МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ "ЛЬВІВСЬКА ПОЛІТЕХНІКА"

ІКНІ
Кафедра **ПЗ**

ЗВІТ

До лабораторної роботи №6
на тему: “Багатопоточність в операційній системі Linux. Створення, керування та синхронізація потоків”
з дисципліни: “Операційні системи”

Лектор:
старший викладач кафедри ПЗ
Грицай О.Д.

Виконав:
студент групи ПЗ-24
Губик А. С.

Прийняв:
доцент кафедри ПЗ
Горечко О. М.

Львів – 2023

Тема роботи:Багатопоточність в операційній системі Linux. Створення, керування та синхронізація потоків

Мета роботи:Навчитися створювати потоки та керувати ними в операційній системі Linux. Ознайомитися з методами синхронізації потоків в операційній системі Linux. Навчитися реалізовувати багатопоточний алгоритм розв'язку задачі з використанням синхронізації в операційній системі Linux.

Теоретичні відомості

Потоки в операційній системі Linux. Поняття потоку, як одиниці виконання процесу в операційній системі (ОС) Linux, аналогічне як і у ОС Windows. Кожен процес можна представити як контейнер ресурсів і послідовність інструкцій виконання. Таким чином, можна сказати, що кожен процес містить хоча б один потік виконання, якому надані лічильник інструкцій, регістри і стек. Крім того, у кожному процесі можна створити додаткові гілки виконання – потоки, тоді такий процес називають багатопоточним. Різниця між потоками в ОС Linux і в ОС Windows полягає у їх представленні в ядрі операційних систем. В ОС Windows потоки виконання у режимі користувача зіставляються з потоками у режим ядра. У перших версіях ядра Linux потоки користувача зіставлялись з процесами у ядрі. Створення потоку відбувалось з допомогою системного виклику clone(). Виклик clone(), як і fork() дозволяє створювати новий процес, але з певними відмінностями : - одразу повне копіювання батьківського процесу - створення власного стеку - необхідно вказати спеціальний набір прапорців успадкування для того, щоб визначити, як будуть розподілятися ресурси (адресний простір, відкритті файли, обробники сигналів) між батьківським і дочірнім процесом. Таким чином, створювався новий потік у режимі користувача, який відображається у процес ядра. Відображення здійснюється за моделлю 1:1. Оскільки керуючий блок процесу в Linux представлений в ядрі структурою Фактично у ядрі потоки і процеси не розрізнялися. Але системний виклик clone() не підтримувався стандартом POSIX і тому розроблялись бібліотеки потоків, що дозволяли працювати з потоками, використовуючи clone() з різними атрибутами виконання. У сучасних версіях Linux підтримуються спеціальні об'єкти ядра - потоки ядра, побудованні на зміненному і розширеному системному виклику clone(). Підтримка потоків здійснюється через POSIX-сумісну бібліотеку NPTL (Native POSIX Threads Library). Типи даних і функції, що застосовуються до потоків

Індивідуальне завдання

3. Бінарний пошук заданого елементу масиву (кількість елементів >10000, елементи рандомні). Вивести значення та індекс. (Синхронізація: м'ютекс, умовні змінні)

I. Розпаралелення бінарного пошуку на 2, 4, 8, 16 потоків

```
1934  g++ main.cpp BinarySearch.cpp HandleInput.cpp -DPROP_SYNC -o main
1935  ./main
1936  ar rcs stuff.a BinarySearch.cpp HandleInput.cpp
1937  rm stuff.a
1938  ar rcs stuff.a BinarySearch.cpp HandleInput.cpp
1939  gcc main.cpp stuff.a -o main
1940  gcc main.cpp -Lstuff.a -o main
1941  ar rcs  BinarySearch.cpp HandleInput.cpp
1942  ar rcs stuff.lib  BinarySearch.cpp HandleInput.cpp
1943  rm stuff.lib
1944  ar rcs stuff.a BinarySearch.cpp HandleInput.cpp
1945  gcc -c BinarySearch.cpp -o BinarySearch.o
1946  gcc -c HandleInput.cpp -o HandleInput.o
1947  ar rcs stuff.a BinarySearch.o HandleInput.o
1948  gcc main.cpp stuff.a -o main
1949  ranlib *.a
1950  gcc main.cpp stuff.a -o main
1951  gcc main.cpp -L. -lstuff -o main
1952  gcc main.cpp -L. -lstuff.a -o main
1953  ar rcs libstuff.a BinarySearch.o HandleInput.o
1954  gcc main.cpp -L. -lstuff.a -o main
1955  gcc main.cpp -L. -lstuff -o main
1956  ran libstuff.a
1957  ranlib libstuff.a
1958  gcc main.cpp -L. -lstuff -o main
1959  gcc -c main.cpp -o main.o
1960  gcc main.o -L. -lstuff -o main
1961  g++ -c BinarySearch.cpp -o BinarySearch.o
1962  g++ -c HandleInput.cpp -o HandleInput.o
1963  g++ main.cpp -L. -lstuff -o main
1964  ./main
1965  g++ main.cpp libstuff.a -o main
1966  g++ main.cpp -I./Source/Include -L./StaticLib -lstuff -o main
1967  g++ main.cpp -I./Source/Include -LStaticLib -lstuff -o main
1968  g++ main.cpp -ISource/Include -LStaticLib -lstuff -o main
1969  g++ ./Source/main.cpp -ISource/Include -LStaticLib -lstuff -o main
1970  g++ -c Source/HandleInput.cpp -o HandleInput.o
1971  g++ -c Source/BinarySearch.cpp -o BinarySearch.o
1972  g++ -shared -fPIC HandleInput.o BinarySearch.o -o libdll.so
1973  g++ HandleInput.o BinarySearch.o -shared -fPIC -o libdll.so
1974  g++ -c -fPIC Source/BinarySearch.cpp -o BinarySearch.o
1975  g++ -c -fPIC Source/HandleInput.cpp -o HandleInput.o
1976  g++ HandleInput.o BinarySearch.o -shared -fPIC -o libdll.so
1977  g++ Source/main.cpp libdll.so -o dllmain
```

Рис. 1: Графік залежності швидкості роботи програми від кількості потоків

Рис. 2: Графік залежності швидкості роботи програми від кількості потоків

Як бачимо синхронізація сповільнює роботу і не впливає на правильність виконання бінарного пошуку.

II. Задання пріоритету, відміна та affinity

Рис. 3:

Висновок: Я ознайомився з бібліотекою POSIX Threads. Ефективність синхронізації залежить від поставленої перед нами задачі. У випадку з бінарним пошуком воно не потрібне і сповільнює роботу.