БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ
Кафедра информатики

Факультет: ИНО
Специальность: ИиТП

Индивидуальная практическая работа № 2
по дисциплине “Операционные системы и среды”
Вариант 6
“Управление процессами и взаимодействие процессов”

Выполнил студент: Дубейковский А.А.
Группа № 893551
Зачётная книжка № 75350046

Условие

Вариант задания № 6.

Копирование файла двумя параллельными процессами: один осуществляет чтение из файла-источника, второй – запись в файл-приемник. Используются 2 независимых буфера: для чтения и для записи, которые меняются местами (передаются между процессами) по мере соответственно заполнения и опустошения. В качестве буферов используются блоки разделяемой памяти (*shared memory*), для синхронизации процессов и управления доступом к буферам – семафоры (*semaphore*).

Примечание. Различие скоростей чтения и записи может приводить к простоям процессов, что необходимо учесть: заполненный «буфер чтения» можно передать как «буфер записи» в распоряжение «процесса записи» независимо от его состояния, но «процесс чтения» все равно вынужден будет ждать освобождения бывшего «буфера записи», чтобы использовать его как «буфер чтения». Представляет интерес оценка эффекта от распараллеливания процессов, но заметной она будет лишь на больших размерах файлов и медленных устройствах

Код и пояснения к нему

Все необходимые пояснения написаны в комментариях в скрипте, который будет показан тут как фото из PyCharm IDE. Также код и тесты к нему выложены на гитхабе, код можно скачать и тесты можно запустить у себя локально:
https://github.com/ElephantT/System_Programming/tree/main/osis_ipr2

Код:
Условие

```
# RU


"""
    ОСИС ИПР 2
    Александр Дубейковский - зачётная книжка 75350046, группа 893551, Вариант 6

    Копирование файла двумя параллельными процессами: один осуществляет чтение из
файла-источника, второй – запись в файл-приемник. Используются 2 независимых буфера:
для чтения и для записи, которые меняются местами (передаются между процессами) по мере
соответственно заполнения и опустошения. В качестве буферов используются блоки разделяемой
памяти (shared memory), для синхронизации процессов и управления доступом к
буферам - семафоры (semaphore).
    Примечание. Различие скоростей чтения и записи может приводить к простоям процессов,
что необходимо учесть: заполненный «буфер чтения» можно передать как «буфер
записи» в распоряжение «процесса записи» независимо от его состояния,
но «процесс чтения» все равно вынужден будет ждать освобождения бывшего «буфера записи»,
чтобы использовать его как «буфер чтения». Представляет интерес оценка эффекта от
распараллеливания процессов, но заметной она будет лишь на больших размерах файлов и медленных
устройствах
"""


```

Multiprocessing.dummy — это библиотека Python, которая позволяет работать именно с потоками, а не процессами, в отличии от самой библиотеки multiprocessing. Deepcopy использовать буду дальше для копирования одного буффера в другой.

Тут показаны две функции, одна для чтения из конкретного файла, начиная с конкретного индекса, конкретное кол-во байт. Вторая записывает в файл (через append, потому и проверка mode на ‘a’), так было удобнее реализовать, чем через использования индексов снова. Делаю по ходу работы функций некоторые выводы чтобы было удобно смотреть за результатом работы.

```python
import os
from multiprocessing.dummy import Pool as ThreadPool
from copy import deepcopy


def fileRead(file, size_of_file_to_copy, index_from_where_to_read, buffer, size_of_buffer):
    buffer[0] = ""
    if index_from_where_to_read >= size_of_file_to_copy:
        return True
    file.seek(index_from_where_to_read)
    buffer[0] = file.read(size_of_buffer)
    return True


def fileWrite(file, buffer):
    if buffer[0] == "":
        return True
    if file.mode != "a":
        print("File isn't opened in right mode to write in it")
        return False
    file.write(buffer[0])
    print("Buffer that is currently writing to file:")
    print(buffer[0])
    print("---")
    return True
```

Эта функция нужна просто для удобного использования библиотеки с потоками для вызова функций с большим количеством параметров

```python


def distributor(args):
    function, arguments = args
    value = function(*arguments)
    return value


```

Сама функция параллельного копирования файла. В начале просто проверяем что файлы с такими именами есть или отсутствуют

```python
def parallelFileCopying(filename_to_copy, filename_where_to_copy, size_of_buffer):
    # check existance and open file to only read from it
    try:
        file_to_read = open(filename_to_copy, "r")
    except FileNotFoundError:
        print(f"File {filename_to_copy} doesn't exist")
        return False
    # get size of that file
    size_of_file_to_copy = os.path.getsize(f"{filename_to_copy}")

    # check that file with that filename doesn't exist
    try:
        open(filename_where_to_copy, "r")
        print(f"You can't copy your file to a file with name as {filename_where_to_copy},"
              f" because it is already exists")
        return False
    except FileNotFoundError:
        print(f"File {filename_where_to_copy} doesn't exist")
        print("We can create such file and continue our copying task to this new file")

```

А вот здесь уже происходят главные вызовы. Создаются два буфера (создаю их как списки, чтобы мы могли передавать именно их в функции, а не их копии, то есть за счёт этого мы при каждом вызове функций будем записывать и считывать из одного места в памяти - по сути как в C/C++ языках передаётся адрес переменной). Затем мы делаем 2 thread-a, и в каждый из них помещаем по функции с определёнными аргументами (для этого и помогает та функция distributor, чтобы каждой из функций выдать нужные ей аргументы). Затем, как оба потока отработают, проверяем результаты, правильно ли они отработали, меняем буфферы (буффер для чтения я просто обнуляю при вызове чтения, а после выполнения одного шага, буффер для записи заполняю значением буффера чтения).

```
    # create file
    open(filename_where_to_copy, "x")
    # open file to only append to it
    file_to_write = open(filename_where_to_copy, "a")

    buffer_for_reading = [""]
    buffer_for_writing = [""]
    index_from_where_to_read = 0
    # parallel reading and writing for file copying
    # we run both functions in parallel till we've read all the file AND buffer that we need to write
    # is not empty
    while index_from_where_to_read < size_of_file_to_copy or (buffer_for_writing != [""]):
        function_names = [fileRead, fileWrite]
        args_list = [[file_to_read, size_of_file_to_copy, index_from_where_to_read,
                      buffer_for_reading, size_of_buffer],
                     [file_to_write, buffer_for_writing]]

        results = ThreadPool(2).map(distributor, zip(function_names, args_list))
        if not all(results):
            print("We couldn't read file or write to file")
            return False
        buffer_for_writing[0] = deepcopy(buffer_for_reading[0])
        index_from_where_to_read += size_of_buffer
    return True
```

Сам вызов функции. Тут можно определить размер буффера (можно выбирать любой). Имя файла, который мы будем копировать (либо путь до него, либо только имя, если держим его со сриптом вместе в одной директории). И новое имя файла записываем, который создастся в ходе работы скрипта.

```
if __name__ == '__main__':
    size_of_buffer = 32
    filename_to_copy = "text.txt"
    print(f"Size of {filename_to_copy} = " + str(os.path.getsize(f"{filename_to_copy}")) + " bytes")
    filename_where_to_copy = "new_file.txt"
    was_it_successful = parallelFileCopying(filename_to_copy, filename_where_to_copy, size_of_buffer)
    print("Was it successful: " + str(was_it_successful))
    print(f"Size of {filename_where_to_copy} = " + str(os.path.getsize(f"{filename_where_to_copy}"))
          + " bytes")

```

Тесты

Тесты запускались на ОС семейства Linux – Ubuntu 20.04.3 LTS (Focal Fossa).

Запустим код (можно скачать его с гитхаба и запустить у себя, либо с файлом для копирования с гитхаба, либо скопировать свой, просто тогда его имя занести в скрипт):

```
(base) elephant@seaelephant:~/University/OS/osis_ipr2$ cat text.txt
Hello world. In Python you can use thread parallelization with 'multiprocessing.dummy' library

(base) elephant@seaelephant:~/University/OS/osis_ipr2$ cat new_file.txt
cat: new_file.txt: No such file or directory
(base) elephant@seaelephant:~/University/OS/osis_ipr2$ python osis_ipr_2_Dubeykovsky.py
Size of text.txt = 96 bytes
File new_file.txt doesn't exist
We can create such file and continue our copying task to this new file
Buffer that is currently writing to file:
Hello world. In Python you can u
---
Buffer that is currently writing to file:
se thread parallelization with '
---
Buffer that is currently writing to file:
multiprocessing.dummy' library


---
Was it successful: True
Size of new_file.txt = 96 bytes
(base) elephant@seaelephant:~/University/OS/osis_ipr2$ cat new_file.txt
Hello world. In Python you can use thread parallelization with 'multiprocessing.dummy' library

(base) elephant@seaelephant:~/University/OS/osis_ipr2$
```