Embedded System Software 프로젝트

(수행 결과 보고서)

과목명: [CSE4116] 임베디드시스템소프트웨어

담당교수: 서강대학교 컴퓨터공학과 박 성 용

학번 및 이름: 20181598, 김광민

개발기간: 2023. 06. 22. - 2023. 06. 25.

최 종 보 고 서

I. 개발 목표

한 학기 동안 습득한 리눅스 시스템 프로그램 및 안드로이드 프로그래밍 기법을 이용해 실습 보드 상에서 임베디드 소프트웨어를 개발하고 발표한다.

II. 개발 범위 및 내용

가. 개발 범위

- 안드로이드 프로그램 개발
- JNI 함수 작성
- game module 개발

나. 개발 내용

- 안드로이드 프로그램 개발
  : 게임 규칙을 설명하고, 게임을 시작할 수 있게 device driver를 제어하는 프로그램 개발
- JNI 함수 작성
- game module 개발
  : device driver 개발
  fpga device control
  interrupt를 통해 HOME, BACK key에 해당하는 동작 구현
  숫자야구 게임을 진행할 수 있도록 함수 작성

III. 추진 일정 및 개발 방법

가. 추진 일정

- 안드로이드 프로그램 개발 : 23.06.22 ~ 06.22
- JNI 함수 작성 : 06.23 ~ 06.23
- game module 개발 : 06.24 ~ 06.25

나. 개발 방법

- 안드로이드 프로그램 개발
  1. class MainActivity

```java
public class MainActivity extends Activity {
    public native int openDevice();

    public native void startNumbaseball(int fd);

    public native void closeDevice(int fd);

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        System.loadLibrary("numbaseball");

        final int fd = openDevice();

        Button startButton = (Button) findViewById(R.id.startButton);
        Button ruleButton = (Button) findViewById(R.id.ruleButton);
        startButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Toast.makeText(MainActivity.this, "Game Started!",
                        Toast.LENGTH_SHORT).show();
                startNumbaseball(fd);
                closeDevice(fd);
            }
        });

        ruleButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Intent intent = new Intent(MainActivity.this,
                        RuleActivity.class);
                startActivity(intent);
            }
        });
    }
}
```

해당 activity에는 2개의 button이 존재한다. Rule button을 누르면 startActivity() 함수를 통해 새 창에서 rule을 설명하도록 하고, start button을 누르면 JNI 함수를 통해 게임을 시작한다.

2. class RuleActivity

```java
public class RuleActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_rule);
    }
}
```

해당 activity에서는 activity_rule.xml에 있는 내용을 단순히 보여주기만 한다.

- JNI 함수 작성 (jni/numbaseball.c)

```
JNIEXPORT jint JNICALL Java_com_example_numbaseball_MainActivity_openDevice
(JNIEnv *env, jobject this){
    int fd = open("/dev/numbaseball", O_RDWR);

    if (fd == -1) {
        LOGV("Error opening device file\n");
        return -1;
    }

    return fd;
}


JNIEXPORT void JNICALL Java_com_example_numbaseball_MainActivity_startNumbaseball
(JNIEnv *env, jobject this, jint fd){
    write(fd, NULL, 0);

    return;
}



JNIEXPORT void JNICALL Java_com_example_numbaseball_MainActivity_closeDevice
(JNIEnv *env, jobject this, jint fd){
    close(fd);

    return;
}
```

- game module 개발

  1. device driver 개발

```
// when insmod
static int __init device_init(void) {
    // register driver
    major = register_chrdev(DEVICE_MAJOR_NUMBER, DEVICE_NAME, &device_fops);

    if (major < 0) {
        printk("Registering char device failed with %d\n", major);
        return major;
    }

    printk("Timer device major number : %d\n", DEVICE_MAJOR_NUMBER);

    //create device file
    cls = class_create(THIS_MODULE, DEVICE_NAME);
    device_create(cls, NULL, MKDEV(DEVICE_MAJOR_NUMBER, 0), NULL, DEVICE_NAME);

    printk("Timer device file created : /dev/%s\n", DEVICE_NAME);

    fpga_iomap_devices();

    return SUCCESS;
}
```

insmod 명령어로 인해 수행되는 부분으로, driver를 등록하고 device file을 생성한다. 이때, 생성되는 device file을 안드로이드 app 상에서 control하기 위해서는 권한을 수정해야 한다. fpga_iomap_devices()를 통해 게임 진행 간 필요한 fpga

device들을 iomap 한다.

```
// when rmmod
static void __exit device_exit(void) {
    //remove device file
    device_destroy(cls, MKDEV(DEVICE_MAJOR_NUMBER, 0));
    class_destroy(cls);

    //unregister driver
    unregister_chrdev(DEVICE_MAJOR_NUMBER, DEVICE_NAME);

    fpga_iounmap_devices();
}
```

rmmod에 의해 수행되는 함수로, device file을 제거하고 driver를 해제한 후 fpga_iounmap_devices()를 통해 fpga device들을 unmap한다.

```
//when open(), check if device is already opened
static int device_open(struct inode* inode, struct file* file) {
    if (already_open != 0) {
        return -EBUSY;
    }
    already_open = 1;

    inter_open();

    return SUCCESS;
}
```

open()에 의해 수행되는 함수로, already_open을 통해 이미 열려있는지 확인하고, inter_open()을 통해 interrupt를 등록한다.

```
//when close()
static int device_release(struct inode* inode, struct file* file) {
    already_open = 0;

    inter_release();

    return SUCCESS;
}
```

close()에 의해 수행되는 함수로, already_open을 0으로 바꿔 열려있지 않음을 명시하고, inter_release()로 등록된 interrupt를 free해 준다.

```
// when write()
static int device_write
(struct file *file, const char __user *buf, size_t count, loff_t *f_pos) {

    startGame();

    return 0;
}
```

write()에 의해 수행되는 함수로, 게임을 시작시킨다.

2. fpga device control

```c
void fpga_dot_write(const int digit) {
    int i;

    for (i = 0; i < 10; i++) {
        outw(fpga_dot_digit[digit][i] & 0x7F, (unsigned int) fpga_addr[DOT] + i * 2);
    }
}
```

fpga dot device에 write 하는 함수이고, digit은 현재 score를 의미한다.

```c
// fnd output
// parameter : 4 digits
void fpga_fnd_write(int num0, int num1, int num2, int num3) {
    unsigned short int value_short = 0;
    value_short = value_short + ((unsigned short int) num0 << 12);
    value_short = value_short + ((unsigned short int) num1 << 8);
    value_short = value_short + ((unsigned short int) num2 << 4);
    value_short = value_short + (unsigned short int) num3;
    outw(value_short, (unsigned int)fpga_addr[FND]);
}
```

fpga fnd device에 write하는 함수이고, 각 num을 현재 유저가 입력한 digit을 의미한다.

```c
// led output
// parameter : current runners on base, out count
void fpga_led_write(const int *base, const int out) {
    unsigned short value = 0;

    if (out >= 1) {
        value = 1;
    }
    if (out >= 2) {
        value += (1 << 4);
    }

    if (base != NULL) {
        if (base[0] == 1) {
            value += (1 << 1);
        }
        if (base[1] == 1) {
            value += (1 << 6);
        }
        if (base[2] == 1) {
            value += (1 << 3);
        }
    }
    else {
        value = 0;
    }

    outw(value, (unsigned int) fpga_addr[LED]);
}
```

fpga led device에 write하는 함수이고, base는 주자의 출루 여부, out은 아웃카운트를 나타낸다.

```c
// switch input
// Return : # of which switch is pressed
int fpga_switch_read(void) {
    int i;
    unsigned short int value;

    for (i = 0; i < SWITCH_CNT; i++) {
        value = inw((unsigned int) fpga_addr[SWITCH] + i * 2);
        if ((value & 0xFF) != 0) {
            return i + 1;
        }
    }

    return -1;
}
```

fpga switch device를 통해 input을 받는 함수이다. 1부터 9까지의 수를 하나씩 입력받는다.

```c
void fpga_text_lcd_write(int ball, int strike) {
    int i;
    unsigned short int value;
    unsigned char buffer[TEXT_LCD_BUFFER_SIZE + 1] = {0};

    memset(buffer, ' ', TEXT_LCD_BUFFER_SIZE);

    if (ball != -1) {
        buffer[0] = ball + '0';
        buffer[1] = 'B';
        buffer[3] = strike + '0';
        buffer[4] = 'S';
    }
    buffer[TEXT_LCD_BUFFER_SIZE] = '\0';

    for (i = 0; i < TEXT_LCD_BUFFER_SIZE; i += 2) {
        value = (buffer[i] & 0xFF) << 8 | (buffer[i + 1] & 0xFF);
        outw(value, (unsigned int) fpga_addr[TEXT_LCD] + i);
    }
}
```

fpga text lcd device에 write하는 함수이고, 현재의 볼 카운트를 출력한다.

3. interrupt

```c
void inter_open(void) {
    int ret, irq;

    gpio_direction_input(HOME);
    irq = gpio_to_irq(HOME);
    ret = request_irq(irq, home_handler, IRQF_TRIGGER_RISING, "home", 0);

    gpio_direction_input(BACK);
    irq = gpio_to_irq(BACK);
    ret = request_irq(irq, back_handler, IRQF_TRIGGER_RISING, "back", 0);
}
```

HOME, BACK key에 interrupt를 등록한다.

```
void inter_release(void) {
    free_irq(gpio_to_irq(HOME), NULL);
    free_irq(gpio_to_irq(BACK), NULL);
}
```

interrupt를 free한다.

```
static irqreturn_t home_handler(int irq, void* dev_id) {
    printk("HOME PRESSED\n");
    gameStatus->flag = 1;
    gameStatus->score = 10;

    return IRQ_HANDLED;
}
```

HOME key에 해당하는 handler이다. 현재 게임의 상태를 저장하고 있는 gameStatus의 두 멤버변수 값을 바꿔 게임이 종료될 수 있게 한다.

```
static irqreturn_t back_handler(int irq, void* dev_id) {
    printk("current status\n");
    printk("base : %d%d%d\n", gameStatus->base[0], gameStatus->base[1],
            gameStatus->base[2]);
    printk("out : %d\n", gameStatus->out);
    printk("score : %d\n", gameStatus->score);
    printk("inning : %d\n", gameStatus->inning);
    printk("flag : %d\n", gameStatus->flag);
    printk("hitternum : %d%d%d%d\n", gameStatus->hitterNum[0],
    gameStatus->hitterNum[1], gameStatus->hitterNum[2], gameStatus->hitterNum[3]);
    printk("usernum : %d%d%d%d\n", gameStatus->userNum[0], gameStatus->userNum[1],
    gameStatus->userNum[2], gameStatus->userNum[3]);
    printk("ball, strike : %dB%dS\n", gameStatus->ball, gameStatus->strike);

    return IRQ_HANDLED;
}
```

BACK key에 해당하는 handler이다. 현재 게임의 상태를 모두 출력한다.

4. 숫자야구 구현

```
typedef struct _payload {
    int base[3];   // base with runner
    int out;    // out count
    int score;  // baseball score (shown in fpga_dot)
    int inning;  // # of rounds player survived
    int flag;   // flag to indicate hitter change
    int hitterNum[4];   // random 4 digit
    int userNum[4]; // user 4 digit
    int ball;   //ball count
    int strike; // strike count
} payload;
```

숫자야구에서 사용되는 payload이고, 다음과 같이 gameStatus로 관리한다.

```
payload gameStatusInstance;
payload *gameStatus = &gameStatusInstance;
```

```c
void startGame() {
    int i;

    initializePayload();
    fpga_initialize();

    while (gameStatus->score <= 4) {
        hitterEnter();

        while (gameStatus->flag == 0) {
            chooseBall();
            throwBall();
            for (i = 0; i < 4; i++) {
                gameStatus->userNum[i] = 0;
            }
        }

        hitterChange();
        inningChange();
        fpga_print();
    }

    printk("You survived %d innings!\n", gameStatus->inning);

    fpga_initialize();
}
```

숫자야구는 위와 같은 flow로 진행된다. 게임을 시작하면 gameStatus와 fpga device들을 초기화한다.

```c
void hitterEnter() {
    int i, j;
    int random;

    for (i = 0; i < 4; i++) {
        get_random_bytes(&random, sizeof(int));
        gameStatus->hitterNum[i] = (((unsigned int)random) % 9) + 1;
        for (j = 0; j < i; j++) {
            if (gameStatus->hitterNum[i] == gameStatus->hitterNum[j]) i--;
        }
    }

    printk("gerenated ran num : %d%d%d%d\n", gameStatus->hitterNum[0],
    gameStatus->hitterNum[1], gameStatus->hitterNum[2], gameStatus->hitterNum[3]);
}
```

위 함수에서는 get_random_bytes()함수를 통해 무작위 4 개의 digit을 hitterNum 배열에 저장한다. 이때, 선택된 무작위 digit이 서로 겹치지 않도록 보장한다.

```
void chooseBall() {
    int i, j;
    int temp;

    for (i = 0; i < 4; i++) {
        temp = fpga_switch_read();
        if (temp == -1) {
            i--;
        }
        else {
            for (j = 0; j < i; j++) {   // if duplicated num, try again
                if (temp == gameStatus->userNum[j]) {
                    temp = 0;
                    i--;
                }
            }
            if (temp != 0) {
                gameStatus->userNum[i] = temp;
            }
            msleep(500);
        }
        fpga_fnd_write(gameStatus->userNum[0], gameStatus->userNum[1],
        gameStatus->userNum[2], gameStatus->userNum[3]);
    }
}
```

chooseBall()에선 fpga_switch_read()를 통해 user digit을 입력받는다. 이 경우에도 중복된 digit을 선택할 수 없도록 한다.

```
//check for direct match (out)
for (i = 0; i < 4; i++) {
    if (gameStatus->userNum[i] == gameStatus->hitterNum[i]) {
        gameStatus->flag = 1;   // change hitter
        gameStatus->out += 1;
        return;
    }
}

// check for homerun
for (i = 0; i < 4; i++) {
    for (j = 0; j < 4; j++) {
        if (gameStatus->hitterNum[i] != gameStatus->userNum[j]) {
            check++;
        }
    }
}
if (check == 16) {
    gameStatus->flag = 1;
    for (i = 0; i < 3; i++) {
        if (gameStatus->base[i] == 1) {
            gameStatus->score++;
            gameStatus->base[i] = 0;
        }
    }
    gameStatus->score++;
    fpga_print();
    return;
}
```

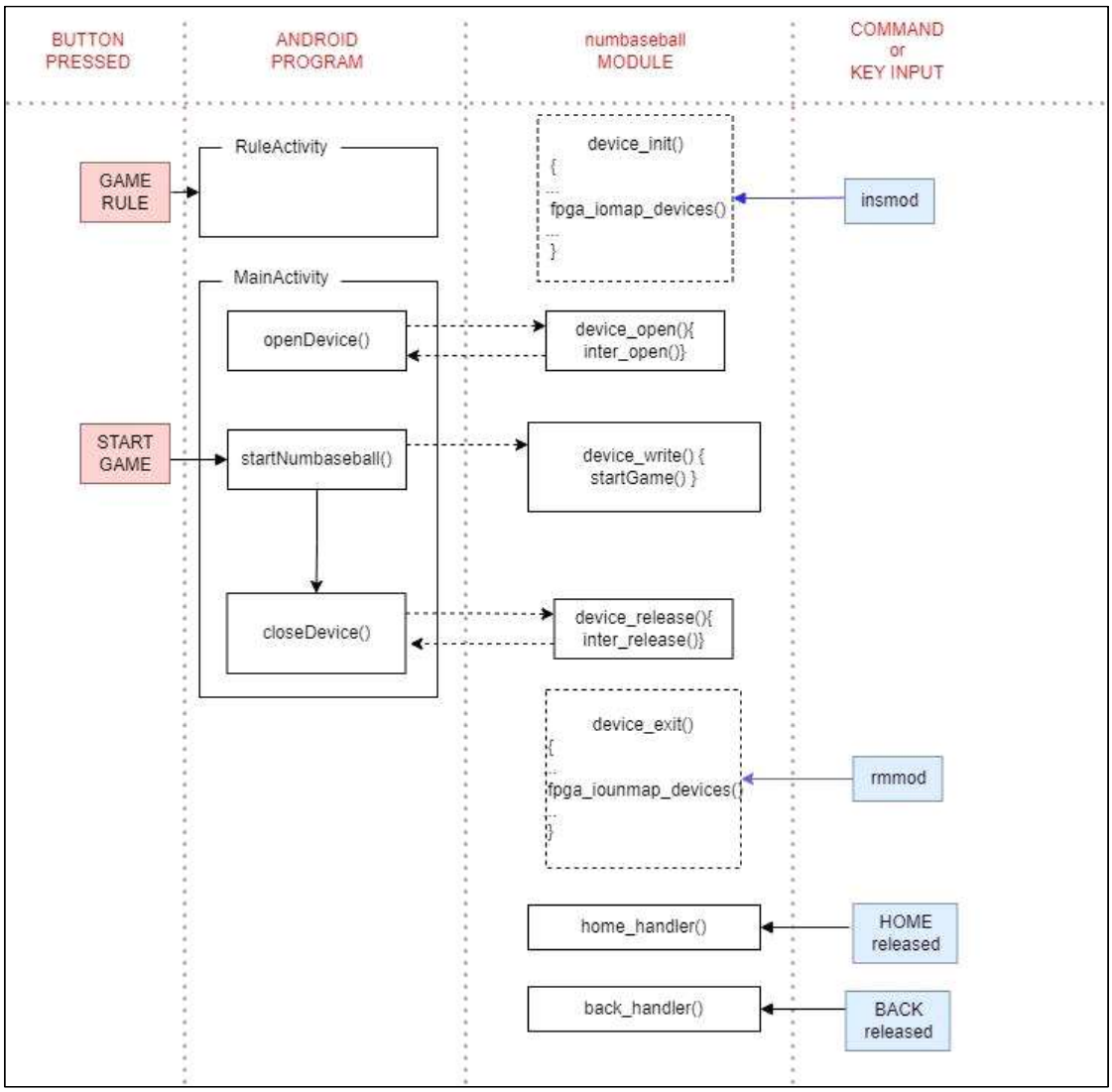
throwBall() 함수에서는 userNum[]과 hitterNum[]을 비교해 결과를 계산한다.

```
void hitterChange() {
    int i;
    for (i = 0; i < 4; i++) {
        gameStatus->userNum[i] = 0;
    }
    gameStatus->ball = 0;
    gameStatus->strike = 0;
    gameStatus->flag = 0;
}

void inningChange() {
    int i;

    if (gameStatus->out == 3) {
        gameStatus->out = 0;
        gameStatus->inning++;
        for (i = 0; i < 3; i++) {
            gameStatus->base[i] = 0;
        }
    }
}
```

hitterChange()와 inningChange()에서는 각각 타자가 바뀌어야 하는지, 이닝이 바뀌어야 하는지 여부를 판단한다.
숫자야구 게임은 실점한 점수가 5점 이상이면 종료된다.

IV. 연구 결과

안드로이드 프로그램 및 numbaseball module의 flowchart는 다음과 같다.



V. 기타

다음은 프로젝트에 추가하고 싶었지만 구현하지 못한 것들이다.

1. 현재 안드로이드 app에서 start game 버튼을 누르면 android app은 numbaseball module이 동작을 멈출 때까지 기다리게 된다. 이를 thread를 이용해 개선하려 했으나 구현하지 못했다.

2. JNI 함수를 추가해 numbaseball module로부터 gameStatus의 정보를 가져오고 android app에서 U/I를 구현해 현재 게임 상태를 android app 상에서도 보여주고, 게임이 끝나면 플레이어들의 점수를 기록해놓는 점수판 기능을 구현하려 했으나 하지 못했다.

3. 현재 게임의 종료는 HOME 버튼에 등록된 interrupt를 통해서 진행하는데, 이는 HOME button을 누른 뒤 switch를 4번 눌러야 종료가 되는 문제가 있다. 이를 thread와 wait

queue를 이용해 android app 상에서 게임을 종료할 수 있게 개선하고 싶었지만 구현하지 못했다.
이번 프로젝트를 통해 한 학기 동안 배운 내용들을 다방면으로 복습해볼 수 있어서 좋았으나, 개발 시작을 지나치게 늦게 해 깊이있는 프로젝트를 진행하지 못한 점이 큰 아쉬움으로 남는다.