КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ
ТАРАСА ШЕВЧЕНКА
ФАКУЛЬТЕТ КОМП'ЮТЕРНИХ НАУК ТА КІБЕРНЕТИКИ

Звіт до лабораторної роботи №4
на тему
«System Call API. Hooks & Event Handlers.»

Студента 3 курсу ФКНК
групи ТТП-31
Корнієнка Олександра

Київ-2024

Зміст

Вступ

Скористатись можливостями ядра системи (System Call API) для наступних задач:
1) прочитати пам'ять іншого процесу (*наприклад*, в 1 процесі в змінну записуємо певне значення (наприклад, вводом з клавіатури), а 2й процес слідкує за цією ділянкою пам'ята 1го процесу та показує зміни, якщо вони відбуваються) або альтернативно – зробити сторінку **wired / non-paged**
2) перехопити (**hook**) клавіатуру або мишку із **неактивного** застосунку (тобто, *наприклад*, виводити у вікні те, що набирають на клавіатурі у іншому application, "перехопити клавіатурне введення") + продумати сценарій демонстрації.

Hints:
* див. /proc/[id]/mem - доступ до пам'яті
* /dev/input... - доступ до I/O-пристроїв

За реалізацію не в Linux (Unix) - штраф - 2 бали.

За розв'язок не на C / C++ / Rust - штраф - 2 бали.

Завдання 1

Вступ

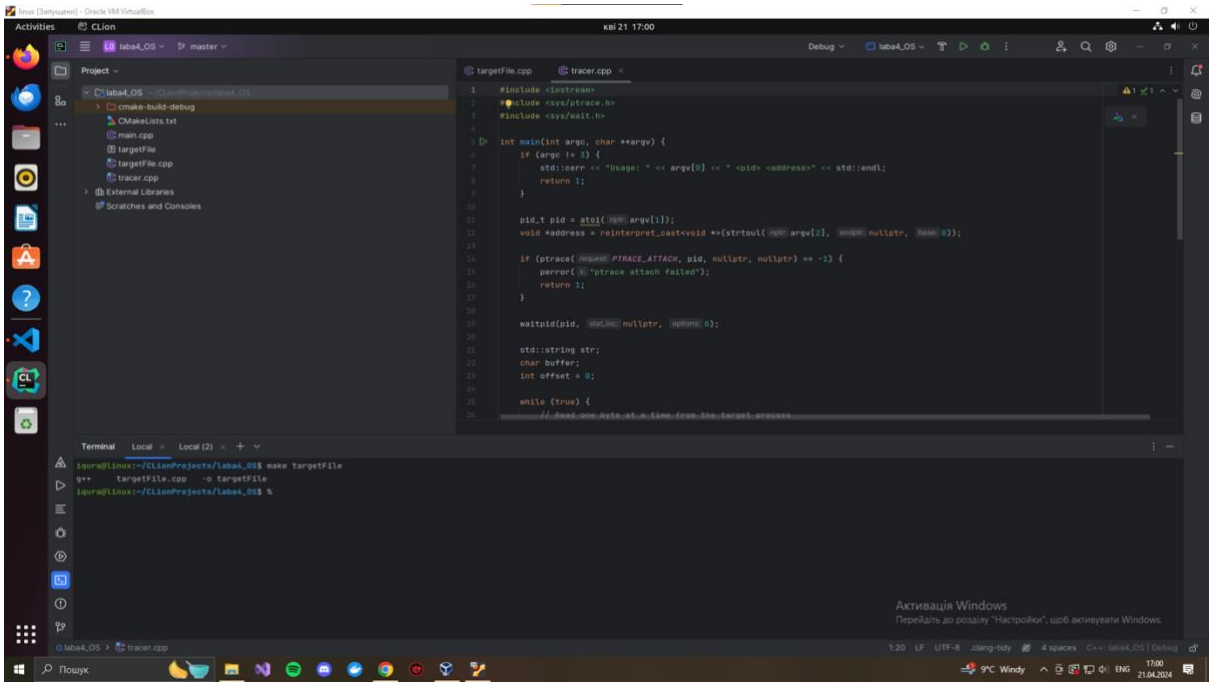
Для реалізації перехоплення процесу було використано операційну систему Linux(Ubuntu) та мову програмування c++ v17.

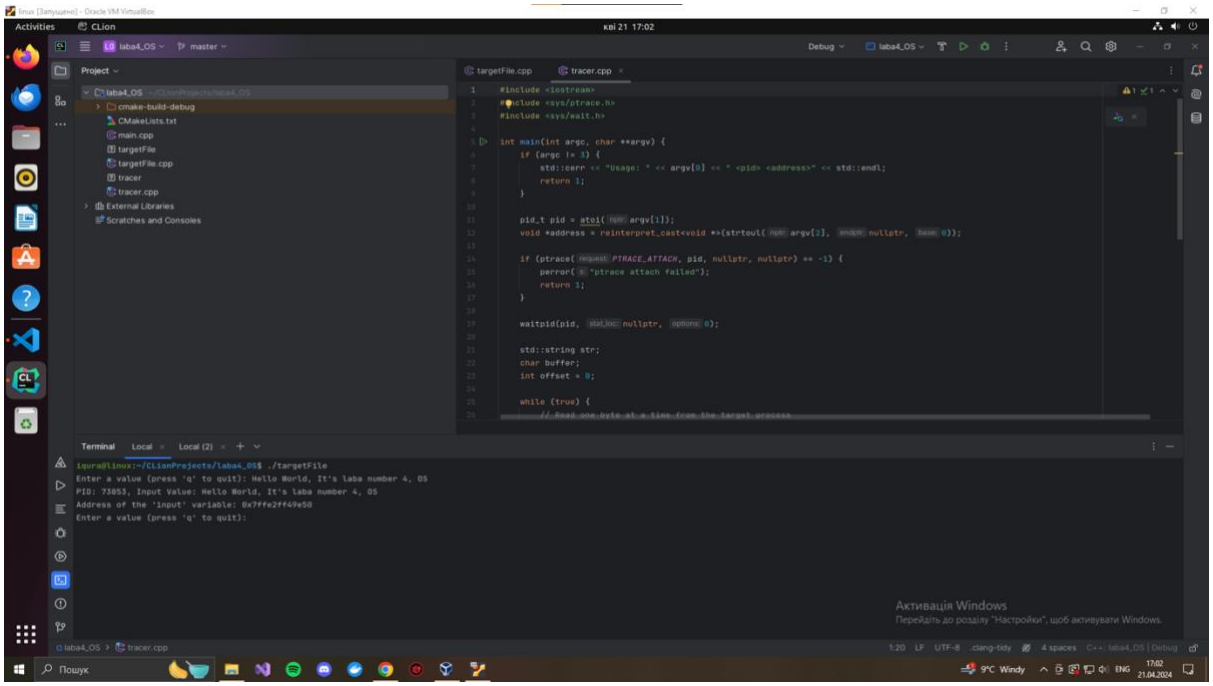
targetFile відображає перший процес, який надає змогу вводити з клавіатури та записувати в пам’яті дані. Результат виконання програми повертає айді процесу та адресу в пам’яті даних.

Tracer відображає другий процес, який слідує за першим.

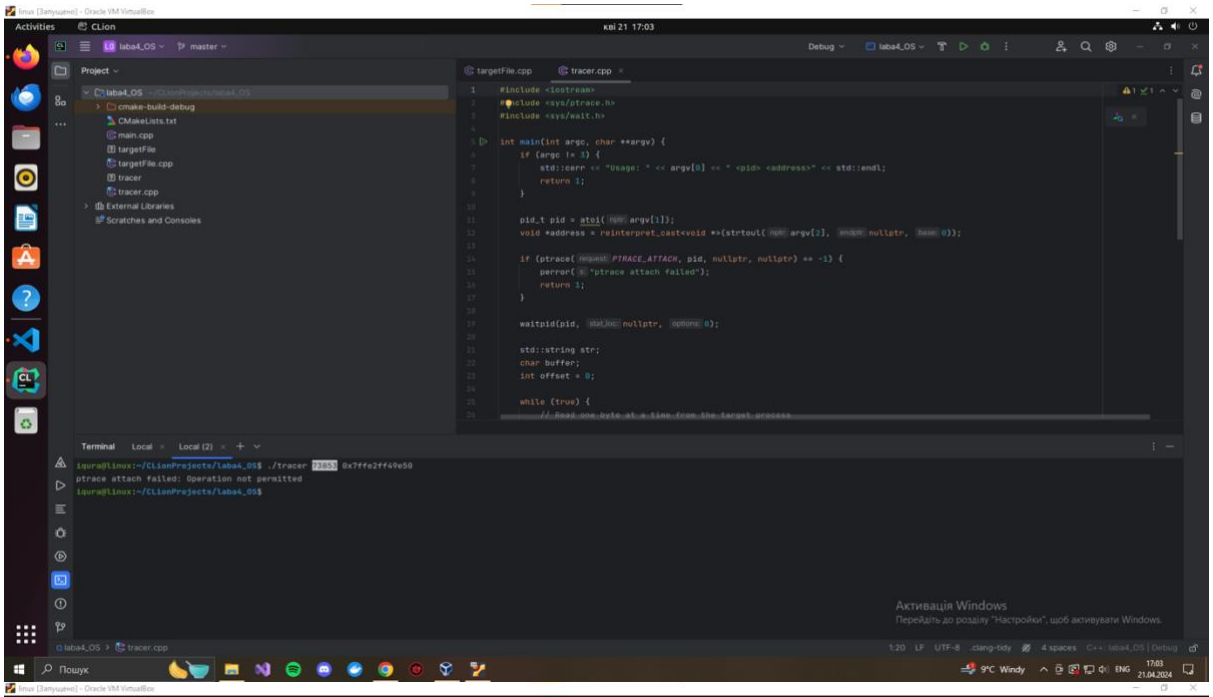
Реалізація

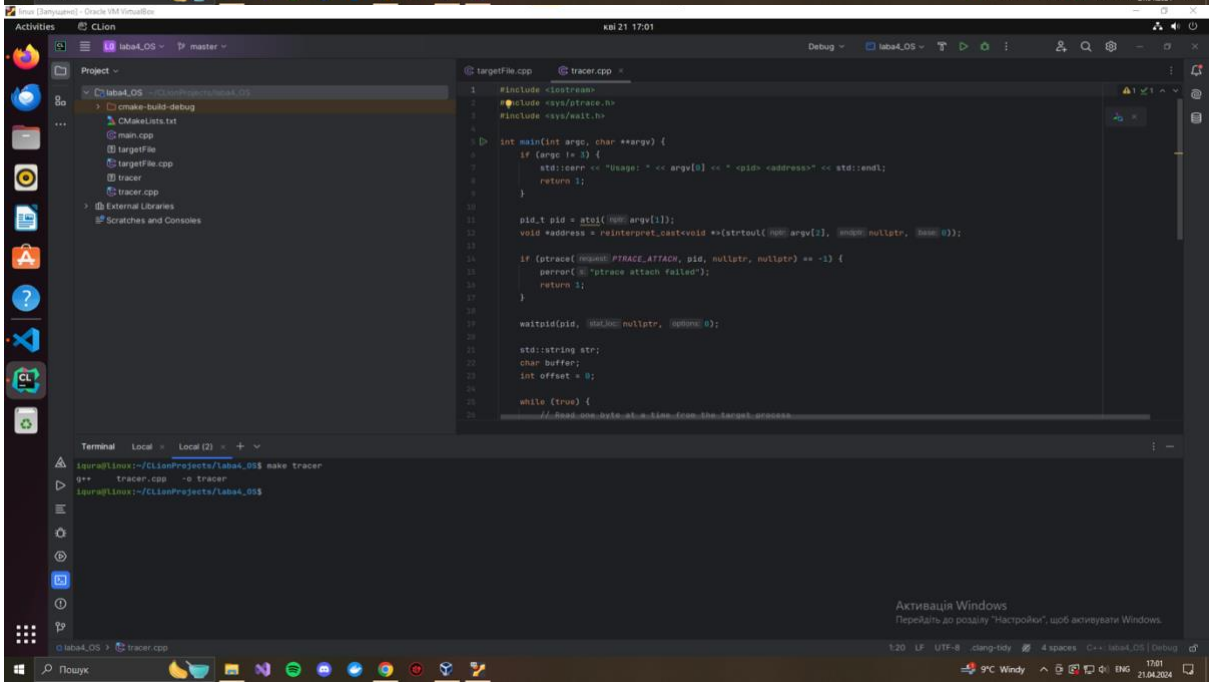
За допомогою команди make targetFile та ./targetFile створюємо та запускаємо процес.
вводим з клавіатури певний текст.

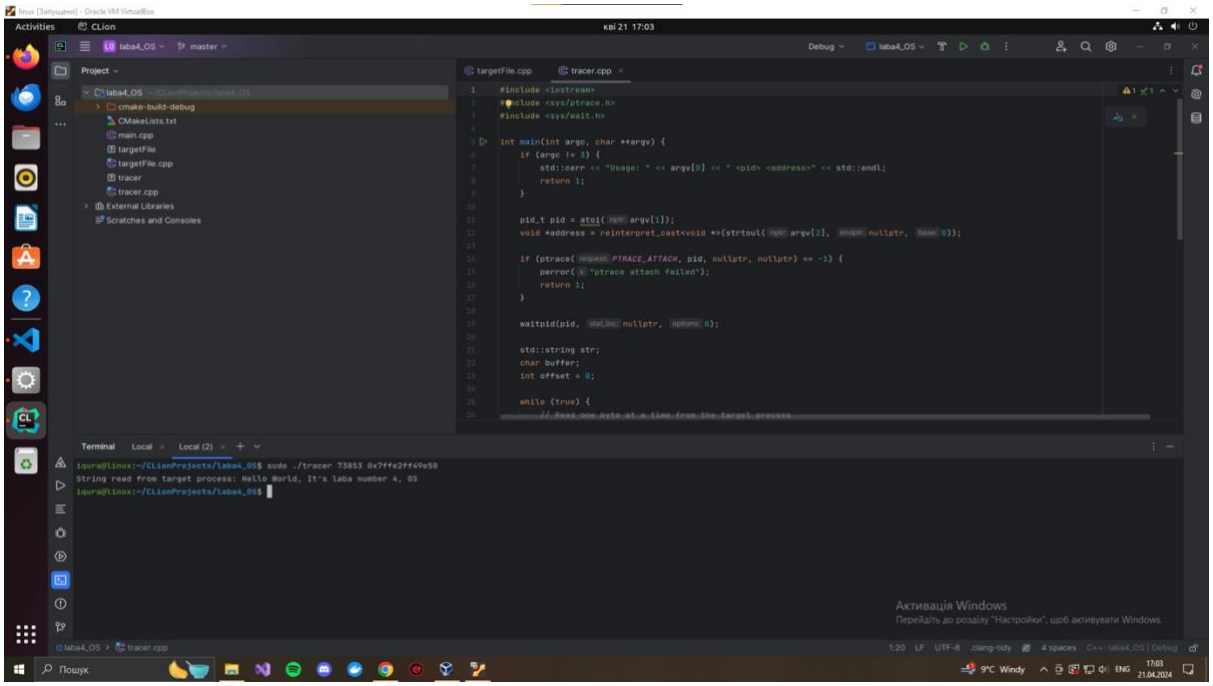




За допомогою команди make tracer та sudo ./tracer <PId> <address> створюємо та запускаємо процес. Нам потрібна команда sudo, так як всередині файлу tracer.cpp хочемо прочитати системні файли.


Activities
CLion
квіт 21 17:03
laba4_OS
master
Debug
Project
targetFile.cpp
tracer.cpp
laba4_OS
cmake-build-debug
CMakeLists.txt
main.cpp
targetFile
targetFile.cpp
tracer
tracer.cpp
External Libraries
Scratches and Consoles
#include <iostream>
#include <sys/ptrace.h>
#include <sys/wait.h>
int main(int argc, char **argv) {
if (argc != 3) {
std::cerr << "Usage: " << argv[0] << " <pid> <address>" << std::endl;
return 1;
}
pid_t pid = atoi(argv[1]);
void *address = reinterpret_cast<void *>(strtoul(argv[2], nullptr, 0));
if (ptrace(PTRACE_ATTACH, pid, nullptr, nullptr) == -1) {
perror("ptrace attach failed");
return 1;
}
waitpid(pid, nullptr, 0);
std::string str;
char buffer;
int offset = 0;
while (true) {
// Read one byte at a time from the target process
Terminal
Local
Local (2)
iqura@linux:~/CLionProjects/laba4_OS$ ./tracer 27651 0x7ffe2ff49e50
ptrace attach failed: Operation not permitted
iqura@linux:~/CLionProjects/laba4_OS$
Активація Windows
Перейдіть до розділу "Настройки", щоб активувати Windows.
laba4_OS
tracer.cpp
1:20
LF
UTF-8
clang-tidy
4 spaces
Пошук
9°C Windy
ENG
17:03
21.04.2024


Activities
CLion
квіт 21 17:01
laba4_OS
master
Debug
Project
targetFile.cpp
tracer.cpp
laba4_OS
cmake-build-debug
CMakeLists.txt
main.cpp
targetFile
targetFile.cpp
tracer
tracer.cpp
External Libraries
Scratches and Consoles
#include <iostream>
#include <sys/ptrace.h>
#include <sys/wait.h>
int main(int argc, char **argv) {
if (argc != 3) {
std::cerr << "Usage: " << argv[0] << " <pid> <address>" << std::endl;
return 1;
}
pid_t pid = atoi(argv[1]);
void *address = reinterpret_cast<void *>(strtoul(argv[2], nullptr, 0));
if (ptrace(PTRACE_ATTACH, pid, nullptr, nullptr) == -1) {
perror("ptrace attach failed");
return 1;
}
waitpid(pid, nullptr, 0);
std::string str;
char buffer;
int offset = 0;
while (true) {
// Read one byte at a time from the target process
Terminal
Local
Local (2)
iqura@linux:~/CLionProjects/laba4_OS$ make tracer
g++ tracer.cpp -o tracer
iqura@linux:~/CLionProjects/laba4_OS$
Активація Windows
Перейдіть до розділу "Настройки", щоб активувати Windows.
laba4_OS
tracer.cpp
1:20
LF
UTF-8
clang-tidy
4 spaces
Пошук
9°C Windy
ENG
17:01
21.04.2024



В результаті отримали дані, які були на вході у targetFile.

Код програми

 # targetFile.cpp

```
#include <iostream>
#include <unistd.h>
#include <cstring>

int main() {
    pid_t pid = getpid();

    while (true) {
        std::cout << "Enter a value (press 'q' to quit): ";
        char input[1000];

        std::cin.getline(input, 1000);

        if (strcmp(input, "q") == 0) {
            break;
        }

        std::cout << "PID: " << pid << ", Input Value: " << input << std::endl;
        std::cout << "Address of the 'input' variable: " << static_cast<void*>(&input) <<
std::endl;
    }

    return 0;
}
```

 # tracer.cpp

```
#include <iostream>
#include <sys/ptrace.h>
#include <sys/wait.h>
```

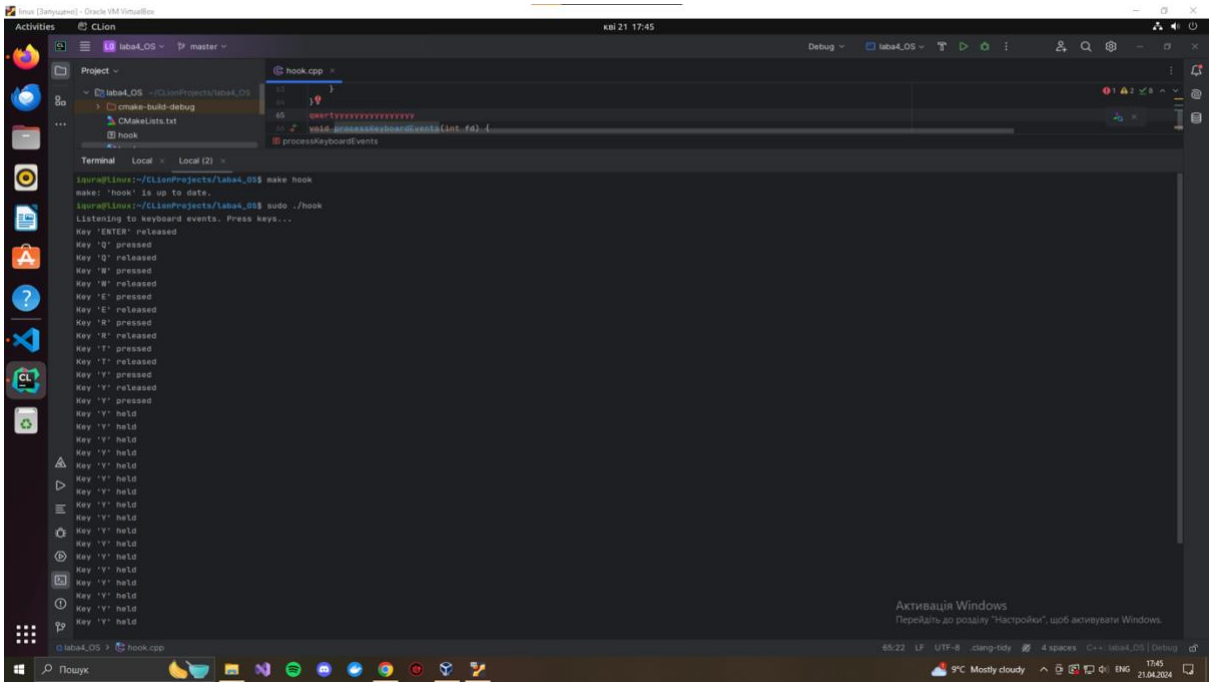
```
int main(int argc, char **argv) {
    if (argc != 3) {
        std::cerr << "Usage: " << argv[0] << " <pid> <address>" << std::endl;
        return 1;
    }

    pid_t pid = atoi(argv[1]);
    void *address = reinterpret_cast<void *>(strtoul(argv[2], nullptr, 0));

    if (ptrace(PTRACE_ATTACH, pid, nullptr, nullptr) == -1) {
        perror("ptrace attach failed");
        return 1;
    }

    waitpid(pid, nullptr, 0);

    std::string str;
    char buffer;
    int offset = 0;

    while (true) {
        // Read one byte at a time from the target process
        long data = ptrace(PTRACE_PEEKDATA, pid, reinterpret_cast<void
*>(reinterpret_cast<char *>(address) + offset), nullptr);
        if (data == -1) {
            perror("ptrace peekdata failed");
            ptrace(PTRACE_DETACH, pid, nullptr, nullptr);
            return 1;
        }
```

```
        buffer = static_cast<char>(data & 0xFF);
        str += buffer;

        if (buffer == '\0') {
            break;
        }

        ++offset;
    }

    std::cout << "String read from target process: " << str << std::endl;

    if (ptrace(PTRACE_DETACH, pid, nullptr, nullptr) == -1) {
        perror("ptrace detach failed");
        return 1;
    }

    return 0;
}
```

Завдання 2

Щоб реалізувати задану задачу, було створено файл hook.cpp, який дивиться у файл /dev/input/event2 в якому зберігається інформація про вхідні дані з клавіатури. Щоб отримати доступ до файлу івенту, нам потрібно права root, тобто використати sudo команду.

Спершу за допомогою make генеруємо файл: *make hook* та за допомогою команди *sudo ./hook* запускаємо процес



Код програми

 # hook.cpp

```
#include <fcntl.h>
#include <stdio.h>
#include <unistd.h>
#include <linux/input.h>

const char* KEYBOARD_DEVICE_PATH = "/dev/input/event2";
const char* UNKNOWN_KEY_TEXT = "Unknown key";
const char* KEY_RELEASED_TEXT = "released";
const char* KEY_PRESSED_TEXT = "pressed";
const char* KEY_HELD_TEXT = "held";
const char* DEVICE_OPEN_ERROR = "Unable to open device";
const char* DEVICE_CLOSE_ERROR = "Error closing device";

const char* getKeyName(int code);
const char* getKeyState(int value);
void processKeyboardEvents(int fd);

int main() {
    int keyboardFd = open(KEYBOARD_DEVICE_PATH, O_RDONLY);
    if (keyboardFd == -1) {
        perror(DEVICE_OPEN_ERROR);
        return 1;
    }

    printf("Listening to keyboard events. Press keys...\n");

    processKeyboardEvents(keyboardFd);

    if (close(keyboardFd) == -1) {
```

```
        perror(DEVICE_CLOSE_ERROR);
        return 1;
    }

    return 0;
}

const char* getKeyName(int code) {
    static const char* keymap[KEY_CNT] = {
            "", "ESC", "1", "2", "3", "4", "5", "6", "7", "8", "9", "0", "-", "=",
            "BACKSPACE", "TAB", "Q", "W", "E", "R", "T", "Y", "U", "I", "O", "P", "[", "]",
"ENTER", "LEFTCTRL",
            "A", "S", "D", "F", "G", "H", "J", "K", "L", ";", "'", "`", "LEFTSHIFT", "\\", "Z", "X",
            "C", "V", "B", "N", "M", ",", ".", "/", "RIGHTSHIFT", "KPASTERISK", "LEFTALT",
"SPACE", "CAPSLOCK", "F1", "F2", "F3",
            "F4", "F5", "F6", "F7", "F8", "F9", "F10", "NUMLOCK", "SCROLLLOCK", "KP7",
"KP8", "KP9", "KPMINUS", "KP4", "KP5",
            "KP6", "KPPLUS", "KP1", "KP2", "KP3", "KP0", "KPDOT", "", "", "102ND", "F11",
"F12"
    };

    if (code >= 0 && code < KEY_CNT) {
        return keymap[code];
    } else {
        return UNKNOWN_KEY_TEXT;
    }
}

const char* getKeyState(int value) {
    switch (value) {
        case 0: return KEY_RELEASED_TEXT;
```

```
        case 1: return KEY_PRESSED_TEXT;
        case 2: return KEY_HELD_TEXT;
        default: return UNKNOWN_KEY_TEXT;
    }
}

void processKeyboardEvents(int fd) {
    struct input_event event;

    while (read(fd, &event, sizeof(struct input_event)) > 0) {
        if (event.type == EV_KEY) {
            printf("Key '%s' %s\n", getKeyName(event.code), getKeyState(event.value));
        }
    }
}
```