Міністерство освіти і науки України

НТУУ «КПІ ім. Ігоря Сікорського»

Навчально-науковий інститут атомної та теплової енергетики

Кафедра цифрових технологій в енергетиці

Лабораторна робота №7 з дисципліни «Операційна система UNIX»

**Потоки, перенаправлення потоків**

**Варіант: 16**

Виконав студент групи ТР-12

Каркушевський Владислав

Перевірила д.т.н., проф. Левченко Л. О.

КИЇВ 2022

**Мета роботи :**

- Ознайомлення з системними викликами для управління потоками в ОС Linux

- Перенаправлення потоків при роботі файлами і командами.

**Теоретична частина**:

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\_.

При роботі з потоками використовуються основні функції:

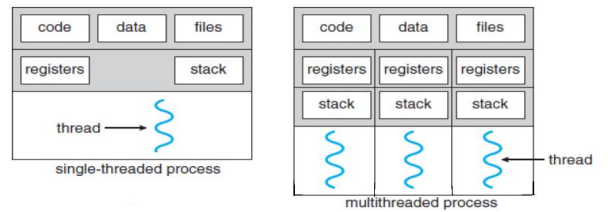
- створення потоку;

- блокування роботи потоку в очікування завершення іншого;

- дострокове завершення потоків;

- завершення роботи потоків.

Якщо процес містить тільки один потік, то в процесі знаходиться лише один виконуваний елемент і тільки одна задача виконується в одиницю часу. Такі процеси можна назвати однопоточними. Це класичні процеси UNIX. Якщо процес містить більше одного потоку, значить одночасно виконується кілька дій. Такі процеси називаються багатопоточними.



**Синхронізація потоків**

Усі потоки виконуються в одному адресному просторі. У зв'язку з цим постає проблема спільного використання загальних змінних, доступу до певного ресурсу, оскільки в один момент часу тільки єдиний потік повинен працювати з певним розділюваним ресурсом. Така задача має назву забезпечення взаємовиключення, а ділянки програмного коду, в яких потоки виконують операції з розділюваними ресурсами, називаються критичними секціями. З іншого боку можлива ситуація, коли потоку для продовження своєї роботи потрібно результат виконання іншого потоку, що потребує синхронізації дій. Для узгодженості взаємодії потоків розроблені засоби синхронізації потоків:

**м'ютекси (взаємні виключення), семафори і умовні змінні.**

Синхронізація і взаємовиключення забезпечуються за рахунок атомарности виконуваних операцій над м'ютексів і семафора. Атомарної називають операцію, яка не може бути перервана в ході свого виконання.

**М’ютекс** дозволяє потокам управляти доступом до даних. При використанні м’ютекса тільки один потік в певний момент часу може заблокувати м’ютекс і отримати доступ до ресурсу (право на його використання). По завершенні роботи з ресурсом потік повинен повернути це право, розблокувавши м’ютекс. Якщо будь-який потік звернеться до вже заблокованого м’ютексу, то він буде змушений чекати розблокування м’ютекса потоком, який їм володіє.

**Семафор** призначений для синхронізації потоків щодо дій та даних. Семафор - це захищена змінна, значення якої можна опитувати і міняти тільки за допомогою спеціальних операцій P і V і операції ініціалізації. Семафор може приймати ціле невід'ємне значення. При виконанні потоком операції P над семафором S значення семафора зменшується на 1 при S> 0 або потік блокується, «чекаючи на семафорі», при S=0. При виконанні операції V(S) відбувається пробудження одного з потоків, які очікують на семафорі S, а якщо таких немає, то значення семафора збільшується на 1. Як випливає з вищесказаного, при вході в критичну секцію потік повинен виконувати операцію P, а при виході з критичної секції операцію V.

**Умовна змінна** дозволяє потокам очікувати виконання деякої умови (події), пов'язаної з розділюваними даними. Над умовними змінними визначені дві основні операції: інформування про настання події і очікування події. При виконанні операції «інформування» один з потоків, які очікують на умовну змінну, відновлює свою роботу. Умовна змінна завжди використовується спільно з м'ютексів. Перед виконанням операції «очікування» потік повинен заблокувати м'ютекс. При виконанні операції «очікування» зазначений м’ютекс автоматично розблокується. Перед відновленням очікує потоку виконується автоматичне блокування м’ютекса, що дозволяє потоку увійти в критичну секцію, після критичної секції рекомендується розблокувати м’ютекс. При подачі сигналу іншим потокам рекомендується так само функцію «сигналізації» захистити м’ютексом.

**Перенаправлення введення/виведення в** **Linux**

Стандартні потоки введення і виведення в Linux є одним з найбільш поширених засобів для обміну інформацією процесів, а перенаправлення «>», «>>» і «|» є однією з найбільш популярних конструкцій командного інтерпретатора. Введення і виведення розподіляється між трьома стандартними потоками:

stdin - стандартне введення (клавіатура), номер потоку - 0;

stdout - стандартне виведення (екран), номер потоку - 1;

stderr - стандартна помилка (виведення помилок на екран), номер потоку - 2.

Зі стандартного введення команда може тільки зчитувати дані, а два інших потоки можуть використовуватися тільки для запису. Дані виводяться на екран і зчитуються з клавіатури, так як стандартні потоки за замовчуванням асоційовані з терміналом користувача. У командному інтерпретаторі bash така операція називається перенаправленням:

*<file* - використовувати файл як джерело даних для стандартного потоку введення,

*>file* - направити стандартний потік виведення в файл; якщо файл не існує, він буде створений, якщо існує - перезаписаний зверху;

*2>file* - направити стандартний потік помилок в файл; якщо файл не існує, він буде створений, якщо існує - перезаписаний зверху;

*>>file* - направити стандартний потік виведення в файл; якщо файл не існує, він буде створений, якщо існує - дані будуть дописані до нього в кінець;

*2>>file* - направити стандартний потік помилок в файл; якщо файл не існує, він буде створений, якщо існує - дані будуть дописані до нього в кінець;

*&>file або> &file* - направити стандартний потік виведення і стандартний потік помилок в файл; інша форма запису:> file 2>&1.

Стандартне введення - стандартний вхідний потік передає дані від користувача до програми.

*cat> myfile* - введення тексту з клавіатури у файл, після введення кожного рядка натискається Enter, по завершенні введення тексту натискається Ctrl + D, що означає кінець файлу EOF.

*cat myfile* - виведення інформації на екран з файлу.

*cat file1 >> file2* - дописати вміст файлу file1 у файл file2.

Команда cat зазвичай використовується для об'єднання вмісту файлів.

*cat file1 file2 file3 > file4* команда об’єднання трьох файлів в один файл file4.

Стандартне виведення - стандартний вихідний потік не перенаправляється в який небудь файл, а виводить текст на дисплей терміналу. Команда echo виводить на екран будь-який аргумент (текст або значення змінних), який передається йому в командному рядку: echo Example.

*echo> file1 "текст"* - перенаправлення виведення за допомогою символу ">", якщо файл із таким ім'ям вже існує, то він буде перезаписаний;

*echo>>file2 "текст додається"* - перенаправлення виведення за допомогою символу ">>", новий текст буде додано в кінець файлу;

*echo "текст на принтер" | lр* - передача стандартного виведення однієї команди на стандартний вхід іншої за допомогою символу "|", текст необхідно роздрукувати на принтері.

**Канали**. Канали використовуються для перенаправлення потоку з однієї програми в іншу. Особливим варіантом перенаправлення виведення є організація програмного каналу (іноді називає трубопроводом або конвеєром, оператор «|»). Для цього дві або декілька команд, таких, як виведення попередньої слугує введенням для наступної та розділяються символом вертикальної риски – «|». При цьому стандартний вихідний потік команди, розташованої ліворуч від символу «|», направляється на стандартне введення програми, розташованої праворуч від символу.

Програмні канали використовуються для того, щоб скомбінувати кілька маленьких програм, кожна з яких виконує тільки певні перетворення над своїм вхідним потоком, для створення узагальненої команди, результатом якої буде якесь більш складне перетворення.

**Завдання:**

1. Опанувати команди по роботі з потоками.

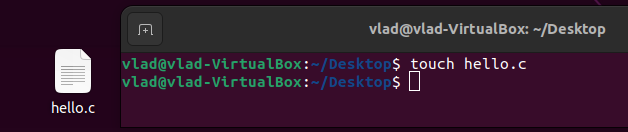
2. Ознайомитися із засобами синхронізації потоків.

3. Ознайомитися із стандартними потоками введення/виведення.

4. Підготувати звіт для викладача про виконання лабораторної роботи і представити його.

**Хід виконання роботи**

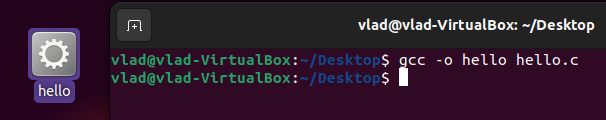
Для початку створимо найпростішу програму. Для цього створимо текстовий файл hello.c за допомогою команди touch.

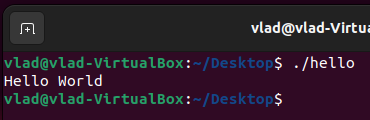


У файлі імпортуємо бібліотеку stdio.h - заголовковий файл стандартної бібліотеки мови C, що містить визначення макросів, константи та оголошення функцій та типів, що використовуються для різних операцій стандартного введення та виведення. У головному методі виведемо у консоль “Hello World”



Далі скомпілюємо наший код та запустимо виконуваний файл

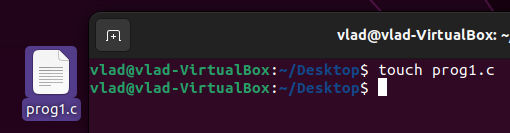




**Використати вихідні тексти, наведених вище п’яти прикладів для потоків, м’ютексів та семафорів, для створення програм.**

**Приклад №1**

Для початку, створимо файл prog1.c, у якому буде зберігатися вихідний код нашої програми. Створюємо пустий файл за допомогою команди touch.



На початку імпортуються усі потрібні бібліотеки. Створюється функція thread\_func(). У тілі функції змінюється стан потоку, та за допомогою циклу виводить 4 рази текст “I`m still running!”. Функція pthread\_setcancelstate() встановлює стан відміни викликаючого потоку в значення, задане в state. Попередній стан відміни потоку повертається в буфері на який вказує oldstate. Аргумент state повинен мати одне з наступних значень:

pthread\_cancel\_enable **-** потік можна відмінити. Це стан за замовчуванням стан відміни у всіх нових потоках, включаючи початкового потоку. Тип відміни потоку визначає, коли скасований потік буде відповідати на запит на скасування.

pthread\_cancel\_disable - потік не є відмінюваним. Якщо отримано запит на скасування то він блокується до тих пір, поки не буде включена можливість відміни.

У кінці викликається pthread\_testcancel(). Виклик pthread\_testcancel() створює точку відміни в межах викликаючого потоку, так що потік, який в іншому випадку виконує код, який не містить точок відміни, буде відповідати на запит відміни.

У головному методі main() створюється потік за допомогою функції pthread\_create(). Ця функція визначена в заголовки .

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

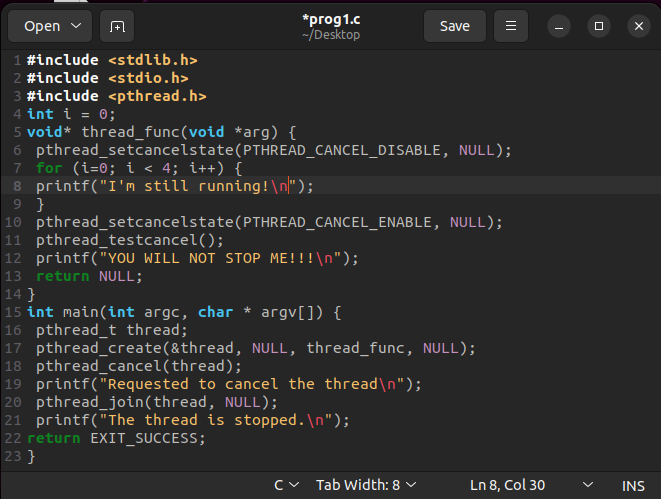
Другий параметр є вказівником на змінну типу pthread\_attr\_t , використовується для установки атрибутів потоку. Цей об'єкт управляє деталями взаємодії потоку з іншою програмою. Якщо параметр дорівнює NULL, то потік буде створений з атрибутами за замовчуванням.

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.

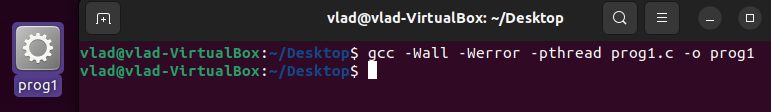
Четвертий параметр функції pthread\_create має тип void\*. Цей параметр може використовуватися для передачі значення як аргумент у функцію потоку. Через нього можна передавати новому потоку параметри.

Після виклику pthread\_create функція потоку буде запущена на виконання паралельно з іншими потоками програми. Після створення викликається функція pthread\_cancel(), яка відповідає за дострокове завершення потоків.

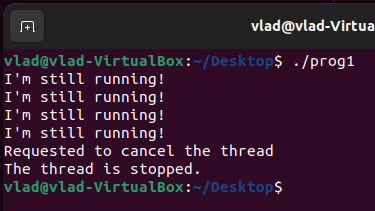
Також, використовується функція pthread\_join(), яка відповідає за блокування роботи потоку. Після успішного виконання викликаючий потік блокується до тих пір, поки потік, вказаний як thread, не завершиться (якщо потік thread вже завершено, функція pthread\_join() повертається негайно). Як тільки thread 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.



Далі потрібно скомпілювати код. Прототипи функцій роботи з потоками і необхідні типи даних містяться в заголовки . Ці функції не включені в стандартну бібліотеку мови С, вони знаходяться в бібліотеці libthread. Тому в командному рядку для gсс необхідно додати опцію «- pthread». У результаті код скомпілюється, і створиться виконуваний файл.

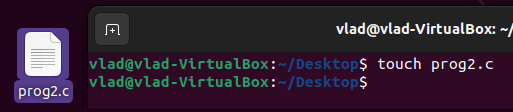
****

Для того аби запустити виконуваний файл пропишемо ./prog1. У результаті виведеться результат програми

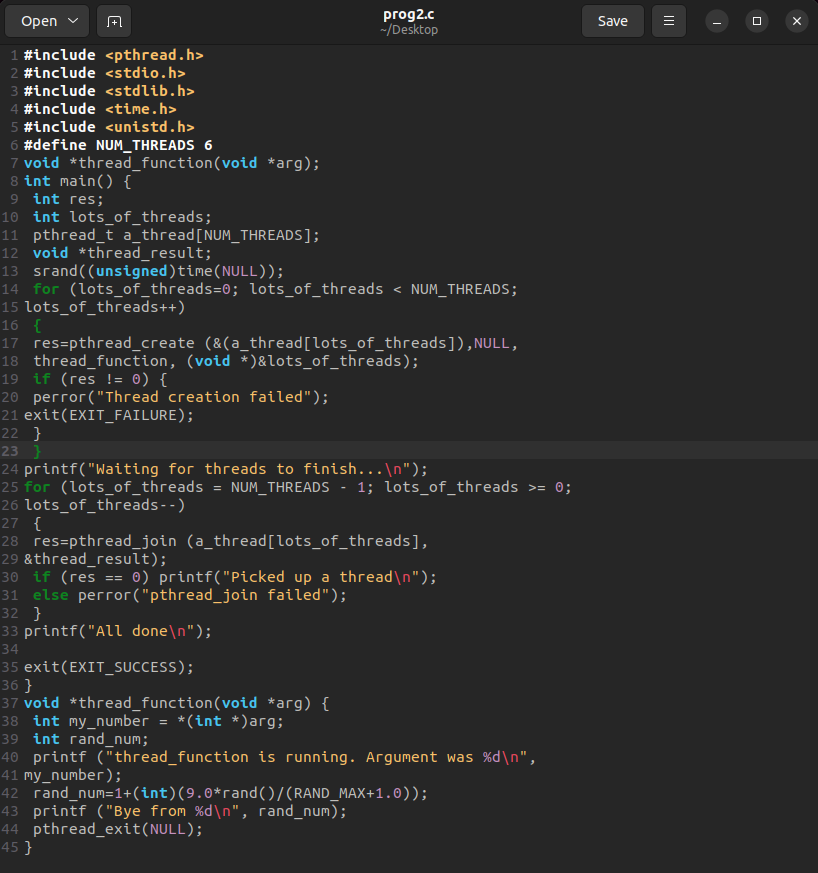
****

**Приклад №2**

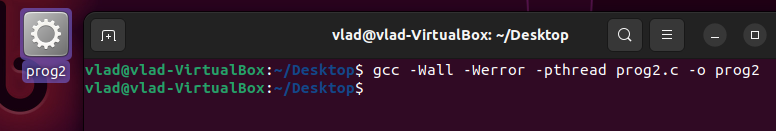
Для початку, створимо файл prog2.c, у якому буде зберігатися вихідний код нашої програми. Створюємо пустий файл за допомогою команди touch.

****

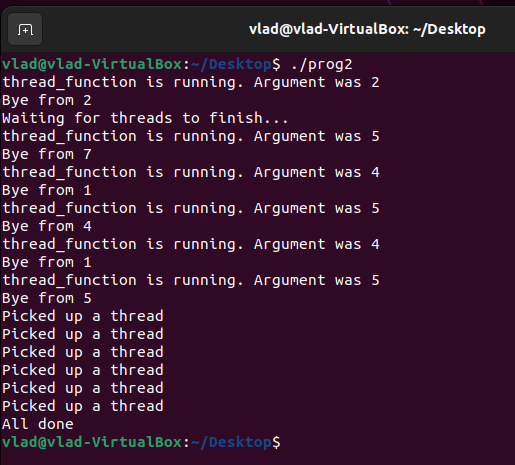
На початку імпортуються усі потрібні бібліотеки. Створюються змінні. У першому циклі створюються потоки за допомогою команди pthread\_crate(). Також у циклі перевіряється умова: чи після створення потоку результат не дорівнює нулю. Якщо не дорівнює, то це означає що при створенні потоку сталася помилка, і процес закривається. У наступному циклі блокується робота попередньо створених потоків за допомогою функції pthread\_join(). Також у циклі відбувається перевірка, на успішність виконання функції pthread\_join(). Якщо все успішно, виводиться текст “Picked up a thread”. Вкінці програми викликається exit(EXIT\_SUCSESS) що означає успішне виконання програми.

****

Скомпілюємо код

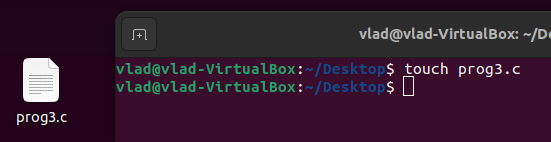
****

Та запустимо виконуваний файл, прописавши ./prog2. У результаті виведеться результат програми.

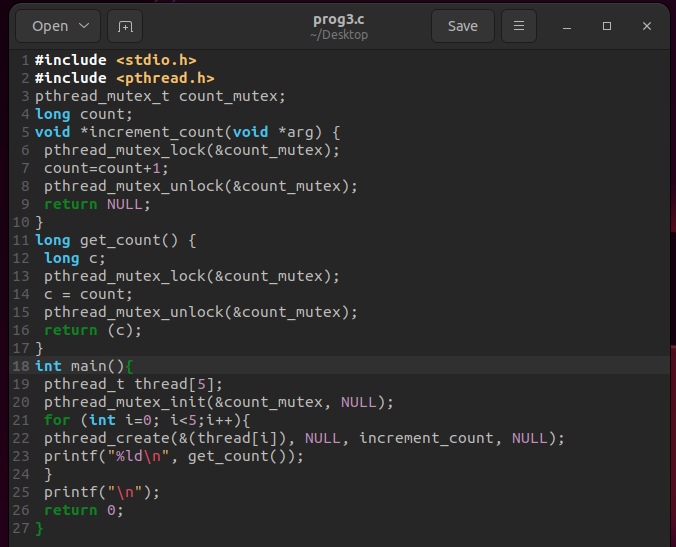
****

**Приклад №3**

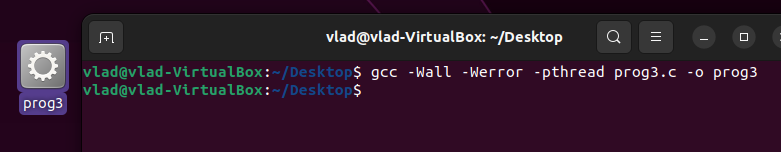
Для початку, створимо файл prog3.c, у якому буде зберігатися вихідний код нашої програми. Створюємо пустий файл за допомогою команди touch.

****

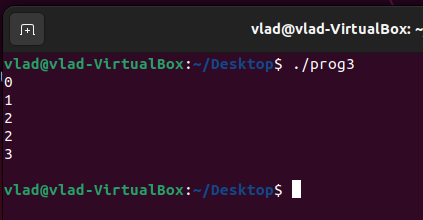
На початку імпортуються усі потрібні бібліотеки. Оголошуються змінні та створюються функції. Перша функція є функція increment\_count(), вона використовує м’ютекс, щоб гарантувати атомарность (цілісність) модифікації розділюваної змінної count. Наступна функція get\_count() використовує м’ютекс, щоб гарантувати, що змінна count атомарному зчитується. У методі main за допомогою циклу створюються та запускаються потоки, і результат виводиться у консоль

****

Скомпілюємо код

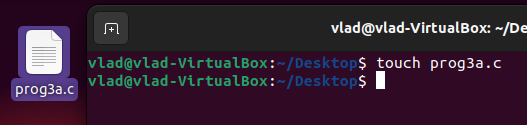
****

Та запустимо виконуваний файл, прописавши ./prog3. У результаті виведеться результат програми.

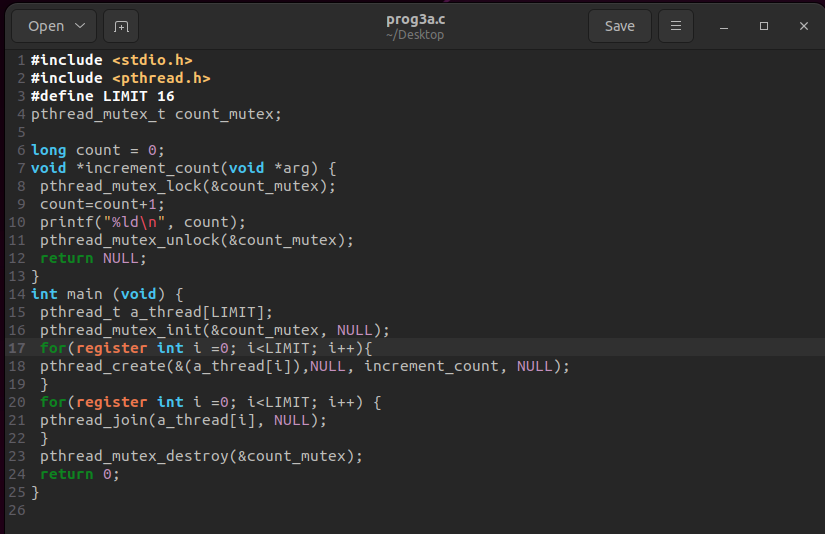
****

**Приклад №3а**

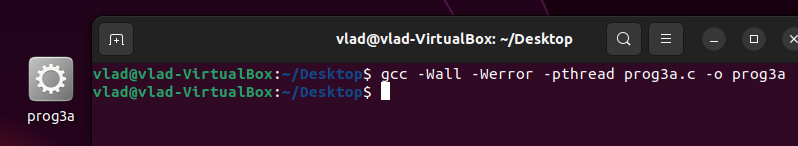
Для початку, створимо файл prog3а.c, у якому буде зберігатися вихідний код нашої програми. Створюємо пустий файл за допомогою команди touch.

****

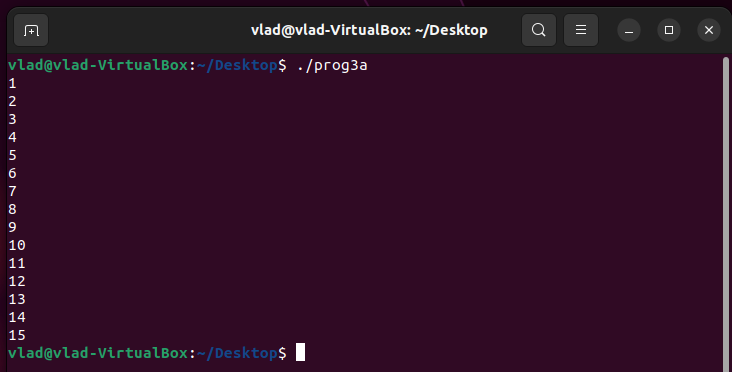
На початку імпортуються усі потрібні бібліотеки. Оголошуються змінні та створюються функції. Перша функція є функція increment\_count(), вона використовує м’ютекс, щоб гарантувати атомарность (цілісність) модифікації розділюваної змінної count та виводить змінну count у консоль. У методі main за допомогою циклу створюються та запускаються потоки. У наступному циклі блокується робота попередньо створених потоків за допомогою функції pthread\_join(). Та вкінці викликається функція pthread\_mutex\_destroy() яка знищує м’ютекс.

****

Скомпілюємо код

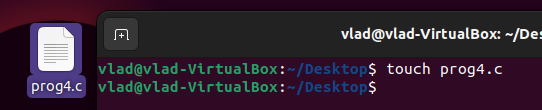
****

Та запустимо виконуваний файл, прописавши ./prog3а. У результаті виведеться результат програми.

****

**Приклад №4**

Для початку, створимо файл prog4.c, у якому буде зберігатися вихідний код нашої програми. Створюємо пустий файл за допомогою команди touch.

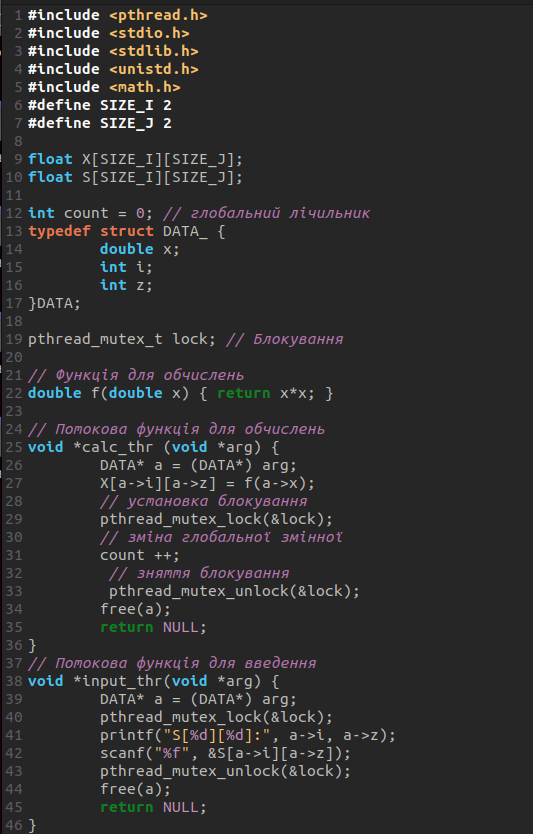
****

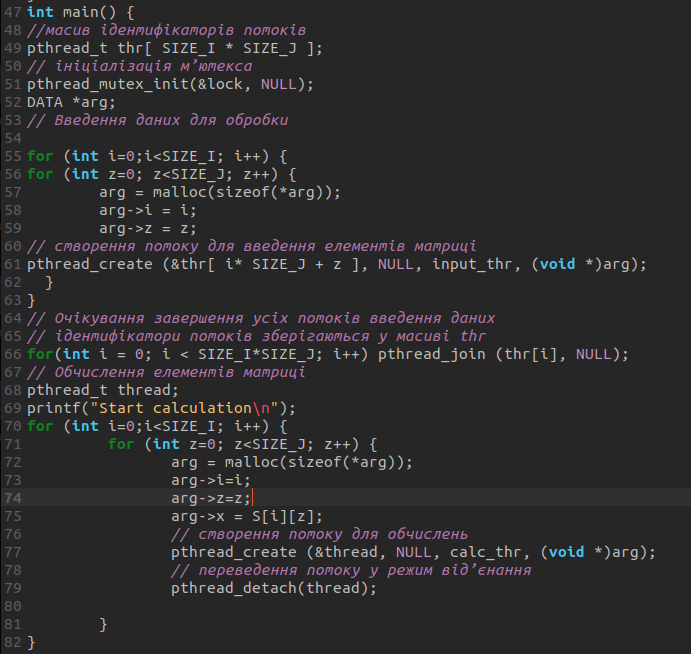
На початку імпортуються усі необхідні бібліотеки. Створюються константи, такі як : SIZE\_I, SIZE\_J. Створюються дві матриці: X , S. Також створюється глобальний лічильник. Оголошується структура DATA\_ .

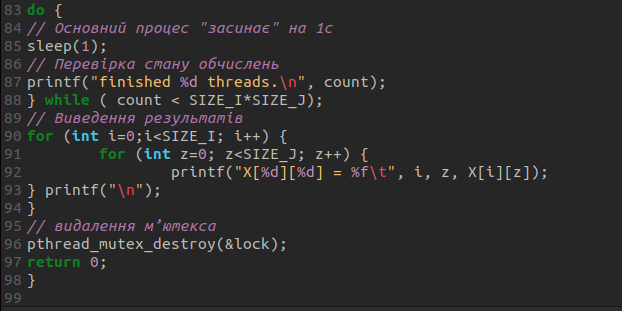
Перша функція f() створена для обчислень, а саме піднесення до квадрату числа, переданого у параметр.

Наступна функція calc\_thr() є потоковою функцією для обчислення. У ній змінюються значення у матриці.

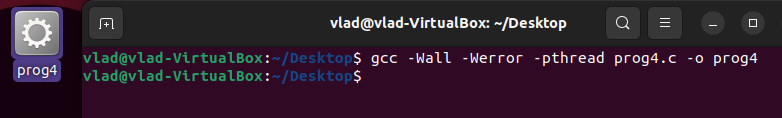
У головній функції main() оголошується масив ідентифікаторів потоків, ініціалізується м’ютекс та за допомогою циклу відбувається введення даних з консолі для обробки та створення потоку для введення елементів матриці. У наступному циклі відбувається обчислення елементів матриці. У середині циклу передаються значення, створюється потік для обчислень та переведення потоку у режим від’єднання. Після завершення циклу створюється цикл do while який перевіряє стан обчислень, і якщо обчислення не закінчені то основний процес “засинає на 1 секунду”. Далі створюється цикл у якому відбувається виведення результатів. У кінці програми відбувається видалення м’ютекса.

****

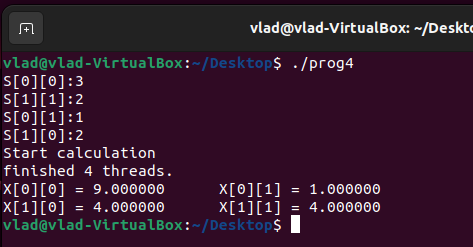
****

****

Скомпілюємо код

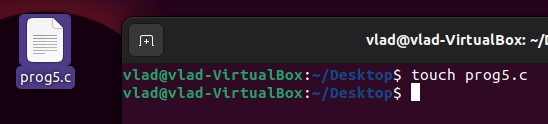
****

Та запустимо виконуваний файл, прописавши ./prog3а. У результаті потрібно ввести 4 значення у відповідну комірку матриці. І вкінці виведуться введені числа у квадраті.

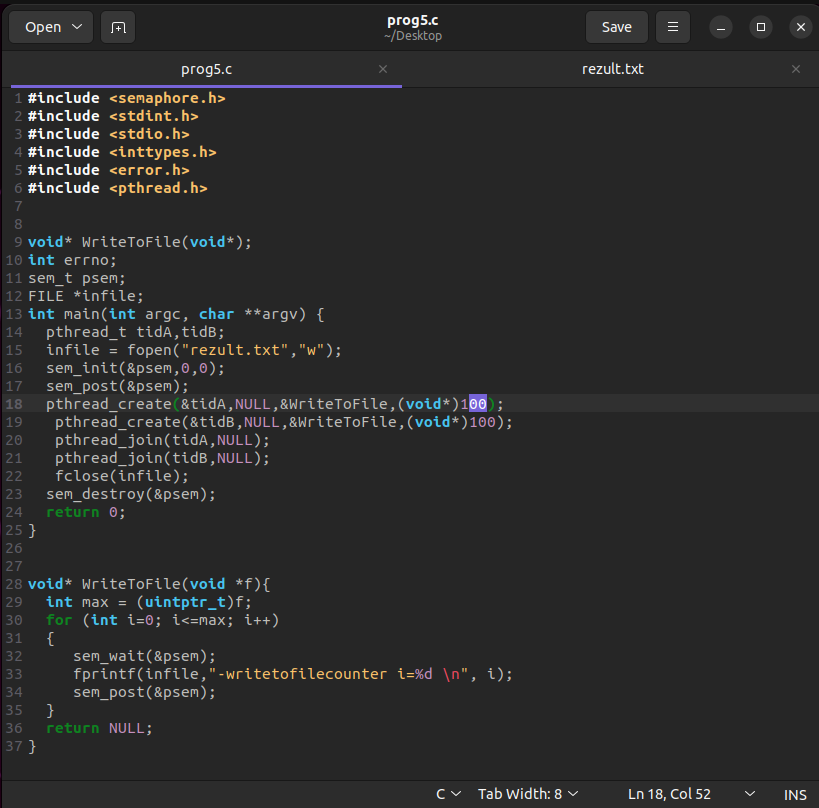
****

**Приклад №5**

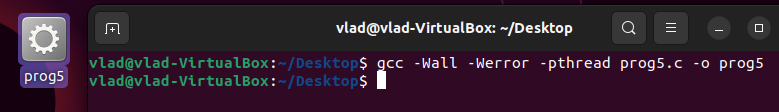
Для початку, створимо файл prog5.c, у якому буде зберігатися вихідний код нашої програми. Створюємо пустий файл за допомогою команди touch.

****

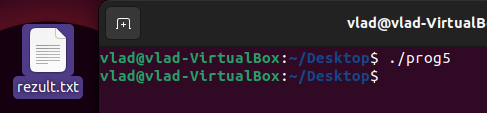
На початку імпортуються усі необхідні бібліотеки. Створюються змінні. Створена функція WriteToFile() у якій використані функції sem\_wait - «очікування на семафорі». Виконання потоку блокується до тих пір, поки значення семафора не стане позитивним. При цьому значення семафора зменшується на 1. Також використовується функція sem\_post - збільшує значення семафора psem. У головному методі main() ініціалізується семафор за допомогою функції sem\_init. Створюються потоки та після створення блокується робота потоку. Та вкінці викликається функція sem\_destroy яка знищує семафор.

****

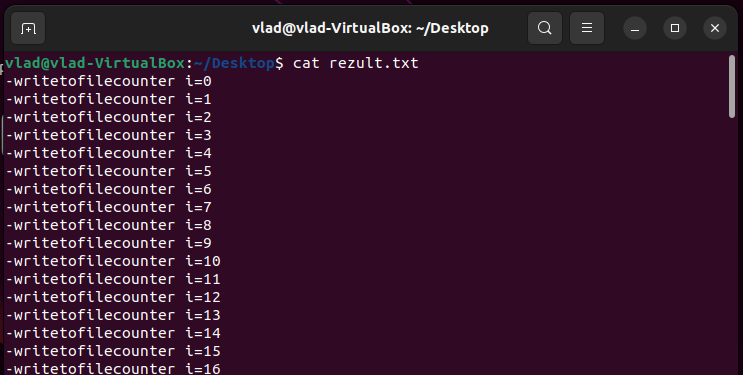
Скомпілюємо код

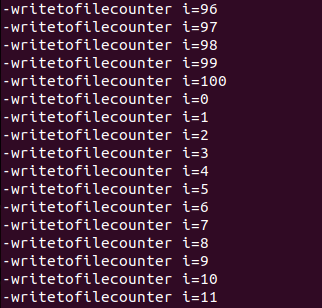
****

Та запустимо виконуваний файл, прописавши ./prog3а. У результаті створиться файл result.txt .

****

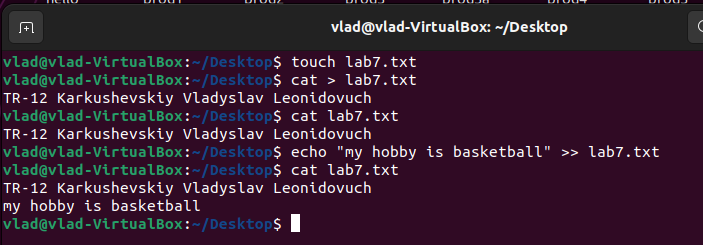
Переглянемо вміст файлу result.txt використавши команду cat**.**

****

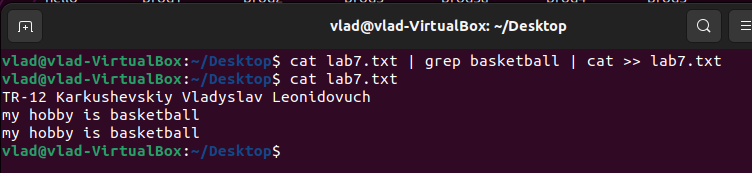
****

**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.**

Першим ділом створимо порожній файл lab7.txt за допомогою команди touch. Далі введемо cat > lab7.txt та заповнимо наший файл інформацією. Для збереження пропишемо Ctrl + D. Переглянемо вміст файлу. За допомогою команд перенаправлення, добавимо до нашого файлу інформацію про хоббі.

****

Також створимо канал, у якому будемо шукати рядок зі значенням “basketball” у файлі lab7.txt та перенаправляти його у той самий файл. Також пропишемо cat lab7.txt аби переглянути результат.

****

**Висновок:** На цій лабораторній роботі було ознайомлено з теорію по лабораторній роботі. Опановано команди по роботі з потоками. Ознайомлено із засобами синхронізації потоків та із стандартними потоками введення/виведення. Використано вихідні тексти, наведених прикладів для потоків, м’ютексів та семафорів, для створення програм. Проаналізовано результати роботи програм та їх особливості. Створено файл з особистими даними (група, прізвище, ім’я, по-батькові), використовуючи стандартне введення з клавіатури. Створено канал.