Міністерство освіти і науки України
Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Навчально-науковий Фізико-технічний інститут

ОПЕРАЦІЙНІ СИСТЕМИ

Комп'ютерний практикум
Робота №7

Виконав:
студент групи ФІ-12
Завалій Олександр
Перевірив:
Кірієнко О.В.

Київ-2023

Робота №7.
Основи роботи з потоками у Linux з використанням бібліотеки pthread

Мета:

Оволодіння практичними навичками роботи з потоками POSIX у Linux з використанням бібліотеки pthread.

Варіант №5

Зміст індивідуального завдання:

1. **Створення потоку.** Напишіть програму, що створює потік. Застосуйте атрибути за умовчанням. Батьківський і дочірній потоки мають роздрукувати по десять рядків тексту.

2. **Очікування потоку.** Модифікуйте програму п. 1 так, щоб батьківський потік здійснював роздрукування після завершення дочірнього (функція pthread_join()).

3. **Параметри потоку.** Напишіть програму, що створює чотири потоки, що виконують одну й ту саму функцію. Ця функція має роздруковувати послідовність текстових рядків, переданих як параметр. Кожний зі створених потоків має роздруковувати різні послідовності рядків.

4. **Примусове завершення потоку.** Дочірній потік має роздруковувати текст на екран. Через дві секунди після створення дочірнього потоку, батіківський потік має перервати його (функція pthread_cancel()).

5. **Обробка завершення потоку.** Модифікуйте програму п.4 так, щоб дочірній потік перед завершенням роздруковував повідомлення про це (pthread_cleanup_push()).

Task I

Створення потоку. Напишіть програму, що створює потік. Застосуйте атрибути за умовчанням. Батьківський і дочірній потоки мають роздрукувати по десять рядків тексту.

```
alex@Oleksandr:~/labs/lab_7$ cat program.cpp
#include <stdio.h>
#include <pthread.h>
#include <iostream>
using namespace std;

void* thread_fun(void* arg) {
        for(int i = 1; i < 11; i++) {
                cout << "Child thread. Line num: " << i << endl;
        }
}

int main() {
        pthread_t child_thread;
        pthread_create(&child_thread, NULL, thread_fun, NULL);

        for(int i = 1; i < 11; i++) {
                cout << "Parent thread. Line num: " << i << endl;
        }

        pthread_join(child_thread, NULL);

        return 0;
}alex@Oleksandr:~/labs/lab_7$ g++ -pthread -o program1 program.cpp
```

```
alex@Oleksandr:~/labs/lab_7$ ./program1
Parent thread. Line num: 1
Parent thread. Line num: 2
Parent thread. Line num: 3
Parent thread. Line num: 4
Parent thread. Line num: 5
Parent thread. Line num: 6
Parent thread. Line num: 7
Parent thread. Line num: 8
Parent thread. Line num: 9
Parent thread. Line num: 10
Child thread. Line num: 1
Child thread. Line num: 2
Child thread. Line num: 3
Child thread. Line num: 4
Child thread. Line num: 5
Child thread. Line num: 6
Child thread. Line num: 7
Child thread. Line num: 8
Child thread. Line num: 9
Child thread. Line num: 10
alex@Oleksandr:~/labs/lab_7$
```

Task II

Очікування потоку. Модифікуйте програму п. 1 так, щоб батьківський потік здійснював роздрукування після завершення дочірнього (функція pthread_join()).

```
alex@Oleksandr:~/labs/lab_7$ cat program.cpp
#include <stdio.h>
#include <pthread.h>
#include <iostream>
using namespace std;

void* thread_fun(void* arg) {
        for(int i = 1; i < 11; i++) {
                cout << "Child thread. Line num: " << i << endl;
        }
}

int main() {
        pthread_t child_thread;
        pthread_create(&child_thread, NULL, thread_fun, NULL);

        pthread_join(child_thread, NULL);

        for(int i = 1; i < 11; i++) {
                cout << "Parent thread. Line num: " << i << endl;
        }

        return 0;
}alex@Oleksandr:~/labs/lab_7$ g++ -pthread -o program2 program.cpp
```

```
alex@Oleksandr:~/labs/lab_7$ ./program2
Child thread. Line num: 1
Child thread. Line num: 2
Child thread. Line num: 3
Child thread. Line num: 4
Child thread. Line num: 5
Child thread. Line num: 6
Child thread. Line num: 7
Child thread. Line num: 8
Child thread. Line num: 9
Child thread. Line num: 10
Parent thread. Line num: 1
Parent thread. Line num: 2
Parent thread. Line num: 3
Parent thread. Line num: 4
Parent thread. Line num: 5
Parent thread. Line num: 6
Parent thread. Line num: 7
Parent thread. Line num: 8
Parent thread. Line num: 9
Parent thread. Line num: 10
alex@Oleksandr:~/labs/lab_7$
```

Task III

Параметри потоку. Напишіть програму, що створює чотири потоки, що виконують одну й ту саму функцію. Ця функція має роздруковувати послідовність текстових рядків, переданих як параметр. Кожний зі створених потоків має роздруковувати різні послідовності рядків.

```
alex@Oleksandr:~/labs/lab_7$ cat program.cpp
#include <stdio.h>
#include <pthread.h>
#include <iostream>
#include <string>
using namespace std;


void* thread_fun(void* arg) {
        string sequence = (char*)arg;
        cout << "Thread: " << sequence << " Value: " << sequence << endl;
        return NULL;
}

int main() {
        pthread_t child_threads[4];

        const char* thread_variable[4] = {"One", "Two", "Three", "Four"};

        for(int i = 0; i < 4; i++) {
                pthread_create(&child_threads[i], NULL, thread_fun, (void*)thread_variable[i]);
        }

        for(int i = 0; i < 4; i++) {
                pthread_join(child_threads[i], NULL);
        }

        return 0;
}alex@Oleksandr:~/labs/lab_7$ g++ -pthread -o program3 program.cpp
```

```
alex@Oleksandr:~/labs/lab_7$ ./program3
Thread: Four Value: Four
Thread: Three Value: Three
Thread: One Value: One
Thread: Two Value: Two
alex@Oleksandr:~/labs/lab_7$
```

Task IV

Примусове завершення потоку. Дочірній потік має роздруковувати текст на екран. Через дві секунди після створення дочірнього потоку, батіківський потік має перервати його (функція pthread_cancel()).

```
alex@Oleksandr:~/labs/lab_7$ cat program.cpp
#include <stdio.h>
#include <pthread.h>
#include <iostream>
#include <string>
#include <unistd.h>
using namespace std;


void* thread_fun(void* arg) {
        string sequence = (char*)arg;
        cout << "Child thread. " <<" Value: " << sequence << endl;
        sleep(5);
        return NULL;
}

int main() {
        pthread_t child_thread;

        const char* thread_variable[] = {"Hello World!"};

        pthread_create(&child_thread, NULL, thread_fun, (void*)thread_variable[0]);

        sleep(2);

        if (!pthread_cancel(child_thread)) {
                cout << "pthread_cancel completed!" << endl;
        }

        pthread_join(child_thread, NULL);

        return 0;
}alex@Oleksandr:~/labs/lab_7$ g++ -pthread -o program4 program.cpp
```

```
alex@Oleksandr:~/labs/lab_7$ ./program4
Child thread.  Value: Hello World!
pthread_cancel completed!
alex@Oleksandr:~/labs/lab_7$
```

Task V

Обробка завершення потоку. Модифікуйте програму п.4 так, щоб дочірній потік перед завершенням роздруковував повідомлення про це (pthread_cleanup_push()).

```
alex@Oleksandr:~/labs/lab_7$ cat program.cpp
#include <stdio.h>
#include <pthread.h>
#include <iostream>
#include <string>
#include <unistd.h>
using namespace std;

void cleanup_fun(void* arg){
        cout << "Thread ended..." << endl;
}

void* thread_fun(void* arg) {
        string sequence = (char*)arg;
        cout << "Child thread. " <<" Value: " << sequence << endl;
        pthread_cleanup_push(cleanup_fun, NULL);
        sleep(5);
        pthread_cleanup_pop(1);
        return NULL;
}

int main() {
        pthread_t child_thread;

        const char* thread_variable[] = {"Hello World!"};

        pthread_create(&child_thread, NULL, thread_fun, (void*)thread_variable[0]);

        sleep(2);

        if (!pthread_cancel(child_thread)) {
                cout << "pthread_cancel completed!" << endl;
        }

        pthread_join(child_thread, NULL);

        return 0;
}alex@Oleksandr:~/labs/lab_7$ g++ -pthread -o program5 program.cpp
```

```
alex@Oleksandr:~/labs/lab_7$ ./program5
Child thread.  Value: Hello World!
pthread_cancel completed!
Thread ended...
alex@Oleksandr:~/labs/lab_7$
```

Висновки

Потоки дозволяють розпаралелити виконання програм, що дозволяє ефективніше використовувати ресурси системи. POSIX Threads або Pthread — стандарт POSIX реалізації потоків виконання, який визначає API для створення та управління. Ця бібліотека може допомогти уникнути гонки даних, блокування та взаємоблокування, тощо. Важливо правильно проектувати та розробляти багатопотокові програми.

Всі процедури Pthreads можуть бути розділені на 4 категорії за призначенням:

- Управляння потоками - створення, об'єднання потоків та інше.
- Mutex.
- Умовні змінні.
- Синхронізація потоків з використанням блокування (lock) і бар'єрів (barriers).

Використання цих механізмів може допомогти досягти коректної взаємодії між потоками та запобігти виникненню помилок.

Основні операції з потоками включають створення потоку «pthread_create()», завершення потоку «pthread_exit()», відміна потоку «pthread_cancel()», блокування потоку до завершення іншого потоку «pthread_join()», тощо.

Одним з ключових аспектів роботи з потоками є синхронізація доступу до спільних ресурсів:

- pthread_mutex_init (), pthread_mutex_destroy (), pthread_mutex_lock (), pthread_mutex_trylock (), pthread_mutex_unlock (): за допомогою mutex.
- pthread_cond_init(), pthread_cond_signal, pthread_cond_wait(): за допомогою умовних змінних.

Залишаються типи даних при роботі з потоками. Всього існує лише два типи:

- pthread_t: дескриптор потоку.
- pthread_attr_t: набір атрибутів потоку.