Лабораторна робота №1

Програмування апаратного забезпечення

Шевченко Е.Р. КС-21

Мета: покращити навички роботи з апаратним забезпеченням за рахунок програмування низькорівневих елементів за відсутності стандартних засобів операційної системи. Покращення навичок структурування програм за рахунок створення гнучких програмних компонентів.

Оголошення

Розроблений код має представляти собою основні компоненти ядра простої операційної системи (ОС), написаної на мові програмування C. Операційна система ChristOS визначається текстовим інтерфейсом, фактичною консоллю, що надає базові функції для взаємодії користувача з системою. Написаний код має слугувати демонстрацією основних аспектів функціонування ядра ОС та містити ключові елементи, такі як робота з файловою системою, обробка введення, кольорова графіка консолі, та інші елементи.

Перший підрозділ – це заголовкові файли, що визначають необхідні бібліотеки для взаємодії з операційною системою, обробки введення/виведення, роботи з часом тощо.

```
1   #include "kernel/kernel.h"
2   #include "drivers/keyboard/keyboard.h"
3   #include "drivers/timer/timer.h"
4   #include "drivers/serial_port/serial_port.h"
5
```

Після чого визначаються макроси для різних кольорів консолі, представлених у восьмеричному форматі. Ці конкретні значення представляють різні кольори, такі як чорний, зелений, жовтий, червоний та білий.

cnsl_state_clrs[25 * 80] – це глобальний масив cnsl_state_clrs створений для зберігання стану кольорів консолі. Даний одновимірний масив розміром 25 * 80, що відповідає розмірності текстового режиму консолі (25 рядків по 80

символів в кожному рядку). Кожен елемент масиву cnsl_state_clrs визначає колір символу в конкретному положенні на екрані. Тобто, якщо потрібно змінити колір символу в певному рядку та стовпці на екрані, потрібно звертатися до відповідного елемента цього масиву та встановлювати новий колір за допомогою вищезазначених макросів.

```
//colors-----------------------------------------------------
#define BLACK_COLOR 0x0
#define GREEN_COLOR 0xa
#define YELLOW_COLOR 0xe
#define RED_COLOR 0xc
#define WHITE_COLOR 0xf
unsigned char cnsl_state_clrs[25 * 80];
```

Наступним підрозділом є змінні та параметри, які використовуються для анімації "screensaver" (екранної заставки) в консольному додатку. Змінна anim_sleep_continue вказує, чи триває анімація "screensaver". Якщо значення true, то анімація продовжується, в іншому випадку - анімація призупинена. cnsl_state_chrs[25 * 80] – це одновимірний масив, який використовується для зберігання стану символів (тексту) на екрані консолі. Кожен елемент масиву відповідає символу в певному положенні на екрані. cnsl_state_pntr = 0 – ця змінна визначає вказівник, який вказує на поточне положення у масиві cnsl_state_chrs для збереження та відновлення стану символів.

Slide_msg відповідає за зсув у рядку символів для анімації, де row і col поточні координати для виведення символів анімації. curr_char – це індекс поточного символу в масиві cnsl_state_chrs. anim_offsite_X і anim_offsite_Y відступ для анімації "screensaver". sleep_img рядок, що містить графічний зразок для анімації. anim_size_X і anim_size_Y: розміри анімації. anim_coord_X і anim_coord_Y: напрямок анімації (1 або -1 для кожної координати).

```
//animation addons-----------------------------------
bool anim_sleep_continue = false;
unsigned char cnsl_state_chrs[25 * 80];
unsigned short cnsl_state_pntr = 0;

unsigned short slide_msg = 0;
unsigned char row = 0;
unsigned char col = 0;
unsigned short curr_char = 0;

unsigned short anim_offsite_X = 0;
unsigned short anim_offsite_Y = 0;
char *sleep_img =
    "   ##   \n"
    "   ##   \n"
    "########\n"
    "   ##   \n"
    "   ##   \n"
    "   ##   \0";
unsigned short anim_size_X = 8;
unsigned short anim_size_Y = 6;
char anim_coord_X = 1;
char anim_coord_Y = 1;
```

Далі визначаються допоміжні елементи для роботи з файлами. edit_mode = false вказує, чи програма знаходиться в режимі редагування. Якщо true, то програма знаходиться в режимі редагування, в іншому випадку - режим вимкнено. file_slot_indx = 0 вказує на індекс файлового слоту, з яким ведеться взаємодія в поточний момент.

Двовимірний масив files_names використовується для зберігання імен файлів. Максимально 10 файлів зі змінної files_count, кожне ім'я може мати до 74 символів плюс символ '\0'. Двовимірний масив files_slots служить для вказівки доступності файлових слотів. 0 означає, що слот доступний, 1 - що він зайнятий. files_content – двовимірний масив для зберігання вмісту файлів. Кожен файл може мати розмір до 2000 байт. files_last_chr_indx масив для вказівки на поточний індекс останнього символу в кожному файлі.

```
// Work with files-----------------------------------------


bool edit_mode = false;
unsigned char file_slot_indx = 0;

unsigned char files_names[10][75]; // Array of file names, maximum 10 files, each 74 characters + '\0'
unsigned char files_slots[10][1]; // 0 - avaliable, 1 - taken
unsigned char files_content[10][2000]; //An array for storing the contents of files, a maximum of 10 files, each file up to 2000 bytes in
unsigned short files_last_chr_indx[10][1]; // The current index of the last character in the file
unsigned char files_count = 10;
```

Методи

Функція find_param_start призначена для знаходження початку параметрів команди в рядку текстового буфера консолі. Функція виконує ітерації по текстовому буферу, починаючи з поточного положення pntr, та переходячи вперед, доки знаходить пробіл. Під час цих ітерацій, забарвлює символи у вигляді білого тексту на чорному фоні.

Функція, read_param, використовується для зчитування параметра команди з текстового буфера консолі. Функція виконує ітерації по текстовому буферу, починаючи з поточного положення pntr, та переходячи вперед, доки не знайде пробіл (кінець параметра) або не досягне положення курсора. Під час цих ітерацій, копіює символи з текстового буфера в масив parameter.

Функція, find_file_name, використовується для пошуку імені файлу у масиві імен файлів. Вона використовує цикл for, щоб пройти через всі існуючі файли. Внутрішній цикл while порівнює кожен символ імені файлу у масиві files_names з відповідним символом у переданому параметрі до тих пір, поки не досягнутий кінець рядків обох масивів або не знайдено відмінності. Якщо обидва символи завершуються (досягнуто кінця рядка), то це означає, що імена файлів однакові, і встановлюється прапорець *same_name_file_exist в true, а індекс файлу file_indx зберігається у змінну *same_name_file_indx.

Масив рядків cmnds_names містить назви команд для взаємодії з системою, а масив вказівників на функції cmnds містить вказівники на різні функції, пов'язані із зазначеними командами. Якщо потрібно додати нову команду, то її потрібно занести до масиву назв команд, а також метод команди ініціалізувати в масиві методів команд.

Команди

cmnd_clear викликає clean_screen(), яка очищує екран, а потім встановлює cursor_pointer в 0. Після цього виводить версійне повідомлення "ChristOS - v 0.0.1" за допомогою функції print з чорним текстом та зеленим фоном. Функція clean_screen у свою чергу призначена для очищення екрана консолі. Вона використовує дві вкладені петлі для ітерації через кожен рядок

і кожний стовпець на екрані (25 рядків по 80 стовпців). У кожній клітинці консольного буфера, що представляє піксель екрана, встановлюються значення кольору. Кожен піксель у консольному буфері має два байти: перший байт відповідає за символ або фоновий колір, а другий байт за вибірковий колір тексту. У даній функції встановлюється чорний фон і білий текст для кожного пікселя, заснованого на визначених раніше константах BLACK_COLOR і WHITE_COLOR.

```
void cmnd_clear()
{
    clean_screen();
    cursor_pointer = 0;
    char *ver_msg = "ChristOS - v 0.0.1\n";
    print(ver_msg, BLACK_COLOR, GREEN_COLOR);
}
```

```
void clean_screen()
{
    for (unsigned char row = 0; row < 25; row++)
    {
        for (unsigned char col = 0; col < 80; col++)
        {
            *(framebuffer + row * (80 * 2) + col * (2)) = BLACK_COLOR;
            *(framebuffer + row * (80 * 2) + col * (2) + 1) = WHITE_COLOR;
        }
    }
}
```

cmnd_help виводить повідомлення довідки за допомогою функції print. Повідомлення містить список доступних команд, таких як help, clear, sleep, list, create, edit, read, delete.

```
void cmnd_help()
{
    char *new_line = "\n";
    print(new_line, BLACK_COLOR, BLACK_COLOR);

    char *msg_help = "======== Welcome to the system! ========\n== You can use the following
    print(msg_help, BLACK_COLOR, YELLOW_COLOR);
}
```

Функція print ітерується по кожному символу в рядку msg. Якщо символ – це символ нового рядка ('\n'), то курсор переміщується на початок наступного рядка (заокруглення вниз до наступного рядка). Якщо символ не є новим рядком, то відповідні дані вставляються в консольний буфер. У

випадку, якщо курсор перевищує розмір консольного буфера (1999 - максимальний розмір, що обрано у вас), викликається функція scroll яка, відповідає за прокрутку екрану вгору, щоб старий текст зникав, і новий текст з'являвся внизу. На кінці, викликається функція put_cursor, яка встановлює позначку курсора на визначену позицію.

```
void print(char *msg, unsigned char clr_bckg, unsigned char clr_fnt)
{
    while (*msg != '\0')
    {
        if (*msg == '\n')
        {
            cursor_pointer += 80 - cursor_pointer % 80;
        }
        else
        {
            *(framebuffer + cursor_pointer * 2) = *msg;
            *(framebuffer + cursor_pointer * 2 + 1) = (clr_bckg << 4) | clr_fnt;
            cursor_pointer += 1;
        }
        msg++;

        if (cursor_pointer > 1999)
        {
            scroll();
        }
        else
        {
            put_cursor(cursor_pointer);
        }
    }
}
```

Основна ідея scroll полягає в тому, що для кожного рядка (від 0 до 23) копіюється вміст рядка вище (рядок row + 1). Останній рядок (24-й) очищується, і курсор переміщується на початок нового рядка в самому низу. Копіювання виконується за допомогою функції memcpy, яка копіює блок пам'яті з одного місця в інше. Це дозволяє здійснити ефективне переміщення даних. Цикл for використовується для встановлення чорного кольору фону та білого кольору тексту. На кінці, курсор переміщується на початок нового рядка в самому низу, і викликається функція put_cursor, яка встановлює позначку курсора на визначену позицію.

```
void scroll()
{
    for (unsigned char row = 0; row < 24; row++) {
        unsigned int src_offset = row * 80 * 2;
        unsigned int dest_offset = src_offset + 80 * 2;
        memcpy(framebuffer + src_offset, framebuffer + dest_offset, 80 * 2);
    }

    // unsigned int clear_offset = 24 * 80 * 2;
    // memset(framebuffer + clear_offset, BLACK_COLOR, 80 * 2);

    for (unsigned char col = 0; col < 80; col++) {
        unsigned int offset = 24 * (80 * 2) + col * 2;
        framebuffer[offset] = BLACK_COLOR;
        framebuffer[offset + 1] = WHITE_COLOR;
    }

    cursor_pointer = 80 * 24;
    put_cursor(cursor_pointer);
}
```

cmnd_sleep викликає save_console_state(), яка зберігає стан консолі. Потім вона викликає clean_screen(), щоб очистити екран, і встановлює флаг anim_sleep_continue в true. Це позначає, що анімація сну повинна продовжуватися.

```
void cmnd_sleep()
{
    save_console_state();

    clean_screen();

    anim_sleep_continue = true;
}
```

save_console_state() призначена для збереження стану консолі. cnsl_state_pntr = cursor_pointer – зберігає поточне положення курсора в консольному буфері в змінну cnsl_state_pntr. cursor_pointer = 0 – встановлює позначку курсора на початок консольного буфера. put_cursor(cursor_pointer); - встановлює позначку курсора на нову позицію (на початок буфера). За допомогою вкладених циклів for, зберігає символи та кольори кожного символу у консольному буфері (framebuffer) в відповідні масиви cnsl_state_chrs і cnsl_state_clrs.

```
void save_console_state()
{
    cnsl_state_pntr = cursor_pointer;

    cursor_pointer = 0;

    put_cursor(cursor_pointer);

    for (unsigned char row = 0; row < 25; row++)
    {
        for (unsigned char col = 0; col < 80; col++)
        {
            cnsl_state_chrs[row * 80 + col] =
                *(framebuffer + row * (80 * 2) + col * (2));

            cnsl_state_clrs[row * 80 + col] =
                *(framebuffer + row * (80 * 2) + col * (2) + 1);
        }
    }
}
```

Функція recover_console_state() відновлює стан консолі до попереднього збереженого стану, що був здійснений за допомогою функції save_console_state(). anim_offsite_X = 0; anim_offsite_Y = 0; anim_coord_X = 1; anim_coord_Y = 1 – скидаються змінні, які використовуються для анімації. cursor_pointer = cnsl_state_pntr – встановлює позначку курсора на те значення, яке було збережено в cnsl_state_pntr. cnsl_state_pntr = 0 – скидає змінну cnsl_state_pntr на нуль. put_cursor(cursor_pointer) – встановлює позначку курсора відповідно до нового значення cursor_pointer. За допомогою вкладених циклів for, відновлює значення символів та кольорів кожного символу в консольному буфері (framebuffer) з відповідних масивів cnsl_state_chrs і cnsl_state_clrs. Після відновлення кожного символу, змінні cnsl_state_chrs і cnsl_state_clrs скидаються на BLACK_COLOR.

```
void recover_console_state()
{
    anim_offsite_X = 0;
    anim_offsite_Y = 0;
    anim_coord_X = 1;
    anim_coord_Y = 1;

    cursor_pointer = cnsl_state_pntr;

    cnsl_state_pntr = 0;

    put_cursor(cursor_pointer);

    for (unsigned char row = 0; row < 25; row++)
    {
        for (unsigned char col = 0; col < 80; col++)
        {
            *(framebuffer + row * (80 * 2) + col * (2)) = cnsl_state_chrs[row * 80 + col];
            cnsl_state_chrs[row * 80 + col] = BLACK_COLOR;

            *(framebuffer + row * (80 * 2) + col * (2) + 1) = cnsl_state_clrs[row * 80 + col];
            cnsl_state_clrs[row * 80 + col] = BLACK_COLOR;
        }
    }
}
```

Функція cmnd_list призначена для виведення списку файлів на екран. Загальна ідея цієї функції полягає в тому, що вона проходить через усі слоти для файлів, виводить певні мітки та імена файлів на екран. Якщо слот порожній, то виводять лише прочерки.

```
void cmnd_list()
{
    char *new_line = "\n";

    print(new_line, BLACK_COLOR, BLACK_COLOR);

    for (unsigned char slot_indx = 0; slot_indx < files_count; slot_indx++)
    {
        for (unsigned char cntr = 0; cntr < line_st_ofst; cntr++)
        {
            char *msg_space = " ";
            print(msg_space, BLACK_COLOR, BLACK_COLOR);
        }

        char *msg = "--";
        print(msg, BLACK_COLOR, YELLOW_COLOR);

        if (files_slots[slot_indx][0] == 1)
        {
            unsigned char shift_name = 0;

            print(&files_names[slot_indx][shift_name], BLACK_COLOR, YELLOW_COLOR);
        }
        else
        {
            //slot is empty
        }

        print(new_line, BLACK_COLOR, YELLOW_COLOR);
    }
}
```

Файлова система

cmnd_create функція отримує вказівник pntr_aftr_cmnd, який вказує на поточну позицію після команди "create". Викликається find_param_start для пошуку початку параметрів. Визначається, чи був наданий параметр, і виводиться відповідне повідомлення. Якщо параметр існує, перевіряється, чи існує файл з таким самим ім'ям. Якщо файл існує, виводиться відповідне повідомлення. Якщо ні, шукається вільний слот для файлу. Якщо слотів немає, виводиться повідомлення про відсутність доступних слотів. Якщо є вільний слот, створюється новий файл із заданим іменем.

cmnd_edit функція отримує вказівник pntr_aftr_cmnd, який вказує на поточну позицію після команди "edit". Викликається find_param_start для пошуку початку параметрів. Визначається, чи був наданий параметр, і виводиться відповідне повідомлення. Якщо параметр існує, перевіряється, чи існує файл з таким самим ім'ям. Якщо файл існує, режим редагування вмикається, поточний файл ініціалізується, і його вміст виводиться на екран.

cmnd_read отримує вказівник pntr_aftr_cmnd, який вказує на поточну позицію після команди "read". Викликається find_param_start для пошуку початку параметрів. Визначається, чи був наданий параметр, і виводиться відповідне повідомлення. Якщо параметр існує, перевіряється, чи існує файл з таким самим ім'ям. Якщо файл існує, виводиться вміст файлу на екран.

cmnd_delete отримує вказівник pntr_aftr_cmnd, який вказує на поточну позицію після команди "delete". Викликається find_param_start для пошуку початку параметрів. Визначається, чи був наданий параметр, і виводиться відповідне повідомлення. Якщо параметр існує, перевіряється, чи існує файл з таким самим ім'ям. Якщо файл існує, відбувається видалення файлу (звільнення пам'яті, встановлення слоту як доступного).

Функція search_command, призначена для визначення наявності введеної команди серед збережених команд. Спочатку відбувається визначення кількості збережених команд, тобто функція обчислює кількість збережених команд у пам'яті на основі масиву cmnds_names. Далі йде

порівняння індивідуальних символів – запускається цикл, який порівнює введену команду з кожною збереженою. Використовуються вказівники та змінні для перевірки, чи співпадають символи. Порівняння частини команди: Якщо частина введеної команди співпадає зі збереженою, перевіряється, чи є символ після цієї частини пробілом або кінцем рядка. Повернення індексу: Якщо знайдена відповідність, та символ після команди - пробіл або кінець рядка, функція повертає індекс збереженої команди. Повернення -1 при невідповідності, якщо жодна збережена команда не виявлена.

timer_tick_handler, виконує обробку таймерного інтервалу та відповідає за анімацію у режимі "screensaver". Якщо флаг anim_sleep_continue встановлено, функція починає з очищення екрану, викликаючи clean_screen(). Рух анімації по горизонталі та вертикалі (anim_coord_X і anim_coord_Y) та визначається, чи потрібно змінювати напрямок руху, або продовжувати в тому ж напрямку. Перевіряється, чи анімація не виходить за межі екрану. Якщо досягнуті межі, змінюється напрямок руху. Оновлення позиції анімації: Оновлюються змінні anim_offsite_X та anim_offsite_Y згідно з напрямком руху. Використовуючи змінні row, col та curr_char, функція проходить крізь рядки та стовпці, записуючи символи зображення на екран. Кожен символ отримує відповідний колір ((BLACK_COLOR << 4) | RED_COLOR). Оновлення індексів та позначення завершення анімації: Індекси slide_msg, row, col та curr_char використовуються для правильного виведення зображення. Якщо кінець зображення досягнутий ('\0'), анімація вважається завершеною, і відповідно, anim_sleep_continue встановлюється в false.

Висновок

Даний код відображає намагання покращити навички роботи з апаратним забезпеченням шляхом програмування низькорівневих елементів без використання стандартних засобів операційної системи.

Існуючий проект дозволяє взаємодіяти з консоллю, виконувати базові команди та демонструє анімаційні ефекти. Однією з ключових особливостей є створення гнучких програмних компонентів, таких як система керування

командами (cmnds та cmnds_names), робота із файлами, анімація та збереження/відновлення стану консолі. Ці компоненти дозволяють легко додавати нові функціональності чи покращати існуючі, роблячи систему більш гнучкою та розширюваною.

Програма використовує низькорівневі концепції, такі як маніпуляція буфером екрану, безпосереднє взаємодія зі структурою пам'яті та робота з апаратним таймером. Це надає можливість глибоко взаємодіяти з апаратурою та оптимізувати роботу системи. Загалом, цей проект служить хорошим прикладом самостійної імплементації операційного середовища, де програміст може ефективно використовувати свої навички для взаємодії з апаратним забезпеченням та створення власних низькорівневих компонентів.