

Міністерство освіти і науки, молоді та спорту України
Національний університет “Львівська політехніка”

Кафедра ЕОМ



Звіт

з домашнє завдання №28.2 з дисципліни:

“Алгоритми та моделі обчислень”

Варіант: № 24.

Виконав:

ст. групи КІ-203

Ширий Богдан Ігорович

Перевірів:

ст. викладач кафедри ЕОМ

Козак Назар Богданович

ЗАВДАННЯ:

УМОВА:

У домашньому завданні №28.3 потрібно буде виконати домашнє завдання №27.1 повторно як альтернативну низькорівневу реалізацію домашнього завдання №28.1 мовою C. В якості домашнього завдання №28.2 пропонується також виконати домашнє завдання №27.1 повторно як альтернативну низькорівневу реалізацію домашнього завдання №28.1 мовою C, але за допомогою бібліотеки `libuv`. Рушій платформи NodeJS побудований на основі цієї бібліотеки, яка була створена для заміни `libeio` (імплементує `Tread Pool`) та `libev` (імплементує `Event Loop`). (Рушій JavaScript для NodeJS це V8, але рушієм подійно-орієнтованої парадигми у NodeJS є саме `libuv`).

ВИБІР ВАРІАНТУ:

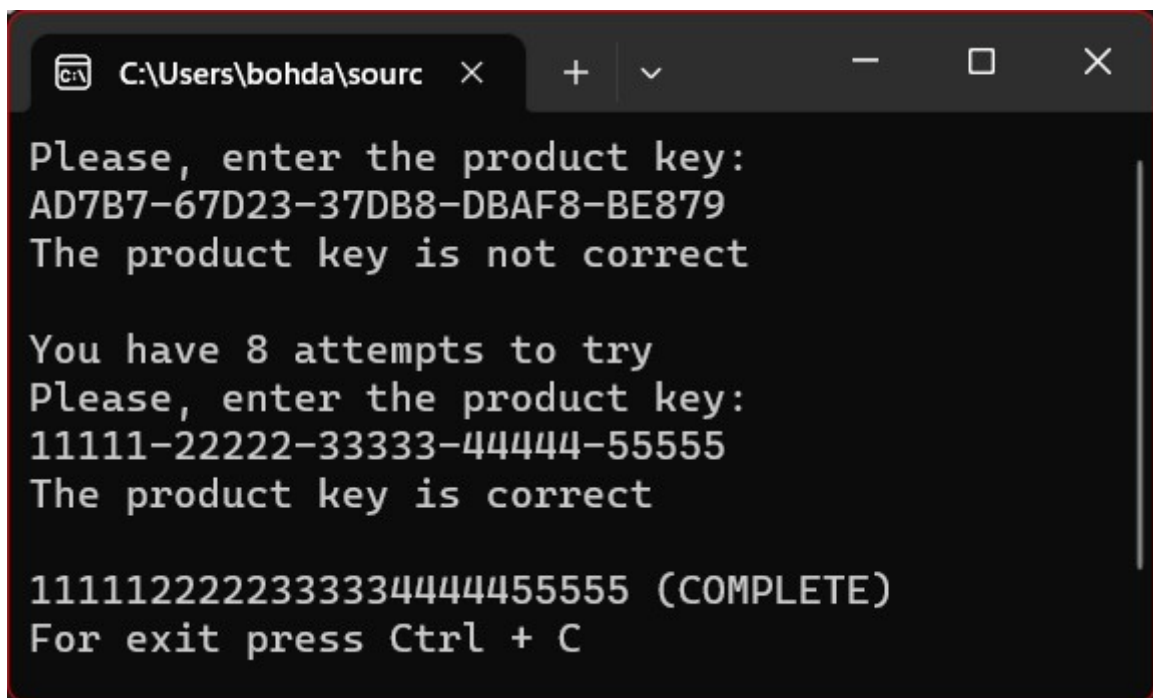
$$(N_{\text{ж}} + N_{\text{г}} + 1) \% 30 + 1 = (24 + 3 + 1) \% 10 + 1 = 28 \% 10 + 1 = 9,$$

де: $N_{\text{ж}}$ – порядковий номер студента в групі, а $N_{\text{г}}$ – номер групи.

Отож, мій шуканий варіант – це 9 можливих спроб введення ключа валідації.

ВИКОНАННЯ:

Склав C програму та зобразив її роботу на рисунку 1.



```
C:\Users\bohda\sourc x + v - □ x

Please, enter the product key:
AD7B7-67D23-37DB8-DBAF8-BE879
The product key is not correct

You have 8 attempts to try
Please, enter the product key:
11111-22222-33333-44444-55555
The product key is correct

1111122222333334444455555 (COMPLETE)
For exit press Ctrl + C
```

Рис. 1. Виконання програми написаної на C.

Відповідно, у лістингу 1 навів код програми написаної на C:

Лістинг 1. Код програми написаної на C.

```
#define _CRT_SECURE_NO_WARNINGS
#define WIN32_LEAN_AND_MEAN
#if _WIN32
#include <Windows.h>
#pragma comment(lib, "Ws2_32.lib")
#pragma comment(lib, "psapi.lib")
#pragma comment(lib, "Iphlpapi.lib")
#pragma comment(lib, "userenv.lib")
#endif

#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#if __linux__
#include <unistd.h>
#endif
#if _WIN32
#include "windows_addon/uv.h"
#else
#include "linux_addon/uv.h"
#endif

#define ATTEMPTS_COUNT 9
int attemptsDownCount = ATTEMPTS_COUNT;

#define GROUPS_DIGITS_COUNT 5
#define GROUP_DIGITS_SIZE 5

const unsigned char PRODUCT_KEY_PART1[] = {
    0xF, 0xF, 0xF, 0xF, 0xF,
    0xD, 0xD, 0xD, 0xD, 0xD,
    0x8, 0x8, 0x8, 0x8, 0x8,
    0xB, 0xB, 0xB, 0xB, 0xB,
    0xF, 0xF, 0xF, 0xF, 0xF
};

const unsigned char PRODUCT_KEY_PART2[] = {
    0xE, 0xE, 0xE, 0xE, 0xE,
    0xF, 0xF, 0xF, 0xF, 0xF,
    0xB, 0xB, 0xB, 0xB, 0xB,
    0xF, 0xF, 0xF, 0xF, 0xF,
    0xA, 0xA, 0xA, 0xA, 0xA
};

#define DIGITS_COUNT (GROUPS_DIGITS_COUNT * GROUP_DIGITS_SIZE)

#define TYPER_FULL_RAW_MODE

#define IS_KEY_UP(CH0, CH1, CH2) (CH0 == 0x1b && CH1 == '[' && CH2 == 'A')
#define IS_KEY_DOWN(CH0, CH1, CH2) (CH0 == 0x1b && CH1 == '[' && CH2 == 'B')
#define IS_KEY_LEFT(CH0, CH1, CH2) (CH0 == 0x1b && CH1 == '[' && CH2 == 'D')
#define IS_KEY_RIGHT(CH0, CH1, CH2) (CH0 == 0x1b && CH1 == '[' && CH2 == 'C')
#define IS_ESCAPE_KEY(CH0, CH1) (CH0 == 0x1b && CH1 == 0x1b)
#define IS_KEY_DELETE(CH0, CH1, CH2, CH3) (CH0 == 0x1b && CH1 == '[' && CH2 == '3')
// && CH3 == '^')
#if _WIN32
#define IS_KEY_BACKSPACE(CH0) (CH0 == 8)
#else
#define IS_KEY_BACKSPACE(CH0) (CH0 == 127)
#endif
#ifdef TYPER_FULL_RAW_MODE
#define IS_KEY_ENTER(CH0) (CH0 == 13)
#else
#define IS_KEY_ENTER(CH0) (CH0 == 10)
#endif
```

```

#define IS_KEY_CTRL(C) (C == 3)

int outOfEdgeIndex = 0;
int currIndex = 0;
unsigned char data[DIGITS_COUNT] = { 0 };

char checkProductKey(unsigned char * productKey){
    unsigned int index;
    for (index = 0; index < DIGITS_COUNT; ++index){
        if (productKey[index] ^ PRODUCT_KEY_PART1[index] ^
PRODUCT_KEY_PART2[index]){
            return 0;
        }
    }

    return ~0;
}

void toDigitPosition(unsigned int currIndex){
    int positionAddon;
#ifdef _WIN32
#else
    char temp[16];
#endif
#ifdef _WIN32
    CONSOLE_SCREEN_BUFFER_INFO cbsi;
    COORD pos;
    HANDLE hConsoleOutput = GetStdHandle(STD_OUTPUT_HANDLE);
    GetConsoleScreenBufferInfo(hConsoleOutput, &cbsi);
    pos = cbsi.dwCursorPosition;
#endif
    positionAddon = currIndex / GROUP_DIGITS_SIZE;
    positionAddon && positionAddon >= GROUPS_DIGITS_COUNT ? --positionAddon : 0;
#ifdef _WIN32
    currIndex += positionAddon;
    pos.X = currIndex;
    SetConsoleCursorPosition(hConsoleOutput, pos);
#else
    write(STDOUT_FILENO, "\033[64D", 5);
    if(currIndex += positionAddon){
        sprintf(temp, "\033[%dC", currIndex);
        write(STDOUT_FILENO, temp, strlen(temp));
    }
#endif
}

void printProductKey(unsigned char * productKey, unsigned int outOfEdgeIndex){
    unsigned int index;
    unsigned char value;
    for (index = 0; index < DIGITS_COUNT && index < outOfEdgeIndex; ++index){
        value = productKey[index];
        value > 9 ? (value += 'A' - 10) : (value += '0');
#ifdef _WIN32
        printf("%c", value);
#else
        write(STDOUT_FILENO, &value, 1);
#endif
    }
}

void printFormattedProductKey(unsigned char * productKey, unsigned int
outOfEdgeIndex){
    unsigned int index;
    unsigned char value;
    for (index = 0; index < DIGITS_COUNT && index < outOfEdgeIndex; ++index){
        value = productKey[index];
        value > 9 ? (value += 'A' - 10) : (value += '0');
#ifdef _WIN32

```

```

        printf("%c", value);
    #else
        write(STDOUT_FILENO, &value, 1);
    #endif
        if (!(index + 1) % GROUP_DIGITS_SIZE) && (index + 1) < DIGITS_COUNT){
    #if _WIN32
        printf("-");
    #else
        write(STDOUT_FILENO, "-", 1);
    #endif
        }
    }
}

void inputHandler(int ch0, int ch1, int ch2, int ch3){
    char chstr[2] = { 0 };
    char * hexDigitScanfPattern = (char*)"[%0-9abcdefABCDEF]"; // "[%0-9A-Fa-f]/g

    if (!attemptsDownCount){
        return;
    }
    if (IS_KEY_ENTER(ch0)) {
        if (checkProductKey(data)) {
    #if _WIN32
            printf("\nThe product key is correct\n\n");
        #else
            write(STDOUT_FILENO, "\nThe product key is correct\n\n", 29);
        #endif
            printProductKey(data, outOfEdgeIndex);
    #if _WIN32
            printf(" (COMPLETE)", 11);
            printf("\nFor exit press Ctrl + C\n");
        #else
            write(STDOUT_FILENO, " (COMPLETE)", 11);
            write(STDOUT_FILENO, "\nFor exit press Ctrl + C\n", 25);
        #endif
            attemptsDownCount = 0;
        }
        else{
    #if _WIN32
            printf("\nThe product key is not correct\n");
            printf("\nYou have %d attempts to try\n", --attemptsDownCount);
        #else
            write(STDOUT_FILENO, "\nThe product key is not correct\n", 32);
            printf("\nYou have %d attempts to try\n", --attemptsDownCount);
        #endif
            if (attemptsDownCount){
    #if _WIN32
                printf("Please, enter the product key:\n");
            #else
                write(STDOUT_FILENO, "Please, enter the product key:\n", 31);
            #endif
                printFormattedProductKey(data, outOfEdgeIndex);
                toDigitPosition(currIndex);
            }
            else{
    #if _WIN32
                printf("The product key is not entered\n");
                printf("For exit press Ctrl + C\n");
            #else
                write(STDOUT_FILENO, "The product key is not entered\n", 31);
                write(STDOUT_FILENO, "For exit press Ctrl + C\n", 24);
            #endif
            }
        }
    }
    else if (IS_KEY_BACKSPACE(ch0)) {
        if (currIndex){

```

```

        --currIndex;
        toDigitPosition(currIndex);
        data[currIndex] = 0;
#ifdef _WIN32
        printf("0");
#else
        write(STDOUT_FILENO, "0", 1);
#endif
        toDigitPosition(currIndex);
    }
}
else if (IS_KEY_DELETE(ch0, ch1, ch2, ch3)) {
    toDigitPosition(currIndex);
    data[currIndex] = 0;
#ifdef _WIN32
    printf("0");
#else
    write(STDOUT_FILENO, "0", 1);
#endif
    toDigitPosition(currIndex);
}
else if (IS_KEY_LEFT(ch0, ch1, ch2)) {
    if (currIndex){
        toDigitPosition(--currIndex); // got to 1.5
    }
}
else if (IS_KEY_RIGHT(ch0, ch1, ch2)) {
    if (currIndex < outOfEdgeIndex){
        toDigitPosition(++currIndex);
    }
}
else if (IS_ESCAPE_KEY(ch0, ch1)){
    ch0 == ' ' || ch0 == '\t' ? ch0 = '0' : 0;

    if (currIndex < DIGITS_COUNT && ch0 && sscanf((char*)&ch0,
hexDigitScanfPattern, chstr_) > 0) {
        data[currIndex] = (unsigned char)strtol(chstr_, NULL, 16);
#ifdef _WIN32
        printf("%X", data[currIndex]);
#else
        sprintf(chstr_, "%X", data[currIndex] );
        write(STDOUT_FILENO, chstr_, 1);
#endif

        if (outOfEdgeIndex <= currIndex){
            outOfEdgeIndex = currIndex + 1;
        }
        if (currIndex + 1 < DIGITS_COUNT) {
            ++currIndex;
            if (currIndex != DIGITS_COUNT && !(currIndex % 5)) {
#ifdef _WIN32
                printf("-");
#else
                write(STDOUT_FILENO, "-", 1);
#endif
            }
        }
        if (currIndex + 1 == DIGITS_COUNT){
            toDigitPosition(currIndex);
        }
    }
}

uv_loop_t mainLoop;

void terminateHandler(int ch0, int ch1, int ch2, int ch3){
    if (IS_KEY_CTRL_C(ch0)) {
        uv_stop(&mainLoop);
    }
}

```

```

    }
}

static void alloc_buffer(uv_handle_t *handle, size_t suggested_size, uv_buf_t *buf)
{
    static char buffer[1 << 16];
    *buf = uv_buf_init(buffer, 1 << 16);
}

static void read_stdin(uv_stream_t *stream, ssize_t nread, const uv_buf_t* buf){
    int ch0 = nread > 0 ? buf->base[0] : 0;
    int ch1 = nread > 1 ? buf->base[1] : 0;
    int ch2 = nread > 2 ? buf->base[2] : 0;
    int ch3 = nread > 3 ? buf->base[3] : 0;

    inputHandler(ch0, ch1, ch2, ch3);
    terminateHandler(ch0, ch1, ch2, ch3);
}

int main(){
    uv_tty_t input;
    uv_loop_init(&mainLoop);

    uv_tty_init(&mainLoop, &input, 0, 1);
    uv_tty_set_mode(&input, UV_TTY_MODE_RAW);

    uv_read_start((uv_stream_t *)&input, alloc_buffer, read_stdin);

#ifdef _WIN32
    printf("Please, enter the product key:\n");
#else
    write(STDOUT_FILENO, "Please, enter the product key:\n", 31);
#endif

    uv_run(&mainLoop, UV_RUN_DEFAULT);

    uv_tty_reset_mode();

    return 0;
}

```