Звіт

Лабораторна робота №4

студента групи ТТП-32

Ходакова Максима

**Умова**

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 2**

**Реалізація**

#include **<stdlib.h>**#include **<unistd.h>**#include **<fcntl.h>**#include **<errno.h>**#include **<linux/input.h>**#include **<string.h>**#include **<stdio.h>  
  
static const char** \***const** evval[3] = {  
 **"RELEASED"**,  
 **"PRESSED "**,  
 **"REPEATED"**};  
  
**int** main() {  
 **const char** \*dev = **"/dev/input/by-path/platform-i8042-serio-0-event-kbd"**;  
 **struct** input\_event ev;  
 ssize\_t n;  
 **int** fd;  
  
 fd = open(dev, **O\_RDONLY**);  
 **if** (fd == -1) {  
 fprintf(**stderr**, **"Cannot open %s: %s.\n"**, dev, strerror(**errno**));  
 **return EXIT\_FAILURE**;  
 }  
 **while** (1) {  
 n = read(fd, &ev, **sizeof** ev);  
 **if** (n == (ssize\_t) -1) {  
 **if** (**errno** == **EINTR**)  
 **continue**;  
 **else  
 break**;  
 } **else if** (n != **sizeof** ev) {  
 **errno** = **EIO**;  
 **break**;  
 }  
 **if** (ev.type == EV\_KEY && ev.value >= 0 && ev.value <= 2)  
 printf(**"%s 0x%04x (%d)\n"**, evval[ev.value], (**int**) ev.code, (**int**) ev.code);  
  
 }  
 fflush(**stdout**);  
 fprintf(**stderr**, **"%s.\n"**, strerror(**errno**));  
 **return EXIT\_FAILURE**;  
}

**Приклад використання**

**A screenshot of a computer

Description automatically generated**

**Опис програми**

Наявний програмний код написаний на мові C для Linux-систем і призначений для моніторингу клавіатурних подій, таких як натискання, утримання та відпускання клавіш. Програма читає дані безпосередньо з пристриєвого файлу клавіатури в системі Linux, що дозволяє отримувати інформацію про всі клавіші, натискані користувачем. Ось детальний опис роботи коду:

1. Підключення необхідних заголовочних файлів: stdlib.h, unistd.h, fcntl.h, errno.h - стандартні заголовочні файли для роботи з системними викликами, обробки помилок і роботи з файлами. linux/input.h - заголовочний файл, що містить визначення, необхідні для роботи з подіями вводу в Linux. string.h і stdio.h - для роботи з рядками та вводу-виводу.
2. Оголошення масиву evval: Масив рядків, що представляє можливі стани клавіші: "RELEASED", "PRESSED ", "REPEATED".
3. Функція main: Основна функція програми, яка виконує весь процес моніторингу.
4. Відкриття пристроєвого файлу клавіатури: Визначається шлях до пристроєвого файлу клавіатури. У цьому випадку використовується файл /dev/input/by-path/platform-i8042-serio-0-event-kbd, який є символічним посиланням на подію клавіатури для певного пристрою вводу. Файл відкривається тільки для читання за допомогою системного виклику open. Якщо відкриття файлу не вдається, програма виводить помилку і завершується з кодом помилки EXIT\_FAILURE.
5. Читання подій з пристроєвого файлу: Виконується нескінченний цикл, в якому програма читає події з файлу за допомогою системного виклику read. Кожне читання спробує прочитати одну структуру input\_event з файлу. Якщо читання повертає помилку EINTR (переривання перед завершенням операції), програма продовжує читання. Інші типи помилок призводять до завершення циклу.
6. Обробка подій клавіатури: Якщо тип події є EV\_KEY (подія клавіатури), і значення події (ev.value) відповідає одному зі станів клавіші (відпущена, натиснута, повторно натиснута), програма виводить стан, код події у шістнадцятковому форматі та у десятковому форматі.
7. Завершення програми: Після виходу з циклу читання, програма виводить повідомлення про помилку (причину виходу з циклу) і завершується з кодом EXIT\_FAILURE.

Цей код дозволяє розробникам та ентузіастам виводити інформацію про клавіатурні події в реальному часі, що може бути корисно для розробки ігор, інтерфейсів користувача або для діагностики проблем з клавіатурою.

**Завдання 1**

**Реалізація**

**pid\_teller.cpp**

#include **<unistd.h>**#include **<iostream>  
  
char** data[] = **"Very secret important data\n"**;  
  
**int** main() {  
 std::cout << **"pid: "** << getpid() << **"\nData address: "** << (**void** \*)data << **"\nData size: "** << **sizeof**(data) << **"\n"**;  
 std::cin.get();  
  
 **return** 0;  
}

**Лістинг програми**

**A screen shot of a computer

Description automatically generated**

**Опис роботи**

Ця програма призначена для демонстрації основних принципів взаємодії між процесами у операційній системі за допомогою читання пам'яті одного процесу іншим. Вона виводить інформацію, необхідну для доступу до певної області пам'яті процесу ззовні, і чекає на вхід від користувача, перш ніж завершитись. Давайте розглянемо її роботу та використання детальніше: Як працює програма Виведення інформації про процес: getpid() використовується для отримання ідентифікатора поточного процесу (PID). Цей PID потрібен для того, щоб інший процес міг ідентифікувати, до пам'яті якого процесу необхідно звернутися. Адреса змінної data в пам'яті виводиться за допомогою приведення типу до (void \*)data. Це дає можливість зовнішньому процесу знати, де саме розташовані дані. Розмір масиву data, який включає текст і нульовий символ в кінці, виводиться за допомогою sizeof(data). Це інформує зовнішній процес про кількість байтів, які потрібно прочитати. Очікування вводу від користувача: Виклик std::cin.get(); зупиняє виконання програми і чекає на натискання клавіші. Це дає змогу користувачу або іншому процесу використовувати виведену інформацію для доступу до пам'яті процесу перш, ніж програма завершиться і пам'ять буде звільнена або змінена. Використання виведеної інформації: Записують або копіюють PID, адресу, і розмір для використання в іншій програмі, яка буде читати пам'ять. Ці дані використовуються для налаштування другої програми, що здійснює доступ до файлу /proc/[pid]/mem із зазначенням початкової адреси і кількості байтів, які необхідно прочитати, щоб отримати доступ до змісту змінної data.

**process\_reader.cpp**

#include **<iostream>**#include **<iomanip>**#include **<fcntl.h>**#include **<unistd.h>***// ./a.out pid address size***int** main(**int** argc, **char** \*\*argv) {  
 **if** (argc != 4) {  
 **return** -1;  
 }  
 std::string pid{argv[1]};  
 **auto** start = std::stoull(argv[2], **nullptr**, 16);  
 **auto** size = std::stoull(argv[3]);  
 **auto** filename = **"/proc/"** + pid + **"/mem"**;  
 std::cout << **"Filename: "** << filename << **"\nAddr is: "** << (**void** \*) start << **"\nSize: "** << size << **"\n"**;  
  
 **int** fd = open(filename.c\_str(), **O\_RDONLY**);  
 **if**(fd == -1) {  
 **return** -2;  
 }  
  
 off\_t res = lseek(fd, (off\_t)start, **SEEK\_SET**);  
 **if**(res == (off\_t)-1) {  
 **return** -3;  
 }  
 std::string output(size, 0);  
 **auto** read\_res = read(fd, output.data(), output.size());  
 std::cout << **"Data is: "** << std::quoted(output) <<**"\n"**;  
 **return** 0;  
}

**Лістинг програми**

**A screen shot of a computer

Description automatically generated**

**Опис роботи**

Ця програма призначена для читання пам'яті іншого процесу в Linux, використовуючи інтерфейс файлової системи /proc. Це робиться шляхом доступу до файлу /proc/[pid]/mem, де [pid] – це ідентифікатор процесу (PID), пам'ять якого потрібно прочитати. Програма розроблена для запуску з командного рядка з трьома аргументами: PID цільового процесу, початковою адресою в пам'яті цього процесу для читання, і розміром блоку даних, який необхідно прочитати. Параметри командного рядка pid - Ідентифікатор процесу (PID) цільового процесу. address - Початкова адреса в пам'яті цільового процесу, з якої почнеться читання. Вказується у шістнадцятковому форматі. size - Кількість байтів, які потрібно прочитати з пам'яті.

Робота програми Перевірка наявності аргументів командного рядка: Програма спочатку перевіряє, чи передано їй рівно 3 аргументи командного рядка (не враховуючи ім'я програми). Якщо ні, програма завершується з кодом помилки -1. Читання та обробка аргументів: PID цільового процесу зчитується з першого аргумента. Початкова адреса для читання визначається з другого аргумента, який конвертується з шістнадцяткового рядка у числове значення. Розмір даних для читання зчитується з третього аргумента і також конвертується у числове значення. Відкриття файлу пам'яті цільового процесу: Створюється шлях до файлу пам'яті цільового процесу /proc/[pid]/mem і відкривається цей файл у режимі тільки для читання. Якщо файл не вдається відкрити, програма завершується з кодом помилки -2. Позиціонування у файлі пам'яті: Використовуючи системний виклик lseek, програма переміщується до початкової адреси в файлі пам'яті, зазначеної у другому аргументі. Якщо ця операція не вдається, програма завершується з кодом помилки -3. Читання даних: Дані читаються з файлу пам'яті в рядок output, розмір якого було задано третім аргументом. Виведення даних: Прочитані дані виводяться у консоль з використанням функції std::quoted для коректного відображення спеціальних символів і потенційного тексту.

**Висновок**

Я використав C++ програми з Hooks та System Call API на Linux операційній системі, для читання пам’яті з інших процесів та реалізував перехоплення процесів. GitHub: [лінк](https://github.com/maksymkhodakov/OSLab4).