НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ

імені Ігоря Сікорського»

Факультет прикладної математики

Кафедра прикладної математики

Звіт

із лабораторної роботи №*1*

З дисципліни “Розподілені і хмарні обчислення”

|  |  |
| --- | --- |
| Виконала: | Керівник: |
| студентка групи КМ-81 | *асистент Ліскін В.О.* |
| *Верзун П.В.* |  |

Київ — 2021

ВСТУП

Пiд *процесом* (process) розумiють виконувану програму. Кожний процес забезпечує систему потрiбними для виконання ресурсами.

Процес має вiртуальний адресовий простiр, виконуваний код, вiдкритi дескриптори системних об’єктiв (open handles to system objects), безпековий контекст (security context), унiкальний iдентифiкатор, змiннi середовища, клас прiоритету, мiнiмальний та максимальний розмiр робочої множини (working set), а також щонайменше один потiк.

*Потiк* (thread) — це базова одиниця в рамках процесу, якiй ОС видiляє процесорний час. Потiк може виконувати будь-яку частину коду процесу, у тому числi частини коду, якi в даний момент виконує iнший потiк. Усi потоки процесу мають спiльний вiртуальний адресовий простiр та системнi ресурси. У процесi своєї роботи процес може створювати додатковi потоки.

ОС Windows пiдтримує режим *витискуючої багатозадачевостi* (preemptive multitasking), яка створює ефект одночасного виконання багатьох потокiв. На однопроцесорнiй машинi це, звiсно, не бiльше нiж iлюзiя, яка створюється за рахунок швидкого перемикання мiж потоками. На мультипроцесорнiй машинi система може справдi одночасно виконувати стiльки потокiв, скiльки в машинi є процесорiв.

У данiй роботi розглядатимемо засоби роботи з багатопотоковими аплiкацiями в ОС Windows версiї XP та вище.

1 ПОСТАНОВКА ЗАДАЧI

У данiй лабораторнiй роботi потрiбно ознайомитися з концепцiєю багатозадачевостi в ОС Windows, навчитися створювати за допомогою засобiв Windows API багатопотоковi аплiкацiї та розбиратися з проблемами, що виникають пiд час спiльного множинного доступу до ресурсiв.

У рамках виконання лабораторної роботи потрiбно:

а) ознайомитися з теоретичними вiдомостями, викладеними в роздiлi 1.3;

б) написати будь-якою мовою програмування програму, яка повинна:

1) за потреби створювати додатковi потоки, окрiм головного;

2) регулювати доступ потокiв до деякого ресурсу (файлу, графiчного об’єкта, статичної змiнної i т.п.); зокрема, у програмi повинно бути передбачено можливiсть як синхронного, так i асинхронного доступу потокiв до ресурсу (на вимогу користувача);

3) регулювати прiоритети потокiв на вимогу користувача;

4) за потреби зупиняти та вiдновлювати потоки;

5) використовувати тiльки засоби Windows API i не повинна використовувати стандартнi процедури MFC, спецiальнi класи для реалiзацiї багатопотокових аплiкацiй тощо;

в) вiдлагодити програму в ОС Windows версiї XP та вище;

г) пiдготувати звiт iз лабораторної роботи вiдповiдно до вимог роздiлу 1.4. Кожен студент повинен написати програму з iндивiдуальними функцiональними можливостями. Програми рiзних студентiв не можуть мати  однаковi функцiональнi можливостi.

Типова програма, яку можна написати в рамках даної лабораторної роботи, може передбачати наявнiсть одного iнтерфейсного потоку, який здiйснює спiлкування з користувачем та керування iншими потоками, а також декiлькох робочих потокiв, якi виконують фоновi задачi (перерахунок даних, друк тексту, уведення/виведення, пошук файлiв тощо).

2 ТЕОРЕТИЧНІ ВІДОМОСТІ

**Режим багатозначності**

Багатозадачевi операцiйнi системи роздiляють наявний процесорний час мiж процесами чи потоками, якi його потребують. ОС Windows розроблено для витискуючої багатозадачевостi: вона видiляє *квант часу* (time slice) кожному потоку, який вона виконує. Щойно квант часу поточного потоку вичерпано, ОС зупиняє його виконання, дозволяючи таким чином виконуватися iншим потокам.

4

Пiд час перемикання з одного потоку на iнший, ОС зберiгає контекст витисненого потоку та вiдновлює збережений ранiше контекст наступного потоку в черзi. Величина кванту часу залежить вiд ОС та вiд процесора. Проте, оскiльки зазвичай квант часу доволi малий (приблизно 20 мс), створюється враження, що одночасно виконуються багато потокiв. Проте, багатопотоковiсть варто використовувати обережно, оскiльки за великої кiлькостi потокiв продуктивнiсть ОС може знизитися.

**Переваги багатозадачевостi**

Використання багатозадачевостi має такi переваги:

- для користувача перевага полягає в можливостi одночасно працювати з декiлькома аплiкацiями. Наприклад, користувач може редагувати файли в однiй аплiкацiї у той час, поки музичний плеєр програє музичнi файли;

- для розробника перевага полягає в можливостi створення аплiкацiй, що використовують бiльше за один процес, а також створення процесiв, що використовують бiльше за один потiк. Наприклад, процес може мати потiк графiчного iнтерфейсу користувача для опрацювання спiлкування з користувачем, а також низку робочих потокiв для виконання певних задач пiд час очiкування уведення з боку користувача. Якщо надати потоку iнтерфейсу бiльший прiоритет, то аплiкацiя буде швидше реагувати на запити користувача, у той час як робочi потоки ефективно використовуватимуть час процесора пiд час очiкування уведення з боку користувача.

Окрiм цього, багатопотоковiсть можна використовувати для

пришвидшення роботи аплiкацiй на мультипроцесорних системах. Фактично, у таких системах це — єдиний варiант розпаралелювання виконання задач.

Якщо аплiкацiя має тiльки один процес з одним потоком, то вона буде виконуватися однаково швидко як на однопроцесорнiй, так i на багатопроцесорнiй машинi. Тому для ефективного використання можливостей мультипроцесорних систем доцiльно володiти основами створення багатопотокових аплiкацiй.

3 ОПИС РОЗРОБЛЕНОЇ ПРОГРАМИ

Дана програма дозволяє користувачу створювати синхронні потоки та асинхронні потоки, їх видалення, призупинненя та відновлювання. Більш того, усі створені потоки діють на глобальну змінну та додают до неї 1, усі результати логуються у лог-файл. (кожні 5 секунд)

4 РЕЗУЛЬТАТИ ВИПРОБУВАННЯ РОЗРОБЛЕНОЇ ПРОГРАМИ

Приклад роботи з програмою:

new thread command - adding a new thread

new process - adding a new process

stop thread - stopping thread

delete

resume

show - show all

----

new thread

Initializing thread

Thread 1 is started

new thread command - adding a new thread

new process - adding a new process

stop thread - stopping thread

delete

resume

show - show all

----

new process

No such command

new thread command - adding a new thread

new process - adding a new process

stop thread - stopping thread

delete

resume

show - show all

----

new process

Initializing process

Process 2 is started

new thread command - adding a new thread

new process - adding a new process

stop thread - stopping thread

delete

resume

show - show all

----

stop thread

Input number of thread you want to stop [1, 2] 2

new thread command - adding a new thread

new process - adding a new process

stop thread - stopping thread

delete

resume

show - show all

----

resume

Input number of thread you want to resume [2] 2

new thread command - adding a new thread

new process - adding a new process

stop thread - stopping thread

delete

resume

show - show all

----

show

1 - <CustomThread(Thread-1, started 140435739461376)>

2 - <CustomProcess(CustomProcess-1, stopped[SIGTERM])>

new thread command - adding a new thread

new process - adding a new process

stop thread - stopping thread

delete

resume

show - show all

----

delete

Input the number of identity to stop [1, 2] 1

new thread command - adding a new thread

new process - adding a new process

stop thread - stopping thread

delete

resume

show - show all

----

show

2 - <CustomProcess(CustomProcess-1, stopped[SIGTERM])>

new thread command - adding a new thread

new process - adding a new process

stop thread - stopping thread

delete

resume

show - show all

----

delete

Input the number of identity to stop [2] 2

new thread command - adding a new thread

new process - adding a new process

stop thread - stopping thread

delete

resume

show - show all

----

show

new thread command - adding a new thread

new process - adding a new process

stop thread - stopping thread

delete

resume

show - show all

----

представлено зміст лог-файлу:

Thread 1 is currently executing Result = 1 counter = 1

Thread 1 is currently executing Result = 2 counter = 1

Thread 1 is currently executing Result = 3 counter = 1

Thread 1 is currently executing Result = 4 counter = 1

Thread 1 is currently executing Result = 5 counter = 1

Process 2 is working Result of working = 6 counter = 1

Thread 1 is currently executing Result = 6 counter = 1

Process 2 is working Result of working = 7 counter = 1

Thread 1 is currently executing Result = 7 counter = 1

Process 2 is working Result of working = 8 counter = 1

Thread 1 is currently executing Result = 8 counter = 1

Process 2 is working Result of working = 9 counter = 1

Thread 1 is currently executing Result = 9 counter = 1

Process 2 is working Result of working = 10 counter = 1

Thread 1 is currently executing Result = 10 counter = 1

Process 2 is working Result of working = 11 counter = 1

Thread 1 is currently executing Result = 11 counter = 1

Process 2 is working Result of working = 12 counter = 1

Thread 1 is currently executing Result = 12 counter = 1

Process 2 is working Result of working = 13 counter = 1

Thread 1 is currently executing Result = 13 counter = 1

Stopping thread 2...

Thread 1 is currently executing Result = 14 counter = 1

Thread 1 is currently executing Result = 15 counter = 1

Thread 1 is currently executing Result = 16 counter = 1

Thread 1 is currently executing Result = 17 counter = 1

Thread 1 is currently executing Result = 18 counter = 1

Deleting identity 1...

Thread 1 is killed

Deleting identity 2…

ВИСНОВКИ

У данiй лабораторнiй роботi ознайомились з концепцiєю багатозадачевостi в ОС Windows,використовуючи мову програмування Python.

Під час виконання лабораторної роботи:

а) ознайомились з теоретичними вiдомостями, викладеними в роздiлi 1.3;

б) написали мовою програмування Python програму, яка:

1) за потреби створювати додатковi потоки, окрiм головного;

2) регулювати доступ потокiв до деякого ресурсу

3) регулювати прiоритети потокiв на вимогу користувача;

4) за потреби зупиняти та вiдновлювати потоки;

ПЕРЕЛІК ПОСИЛАНЬ

1. https://medium.com/velotio-perspectives/an-introduction-to-asynchronous-p rogramming-in-python-af0189a88bbb

2. https://docs.python.org/3/library/threading.html

3. https://docs.python.org/3/library/logging.html

4. https://www.cs.uic.edu/~jbell/CourseNotes/OperatingSystems/4\_Threads.ht ml

5. https://www.baeldung.com/cs/async-vs-multi-threading

Додаток А

import threading

from time import sleep

from multiprocessing import Process, current\_process, Lock, Event

import logging

import sys

import random

GLOBAL\_COUNTER = 0

print\_to\_logs = logging.getLogger()

print\_to\_logs.setLevel(logging.DEBUG)

output\_file\_handler = logging.FileHandler("shared.log")

print\_to\_logs.addHandler(output\_file\_handler)

class MultiThread(threading.Thread):

def \_\_init\_\_(self, id, counter):

print("Initializing thread")

threading.Thread.\_\_init\_\_(self)

self.process\_id = id

self.condition = False

self.state = threading.Condition()

self.type = "parallel"

self.status = "run"

self.counter = counter

def run(self):

print(f"\nThread {self.process\_id} is started\n\t\t")

t = threading.currentThread()

while getattr(t, "run", True):

with t.state:

if getattr(t, "condition", True):

print\_to\_logs.info(f"Thread {self.process\_id} stopped")

t.state.wait()

global GLOBAL\_COUNTER

print\_to\_logs.info(

f"Thread {self.process\_id} is currently executing\t Result = {GLOBAL\_COUNTER + self.counter}\t counter = {self.counter}"

)

GLOBAL\_COUNTER += self.counter

sleep(5)

print\_to\_logs.info(f"Thread {self.process\_id} is killed")

class MultiProcess(Process):

def \_\_init\_\_(self, id, counter):

print("Initializing process")

Process.\_\_init\_\_(self)

self.process\_id = id

self.condition = False

self.state = threading.Condition()

self.type = "async"

self.status = "run"

self.event = Event()

self.counter = counter

def run(self):

print(f"\nProcess {self.process\_id} is started\n ")

t = current\_process()

while getattr(t, "run", True):

if getattr(t, "condition", True):

print\_to\_logs.info(f"Process {self.process\_id} stopped")

global GLOBAL\_COUNTER

print\_to\_logs.info(

f"Process {self.process\_id} is working\t Result of working = {GLOBAL\_COUNTER + self.counter}\t counter = {self.counter}"

)

GLOBAL\_COUNTER += self.counter

sleep(5)

print\_to\_logs.info(f"Process {self.process\_id} killed")

menu = """

new thread command - adding a new thread

new process - adding a new process

stop thread - stopping thread

delete

resume

show - show all

----

"""

identities = {}

identities\_n = 1

stopped\_n = []

while True:

sleep(1)

command = str(input(menu))

if command == "new thread":

thread = MultiThread(identities\_n, 1)

thread.start()

identities[identities\_n] = thread

identities\_n += 1

elif command == "new process":

process = MultiProcess(identities\_n, 1)

process.start()

identities[identities\_n] = process

identities\_n += 1

elif command == "delete":

thread\_numbers = list(identities.keys())

m = int(input(f"Input the number of identity to stop {thread\_numbers} "))

if m in thread\_numbers:

print\_to\_logs.info(f"Deleting identity {m}...")

if identities[m].type == "parallel":

identities[m].run = False

del identities[m]

elif identities[m].type == "async":

identities[m].terminate()

del identities[m]

else:

print(f"Identity {m} doesn't exist")

elif command == "stop thread":

thread\_numbers = list(identities.keys())

m = int(input(f"Input number of thread you want to stop {thread\_numbers} "))

if m in thread\_numbers:

stopped\_n.append(m)

print\_to\_logs.info(f"Stopping thread {m}...")

identities[m].status = "stop"

if identities[m].type == "parallel":

with identities[m].state:

identities[m].condition = True

else:

identities[m].terminate()

else:

print(f"Thread {m} doesn't exist")

elif command == "resume":

m = int(input(f"Input number of thread you want to resume {stopped\_n} "))

if m in stopped\_n:

identities[m].status = "run"

if identities[m].type == "parallel":

with identities[m].state:

identities[m].condition = False

identities[m].state.notify()

stopped\_n.remove(m)

else:

identities[m].event.set()

else:

print(f"Thread {m} doesn't exist or it is running now")

elif command == "show":

for key in identities:

print(key, "-", identities[key])

else:

print("No such command\n")