
|  |
| devices = list() |
|  |
| logDirectory = "Logs/" |
|  |
| devicesCount = 10 |
|  |
|  |
| def onAllDevices(): |
| for el in range(devicesCount): |
| devices[el].run() |
|  |
|  |
| def offAllDevices(): |
| for el in range(devicesCount): |
| devices[el].turnOff() |
|  |
|  |
| def startGUI(): |
| win = tk.Tk() |
| win.title('Вар 8. Распредленное файловое хранилище') |
| win.geometry("980x590+200+50") |
| win.resizable(False, False) |
|  |
| text = tk.Text(win, height=31, width=84, bg='black', |
| fg='white', |
| wrap=WORD, |
| insertbackground='white', |
| relief=RIDGE, |
| border=5, |
| ) |
| text.grid(column=5, row=0, rowspan=100) |
|  |
| scrollBar = tk.Scrollbar(command=text.yview) |
| text.config(yscrollcommand=scrollBar.set) |
| scrollBar.grid(column=6, row=0, rowspan=15, sticky=N + S, pady=20) |
|  |
| for i in range(devicesCount): |
| devices.append(Device(f"{logDirectory}Прибор{i + 1}", text)) |
|  |
| for i in range(devicesCount): |
| label = tk.Label(win, text=f"{devices[i].name}:", height=3) |
| label.grid(row=i, column=0, columnspan=2, padx=10) |
|  |
| buttonOn = tk.Button(win, text='вкл', bg='green', |
| command=devices[i].run) |
| buttonOff = tk.Button(win, text='офф', bg='red', |
| command=devices[i].turnOff) |
|  |
| buttonOn.grid(row=i, column=3) |
| buttonOff.grid(row=i, column=4) |
|  |
| buttonOnAll = tk.Button(win, text='вкл всё', bg='green', |
| command=onAllDevices) |
|  |
| buttonOffAll = tk.Button(win, text='выкл всё', bg='red', |
| command=offAllDevices) |
|  |
| buttonOnAll.grid(row=11, column=3) |
| buttonOffAll.grid(row=11, column=4) |
|  |
| win.mainloop() |
|  |
|  |
| if \_\_name\_\_ == '\_\_main\_\_': |
| startGUI() |

1. **Вывод**

В ходе данной работы были закреплены навыки разработки многопоточных приложений на языке Python средствами библиотеки threading и разработки GUI при помощи Tkinter.

Возможные пути развития:

* Держать в памяти читателя не весь файл, а только последнюю изменённую строку
* Производить чтение из файла только при его изменении.
* Сделать возможным задание максимального кол-во приборов.

1. **Приложения**

**Приложение 1.**

