Міністерство освіти й науки України

Національний технічний університет України

«Київський політехнічний інститут ім. Ігоря Сікорського»

Кафедра автоматизації проектування енергетичних процесів і систем

ЗВІТ

Про виконання лабораторної роботи №7

З дисципліни «Архітектура системного програмного забезпечення»

**Потоки, перенаправлення потоків**

TI-92 Черноусова Дениса

Перевірила д.т.н. Левченко Л.О.

Київ - 2021

**Мета роботи:**

* ознайомлення з системними викликами для управління потоками в ОС Linux,
* перенаправлення потоків при роботі файлами і командами.

**Теоретичні відомості**

***Управління потоками***

Процес розбивається на виконувані одиниці - потоки (*threads*) (один і більше потоків).

*Потік* – *набір послідовність виконуваних команд процесора, які використовують загальний адресний простір процесу,* це елемент виконання всередині процесу: віртуальний процесор, стек або статус програми. У порівнянні з процесами взаємодія і синхронізація потоків вимагає менше часу, оскільки потоки одного процесу виконуються в одному адресному просторі.

*Процес містить один або кілька потоків*. Якщо процес містить тільки один потік, такий процес називається однопоточними. Це класичні процеси UNIX. Якщо процес містить більше одного потоку, такі процеси називаються багатопоточними.

Існує дві основні категорії реалізації потоків: *користувацькі потоки* - потоки, що реалізуються через спеціальні бібліотеки потоків і працюють в просторі користувача. *Потоки ядра* - потоки, що реалізуються через системні виклики і працюють в просторі ядра.

В ОС UNIX/Linux для потоків реалізований стандарт Р-потоків - POSIX (*Portable Operating System Interface*) - **pthreads** ("P" - от POSIX). Для написання багатопотокової програми API для роботи з Р-потоками надає біля 100 інтерфейсів. Кожна функція в API забезпечена префіксом **pthread\_.**

Прототипи функцій роботи з потоками і необхідні типи даних містяться в заголовки <**pthread.h>.** Ці функції не включені в стандартну бібліотеку мови С, вони знаходяться в бібліотеці ***libthread***. Тому в командному рядку для ***gсс*** необхідно додати опцію «***-pthread»:***

***gcc -Wall -Werror -pthread beard.c -o beard***

Кожний потік має свій ідентифікатор потоку, ID потоку. В програмах на С/С++ для ID потоків слід використовувати тип ***pthread\_t*** з ***<sys/types.h>.***

При роботі з потоками використовуються основні функції:

* створення потоку;
* блокування роботи потоку в очікування завершення іншого;
* дострокове завершення потоків;
* завершення роботи потоків.

***Створення потоку***. Потоки створюються функцією ***pthread\_create***, яка має наступну сигнатуру

***#include <pthread.h>.***

***int pthread\_create (pthread\_t \*thread,***

***const pthread\_attr\_t \*attr,***

***void \*(\*start\_routine) (void \*),***

***void \*arg);***

Ця функція визначена в заголовки <*pthread.h*>.

*Перший параметр* цієї функції є вказівником на змінну типу ***pthread\_t***, в яку буде записано адресу ідентифікатора створюваного потоку – ID.

*Другий параметр* є вказівником на змінну типу ***pthread\_attr\_t*** , використовується для установки атрибутів потоку. Цей об'єкт управляє деталями взаємодії потоку з іншою програмою. Якщо параметр дорівнює NULL, то потік буде створений з атрибутами за замовчуванням.

*Третім параметром* функції *pthread\_create* повинна бути адреса функції потоку – вказівник на функцію потоку. Ця функція відіграє для потоку ту ж роль, що функція *main* для головної програми. Функція потоку приймає один параметр типу покажчик на *void* і повертає значення типу вказівник на *void*.

*Четвертий параметр* функції *pthread\_create* має тип void\*. Цей параметр може використовуватися для передачі значення як аргумент у функцію потоку. Через нього можна передавати новому потоку параметри.

Після виклику *pthread\_create* функція потоку буде запущена на виконання паралельно з іншими потоками програми.

Функція повертає 0 в разі успіху, код помилки – в разі невдачі.

***Ідентифікатори потоків (TID* -***Thread ID***)** для потоків є аналогами ідентифікаторів процесів (PID). У той час як *PID призначаються ядром Linux, TID призначаються лише бібліотекою Р-потоків*. Цей тип представлений *pthread\_t*, і POSIX не вимагає, щоб він був арифметичним. TID нового потоку визначається за допомогою аргументу *thread* при успішному виклику *pthread\_create()*. Потік може отримати свій TID при запуску за допомогою функції ***pthread\_self():***

***#include <pthread.h>***

***pthread\_t pthread\_self (void);***

Використання функції: ***const pthread\_t me=pthread\_self();***

***Блокування роботи потоку.*** Приєднання дозволяє одному з потоків блокуватися в очікуванні завершення іншого:

#include <pthread.h>

***int pthread\_join(pthread\_t thread, void \*\*retval);***

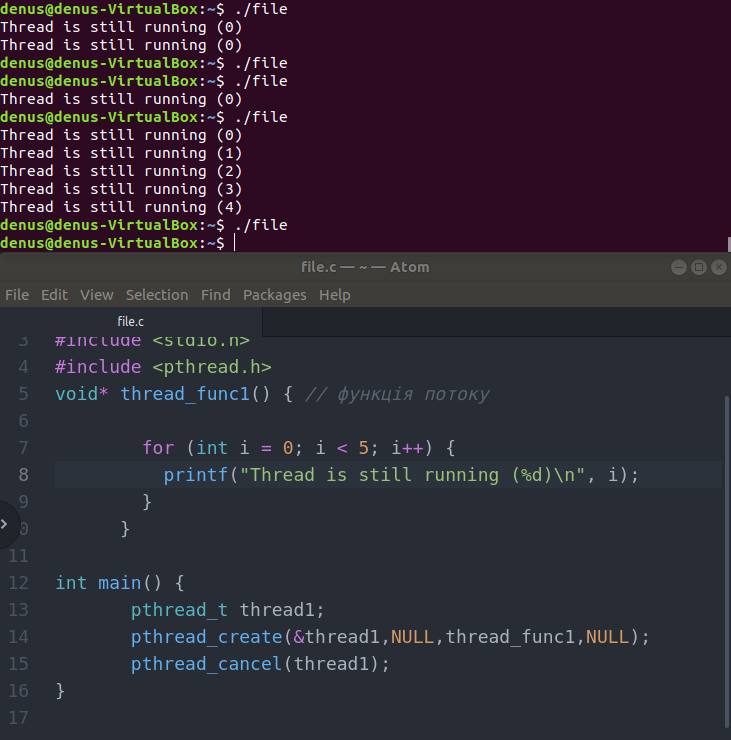
Після успішного виконання викликаючий потік блокується до тих пір, поки потік, вказаний як *thread*, не завершиться (якщо потік *thread* вже завершено, функція *pthread\_join()* повертається негайно). Як тільки *thread* завершується, викликаючий потік активізується і, якщо *retval* не дорівнює NULL, отримує значення завершеного процесу, яке повертається, і передане *pthread\_exit()* або повернене від його стартової процедури. Після цього можна сказати, що потоки *приєдналися* один до одного. *Приєднання завжди дозволяє потокам синхронізувати своє виконання по відношенню до періоду існування інших потоків*. Всі потоки в Р-потоках є рівноправними; кожний потік може приєднуватися до будь-якого іншого. Один потік може приєднуватися до багатьох (фактично, як ми скоро побачимо, найчастіше один головний потік очікує інших потоків, які сам і створив), але тільки один потік може намагатися приєднатися до певного іншого, декілька потоків не повинні намагатися приєднатися до будь-якого одного.

В разі успіху функція *pthread\_join()* повертає 0, в разі помилки - код помилки.

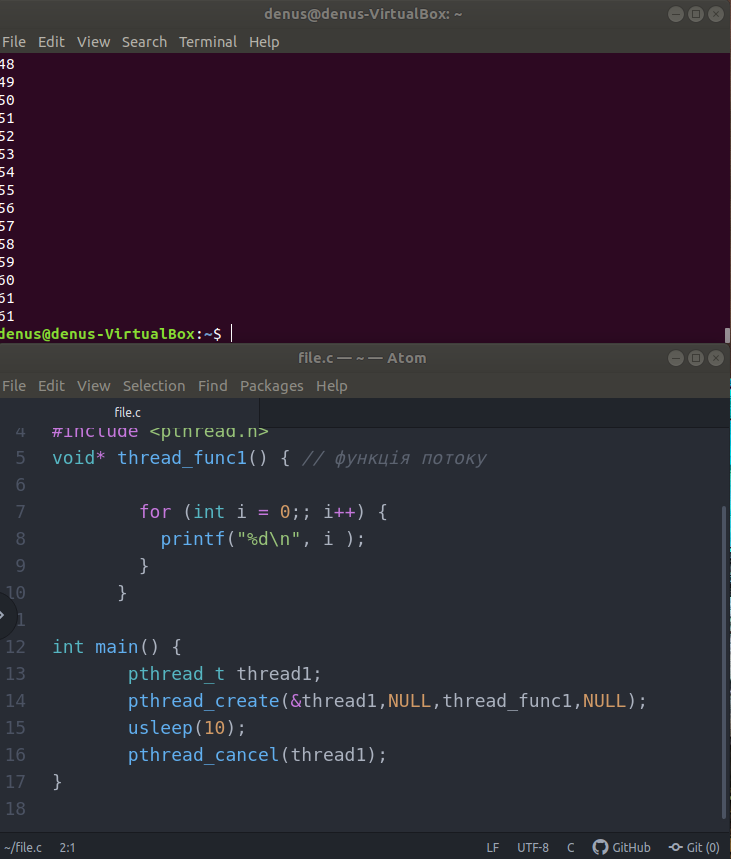
**Завдання :**

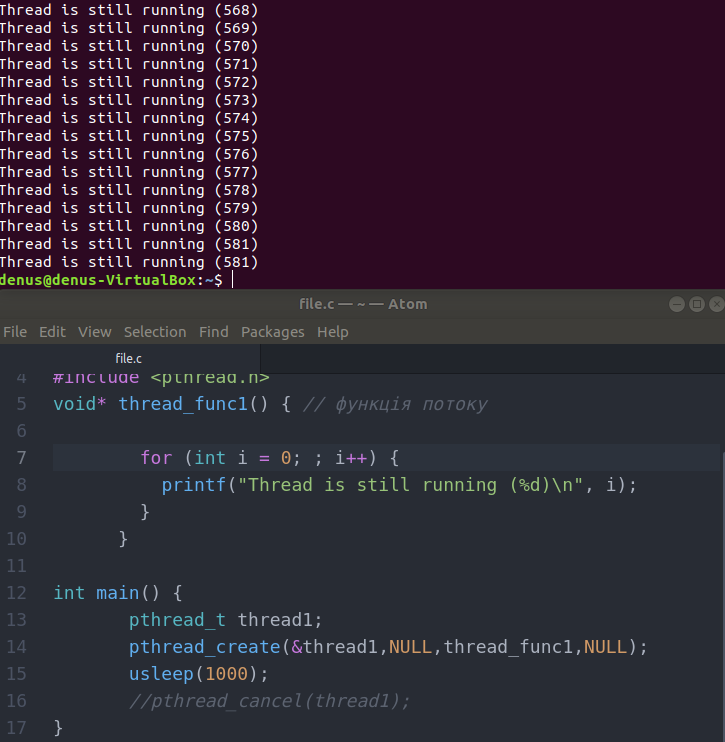
1. Опанувати команди по роботі з потоками.
2. Ознайомитися із засобами синхронізації потоків.
3. Ознайомитися із стандартними потоками введення/виведення
4. Використати вихідні тексти для створення програм.
5. Пояснити результати роботи програм та їх особливості.
6. Створити файл з особистими даними (група, прізвище, ім’я, по-батькові), використовуючи стандартне введення з клавіатури. Вивести дані на екран. Додати у файл інформацію про ваше хоббі. Створити канал.

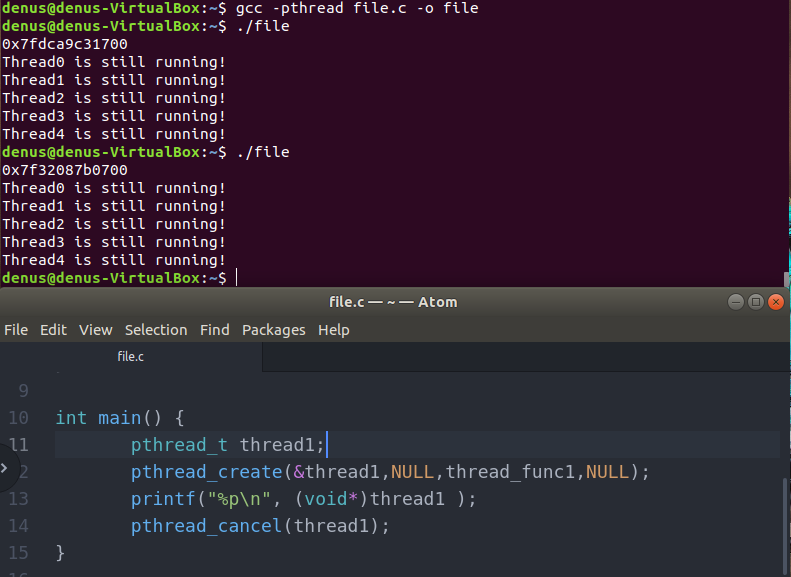
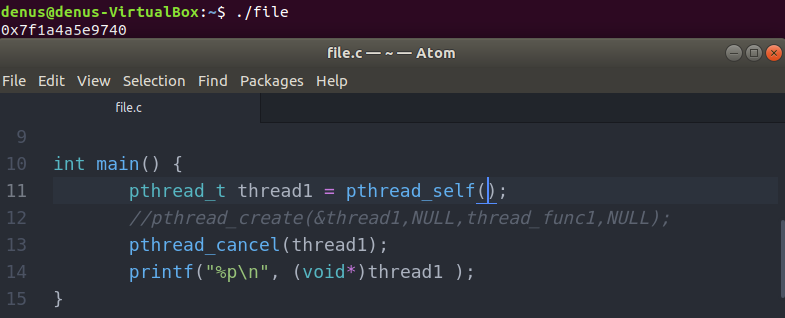
**Хід виконання роботи**

Компіляція файлу з необхідним ключем

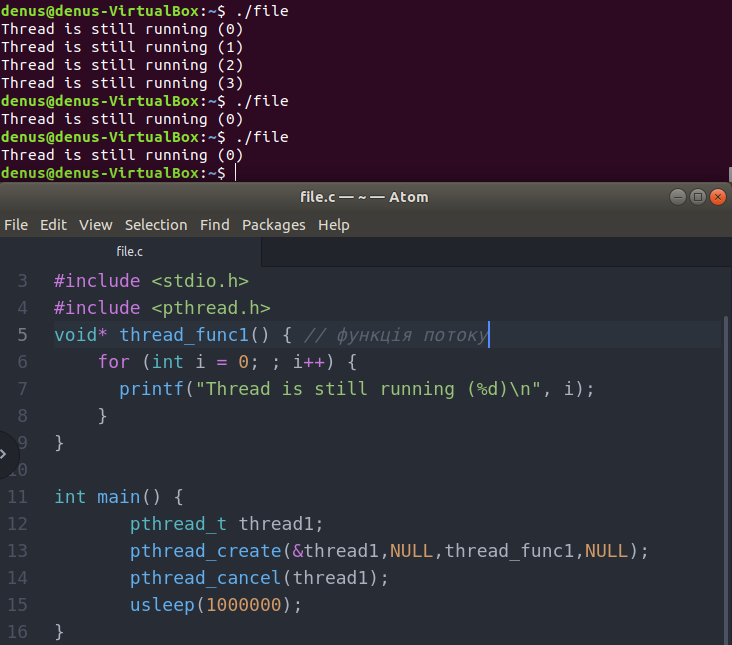
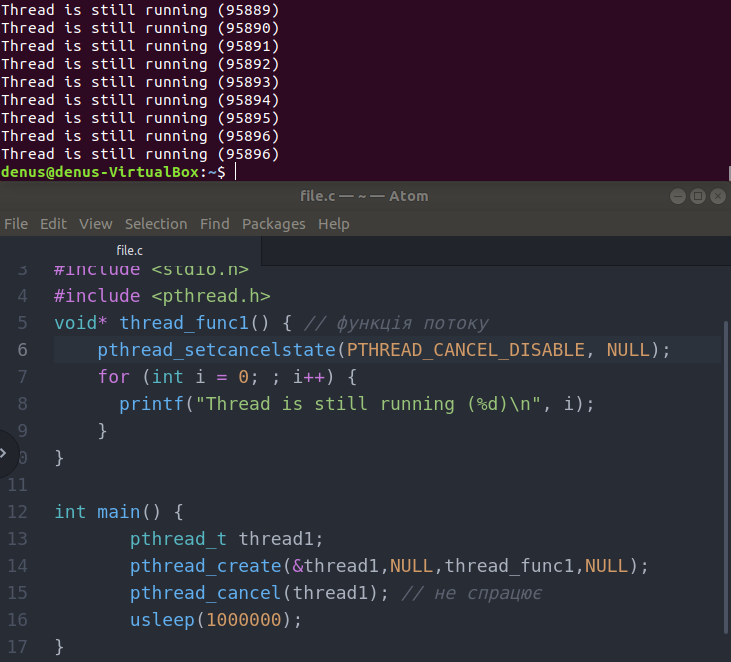
Створення однопоточного процесу, де головний потік та його підпотік «змагаються в швидкості»

Використання usleep аби дати вивести дані з безкінечного підпотоку

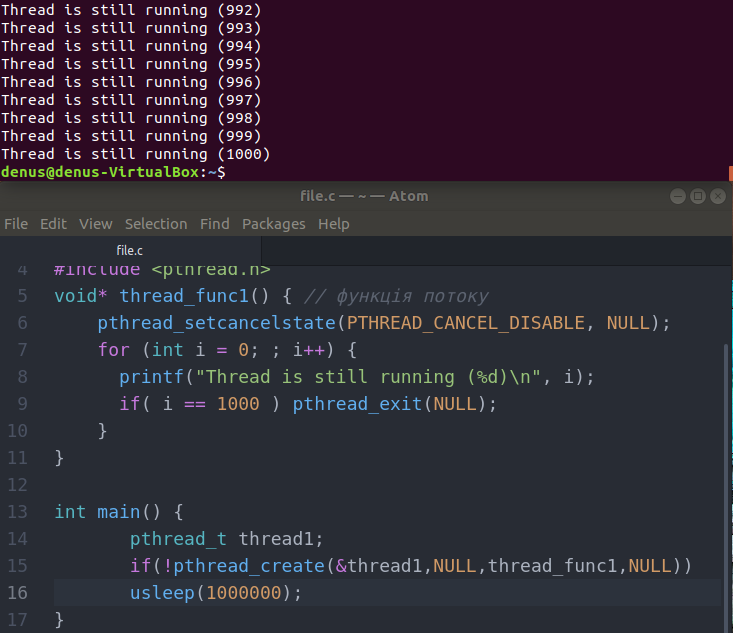
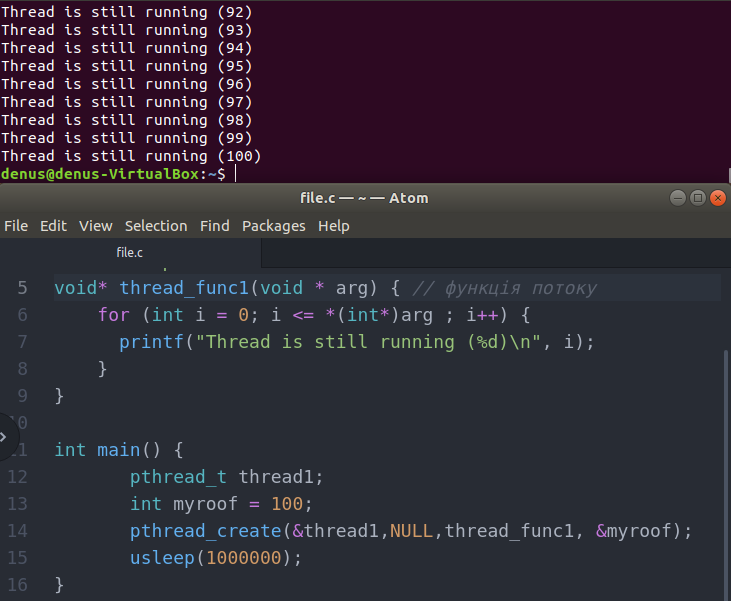
Очевидно, що при 1000 мкс = 1 с підпотік буде довше працювати

Виведення адреси потоку, створеної через pthread\_create

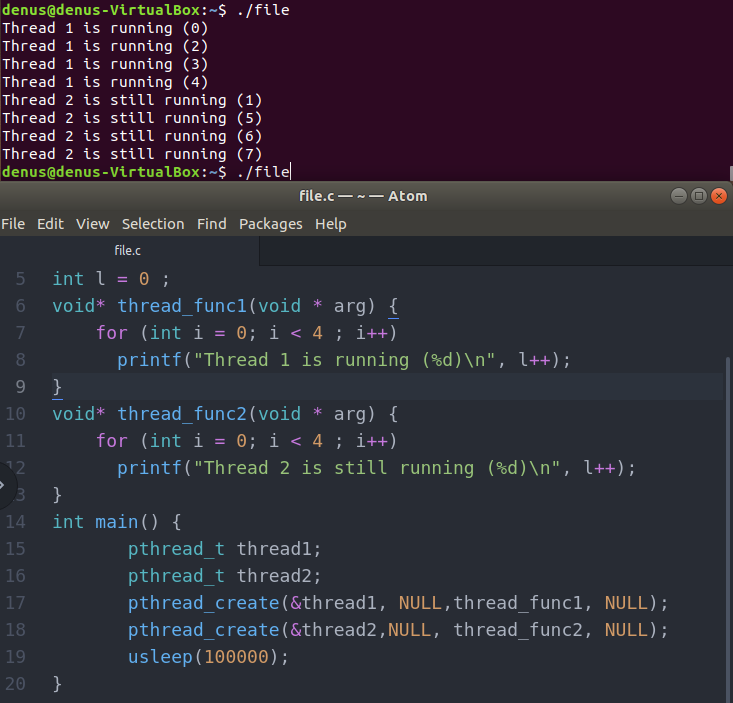
Виведення адреси потоку, створеної через pthread\_self

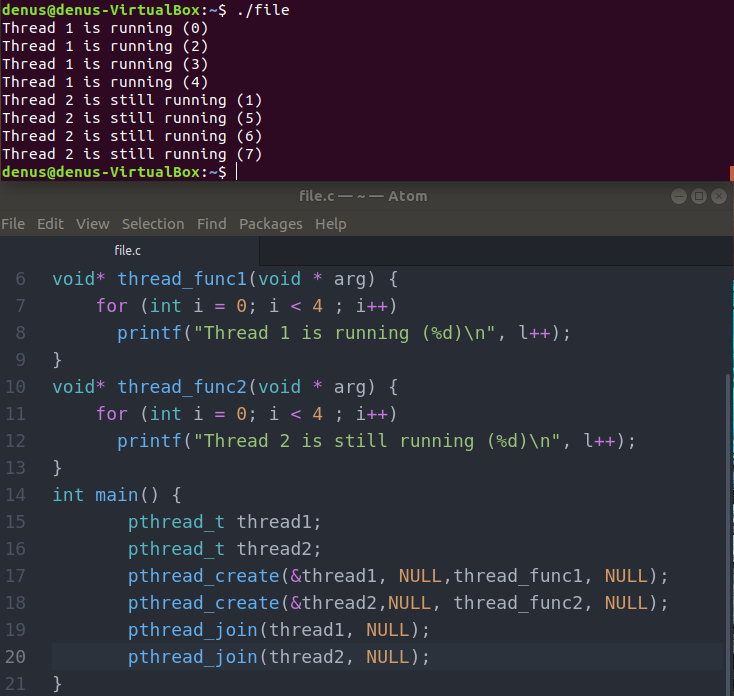
Команда pthread\_cancel надсилає запит припинення роботи процесу за його зміної адресу

Проте, чи може потік бути скасований і коли, залежить від його *стану відміни і типу скасування відповідно*

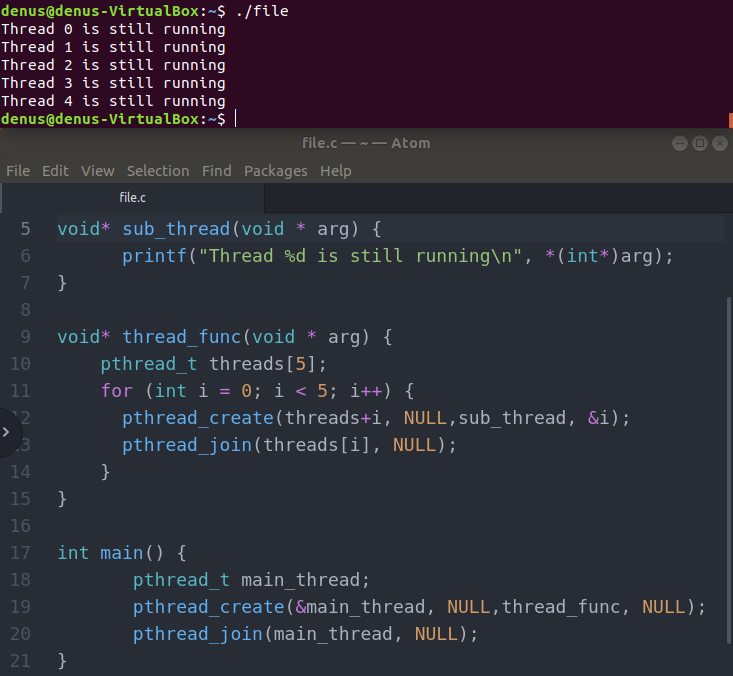
 У більшості команд бібліотеки pthread при коректному спрацювані команди йде вивід 0, інакше виводить код помилки

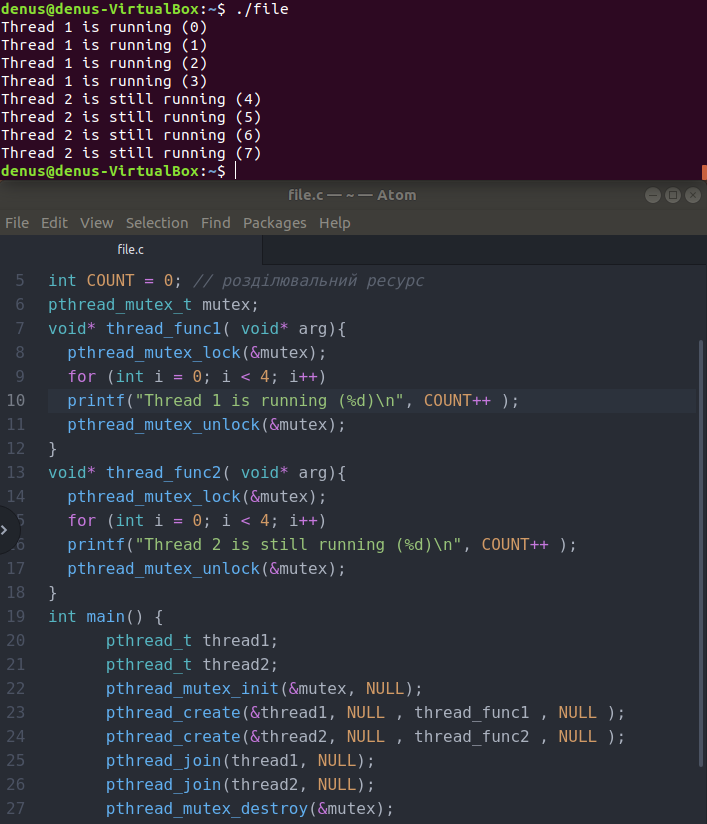
Передача та використання аргументу в потоці

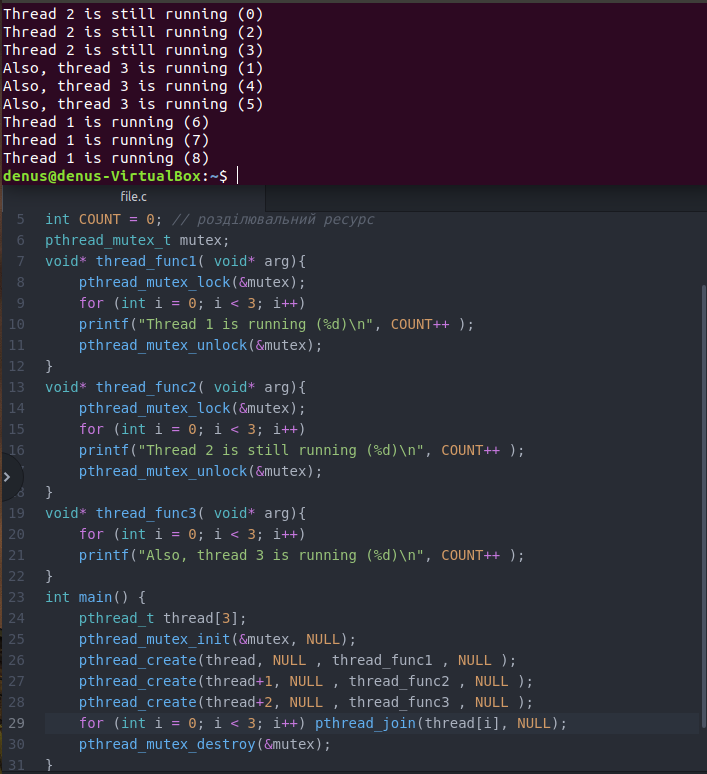
 Робота двох паралельних потоків у одному процесі

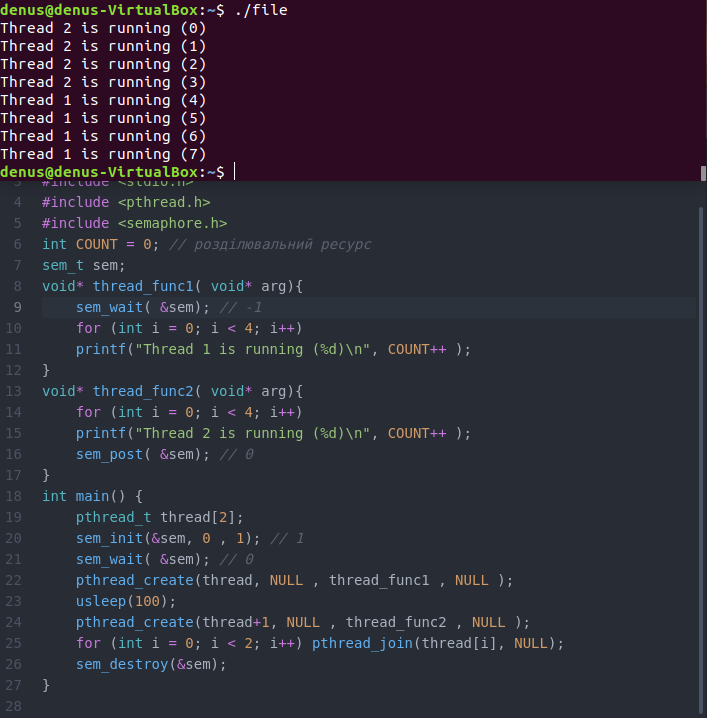
Команда pthread\_join приєднує до потоку на якому знаходиться, той потік що вказаний в дужках. Тобто головний завершить тоді, коли приєднані потоки до нього теж завершаться.

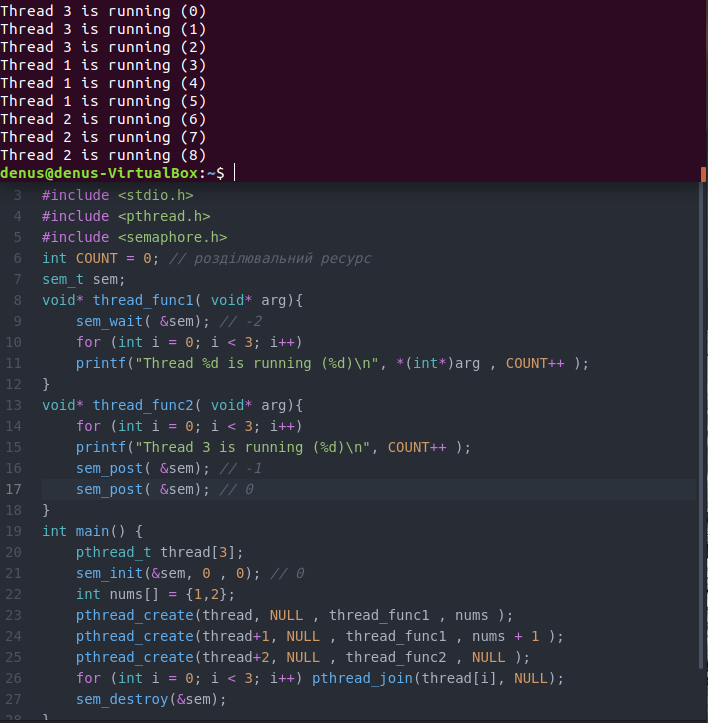
При цьому pthread\_join не можна використовувати на адреси нестворених потоків (Segmentation fault)

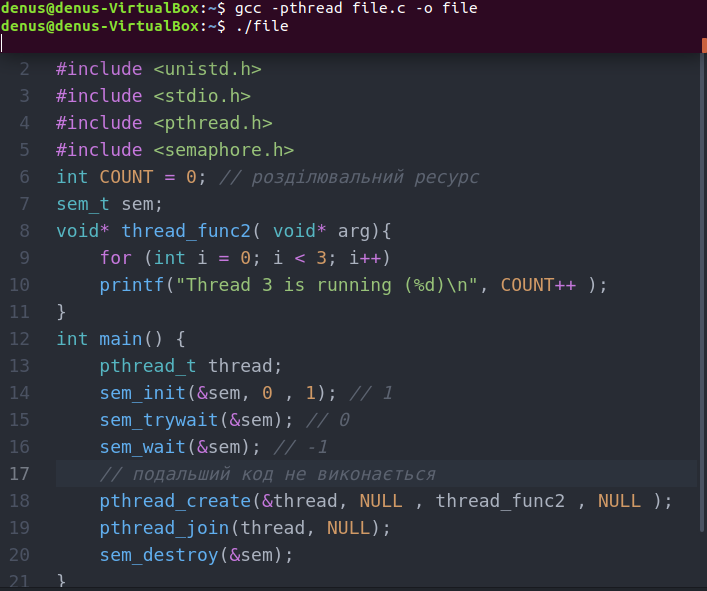
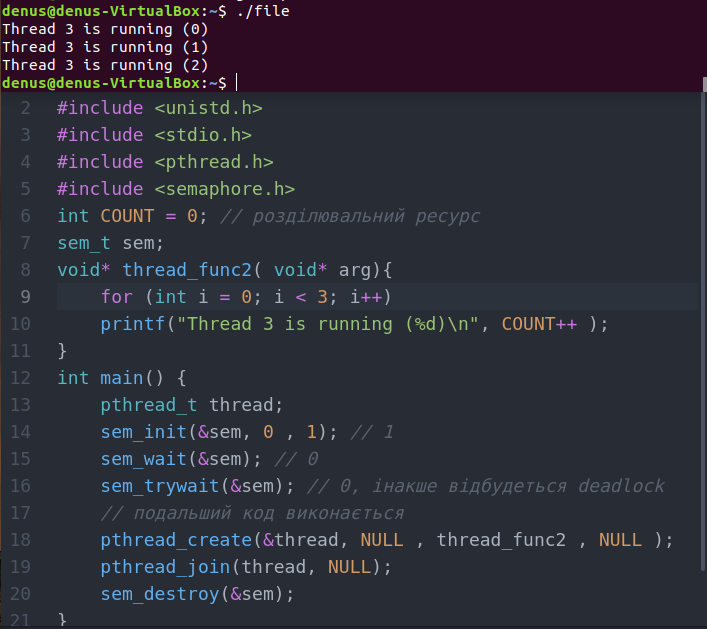
Приєднання до головного потоку потоку main\_thread до якого приєднані інші 5 потоків(threads)

Робота mutex з розділеним ресурсом COUNT. Потоки з одним mutex працюють послідовно.

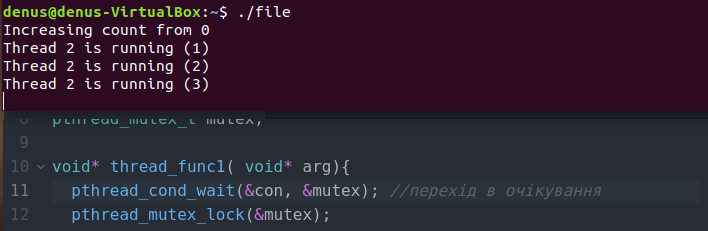
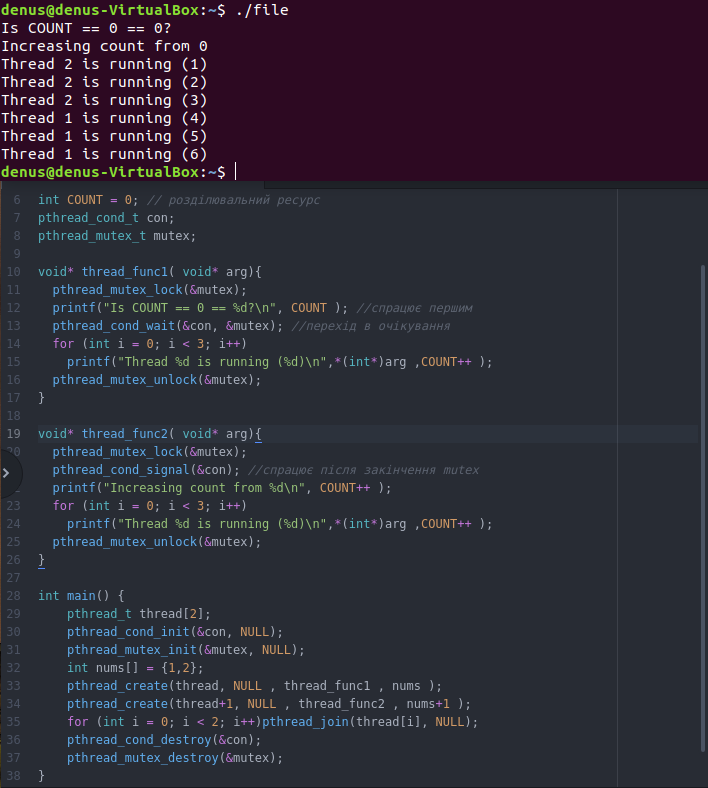
По суті 2 паралельні потоки : перший потік – це два потоки об’єднані mutex. Другий потік відбирає роздільний ресурс у першого потоку.

Робота з Семафором. Гра зі значенням семафора

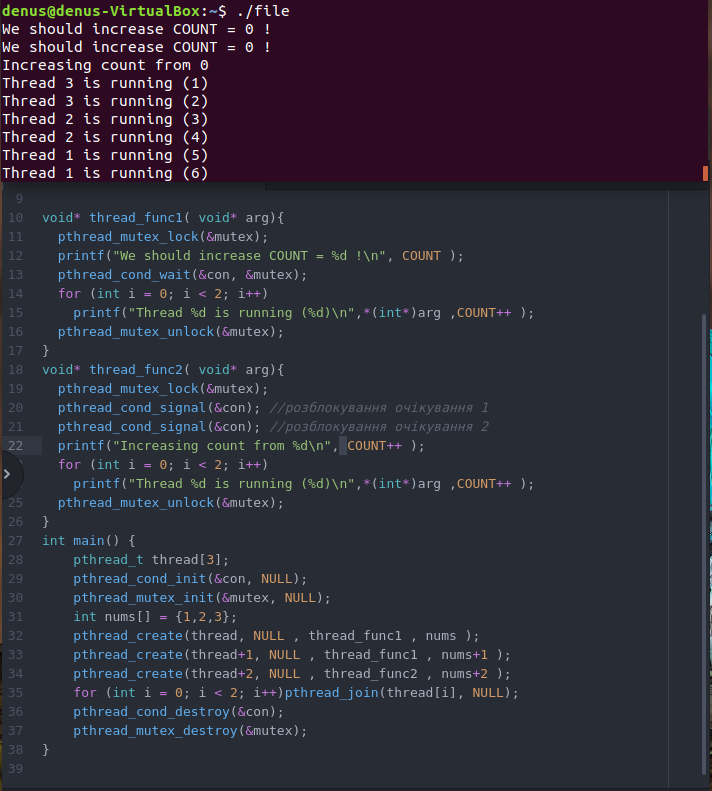
Існує певна потреба аби назначати значення Семафору інакше потрібно прописувати цикл з sem\_post()

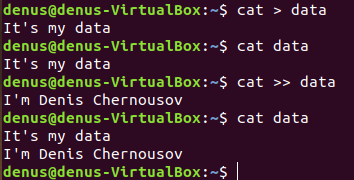
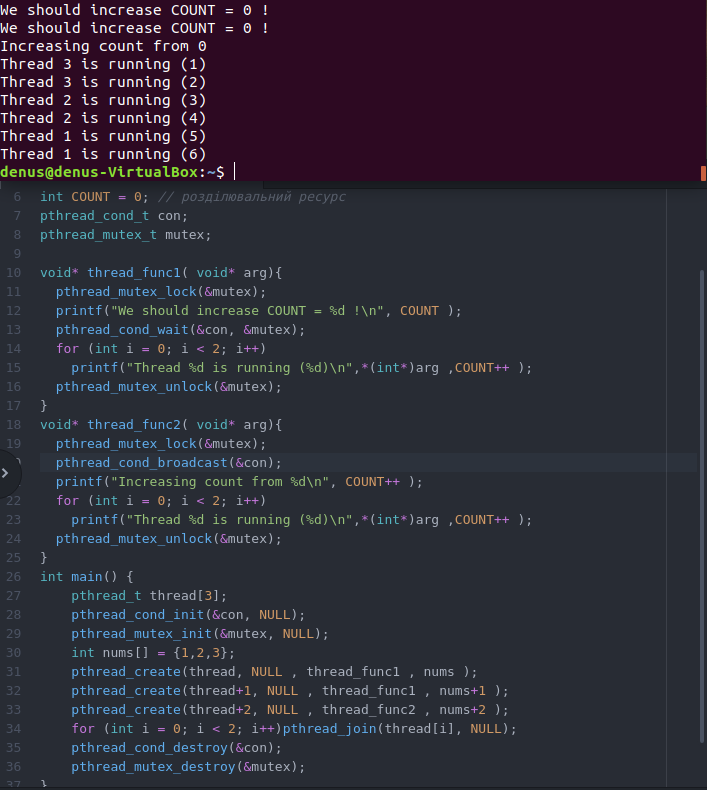
sem\_trywait() понижує значення Семафору

Проте ця функція не блокує Семафор, тобто не понижує значенння Семафору нище 0

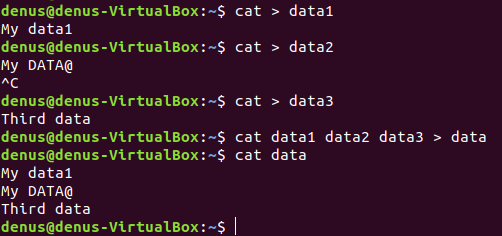
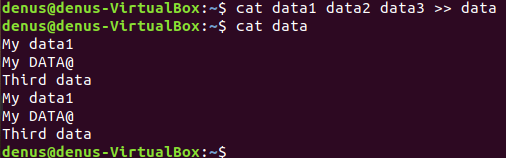
Використання умовної зміни разом з mutex

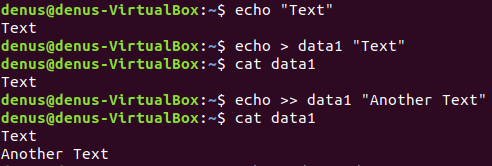
При використанні умовної змінної поза mutex процес попадає у вічне очікування

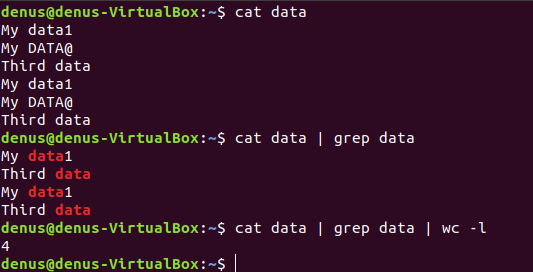
Можна розблокувати усі очікування за допомогою прорису cond\_signal

Або можна розблокувати усі очікування через broadcast

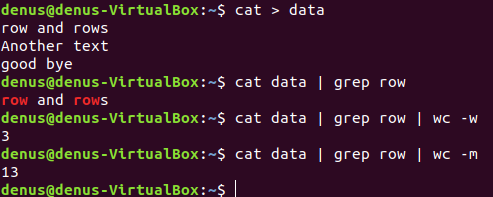
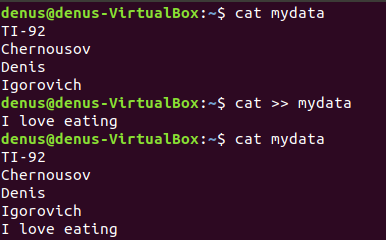
Використання потоку вводу і виводу аби заповнити файл

Конкатенація декількох файлів в один

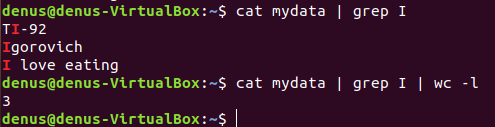
Дописування файлу даними

Використання echo

Перенаправлення потоку через канал з програми cat в grep і далі в wc

Використання каналів. Ключ -w виводить кількість слів. Ключ -w виводить кількість символів і кінець рядка

Створення персонального файлу



Використання в ньому перенаправлення потоку

**Висновок**

Під час роботи над лабораторною роботою були здобуті навички створення і управління потоками в ОС Linux. Були опануванні на практиці й вивчені команди для синхронізації потоків, а саме м’ютекси, семафори і умовні змінні*.* Також здобуті навички роботи з перенаправлення потоків та каналами, перенаправлення потоків між програмами.