Embedded System Software 과제 1

(과제 수행 결과 보고서)

과목명: [CSE4116] 임베디드시스템소프트웨어
담당교수: 서강대학교 컴퓨터공학과 박 성 용

학번 및 이름: 20141494, 강동욱
개발기간: 2020. 04. 23. -2020. 04. 26.

최 종 보 고 서

I. 개발 목표

- 디바이스 컨트롤과 IPC를 이용하여 주어진 Clock, Counter, Text editor, Draw board를 구현한다. IPC는 shared memory 방법을 이용하여 IPC를 구현한다.

II. 개발 범위 및 내용

가. 개발 범위 및 개발 내용

1. Shared memory 방법을 이용한 IPC 구현.

메인 프로세스에서 포크된 입력 프로세스, 출력 프로세스간의 데이터 공유를 위한 IPC를 system v shared memory 기법을 사용하여 구현한다.

2. Device Control 구현

LED 디바이스를 mmap을 이용하여 컨트롤하고, 나머지 fpga text lcd, dot matrix, dot matrix, fnd 디바이스들을 device driver를 사용하여 제어하는 기능을 구현한다.

3. Clock, Counter, Text editor, Draw board 기능 구현

Device Control을 구현한 내용을 바탕으로, Input process에서 사용자 입력을 받아 shared memory IPC로 공유된 데이터를 사용하여 main process에서 연산하고 output process에서 Device control을 사용하여 기기에 출력하는 프로그램을 작성한다.

III. 추진 일정 및 개발 방법

가. 추진 일정

2020.04.24 소프트웨어 설계 및 IPC 구현
2020.04.25 ~ 2020.04.26 Device Control 및 기능 구현
2020.04.26. 디버깅 및 테스트, 보고서 작성

나. 개발 방법

Windows 10 환경에서 CLion 에디터를 사용한 코드작성
Oracle VM VirtualBox를 사용한 Ubuntu 환경에서 컴파일
Achro-I.MX6Q 보드에서 안드로이드 4.3 환경에서 실행 및 테스트

Ⅳ. 연구 결과

1. Shared memory 방법을 이용한 IPC 구현.

우선, input main output 3개의 프로세스 사이에 공유할 데이터를 버튼 입력, 현재 모드, 각 모드에 사용할 정보 3개로 정의하였고, input과 main에서는 버튼 입력 데이터를 main과 output에서는 현재 모드와 각 모드에서 사용할 정보를 사용하도록 설계하였다.

각 프로세스의 공유 메모리 사용을 정리하면 아래와 같다.

	button	mode	value
input	Read, Write	Read, Write	
main	Read	Read	Read, Write
output		Read	Read

각 공유 메모리는 하나의 프로세스에서만 수정이 일어나고, 주기적 실행을 기반으로 하므로 세마포어 또는 시그널 없이 구현하도록 하였다.

main 프로세스에서 fork 하기 전에 미리 공유 메모리를 생성하고, 전역변수로 shared memory id들을 저장한 후, fork 이후 input output 프로세스에서 해당 id를 사용하여 공유 메모리를 불러와 사용하였다. fork 한 자식 프로세스 사이에만 공유하는 메모리이기 때문에 공유메모리 생성시 IPC_PRIVATE를 키로 사용하였다.

value는 모드에서 사용하는 struct 중 가장 큰 값보다 크게 할당하고, 각 모드에 따른 struct를 정의하여 모드 변경시 초기화하고 해당 struct의 포인터로 캐스팅하여 사용하였다.

2. Device Control 구현

Device Control은 Input 프로세스에서 push switch, hardware key 컨트롤이 필요하고, output 프로세스에서 led mmap, fnd, text lcd, dot matrix를 사용한다.

```
// 디바이스의 fd를 저장할 변수들
int fpga_switch_device;
int hw_button_device;
int fpga_fnd_device;
int fpga_led_mmap;
int fpga_lcd_device;
int fpga_dot_device;
```

각 프로세스 시작 시 필요한 디바이스를 열고 파일 디스크립터를 저장하여 사용하였다.

```
void read_hw_key();
void read_fpga_key();
void set_fnd(int value);
void set_lcd_text(char *string);
void set_dot_matrix(unsigned char *num_matrix);
```

각 디바이스에서 정보를 읽거나, 변경, 표시하는 함수를 각각 만들어 캡슐화하여 사용하였다.

스위치 입력과 하드웨어 버튼 입력을 현재 버튼의 눌림 여부를 0.2초마다 판별하여 저장한다. 너무 길게 누르면 반복 입력되고, 짧게 누르면 입력되지 않는 문제가 있다.

3. Clock, Counter, Text editor, Draw board 기능 구현

1. 모드 변경 구현

Device control에서 구현한 하드웨어 키를 Input 프로세스에서 인식하고 모드 값을 변경한다. Main 프로세스에서 모드가 변경되었을 때 초기해야 하므로, Input 프로세스에서는 모드가 변경되었을 때 특정 값을 더해 모드가 변했음을 main 프로세스에 알리도록 하였다.

```
// 모드 변경되었음을 구분하기위한 큰 수
#define MODE_CHANGED 100
```

모드가 변경되었음을 인식한 main 프로세스는 각 모드에 맞는 struct로 공유 메모리의 value를 변경시킨다.

```
// 모드가 변경되었을 때, 각 모드에서 사용하는 정보를 초기화한다.
int i, j;
unsigned char *value_addr = (unsigned char *) shmat(value_mid, (unsigned char *) NULL, 0);
memset(value_addr, 0, 1000);
case CLOCK_MODE:
    a = (clock_values *) value_addr;
    a->bonus_time = 0;
    a->editable = False;
    break;
```

2. Clock 구현

Clock 기능이 output 프로세스에서 제대로 표기되기 위해서는, 현재 시각과 내가 증가시킨 시간, 그리고 LED 표시를 위한 수정모드 여부를 알아야 한다. 현재 시각은 time() 함수를 사용하여 쉽게 얻을 수 있으므로, 아래와 같은 struct를 공유 메모리에 저장하였다.

```
// clock 모드에서 사용할 값들
typedef struct _clock_values {
    int bonus_time;
    int editable;
} clock_values;
```

수정 모드는 editable 값 수정을 통해 구현하였고, 1분 증가는

bonus_time의 60증가, 1시간 증가는 bonus_time의 3600 증가로 구현하였다.

```
// 시간 수정 모드에서 1번 입력시 현재 시간 값을
// 시스템시간으로 설정하고 초기화한다.
if (clockValues->editable == True)
{
    clockValues->editable = False;

    gettimeofday(&timeval, NULL);
    timeval.tv_sec += clockValues->bonus_time;
    settimeofday(&timeval, NULL);
    clockValues->bonus_time = 0;
}
```

수정모드 종료시 settimeofday 함수를 통해 시스템 시간 bonus_time 만큼 증가시키는 것으로 시스템 시간 변경 기능을 구현하였다.

output 프로세스에서 디바이스에 정보를 표시하는 부분은 아래와 같이 구현하였다.

```
// clock_process 에서 추가된 bonus_time 만큼 시간 증가
now = time(NULL) + clockValues->bonus_time;
int hour = now / 3600 % 24;
int min = now / 60 % 60;

set_fnd(hour * 100 + min);
```

led 점멸은 editable 값을 참조해서 True(1)이면 첫 번째 LED를, False면 현재 시각을 2로 나눈 나머지에 따라 3,4번 LED가 점등하도록 구현하였다.

3. Counter 구현

counter 모드에서는 실제값과 지수만 저장하고 있으면, output에서 지수에 따라 출력해 줄 수 있다고 생각하여 아래와 같은 struct를 사용하였다.

```
// counter 모드에서 사용할 값들
typedef struct _counter_values {
    int exponent;
    int value;
} counter_values;
```

exponent에는 지수를, int에는 값을 저장한다.

1번 스위치 입력 시 다음 진수로 변경되어야 하므로, 다음 진수를 알기 위해 다음과 같은 배열을 사용하였다.

```
// counter 모드에서 현재 진수의 다음 진수를 알기위한 배열,
// next_exp_map[2] 는 2진수 다음으로 설정될 진수를 나타낸다.
unsigned char next_exp_map[11] = {0, 0, 10, 0, 2, 0, 0, 0, 4, 0, 8};
```

위의 배열을 사용해서 진수 변경은 아래와 같이 구현하였고, 현재의 exponent 값에 따라 버튼 입력시 증가시키는 값을 다르게하여 구현하였다.

```
if (button_addr[0] == True)
{
    // 표기 진수 변경
    counterValues->exponent = next_exp_map[counterValues->exponent];
}
else if (button_addr[1] == True)
{
    // 현재 표기의 백의 자리 증가
    counterValues->value += counterValues->exponent * counterValues->exponent;
}
```

output 프로세스에서는 현재 exponent와 value 값을 사용하여 적절한 LED, FND값을 표시한다. 10진수는 3자리만 표시하였다.

```
// 진수에 따라 표기 변환 후 fnd에 표시
if (exp == 10)
{
    set_led(0b01000000);
    set_fnd(val % 1000);
}
else if (exp == 2)
{
    set_led(0b10000000);
    fnd_val = int_to_four_digit(val, 2);
    set_fnd(fnd_val);
}
else if (exp == 4)
{
    set_led(0b00010000);
    fnd_val = int_to_four_digit(val, 4);
    set_fnd(fnd_val);
}
```

4. Text editor 구현

텍스트 에디터에서 사용할 값들은, 입력 횟수, 숫자 또는 문자 입력모드, 문자열, 연속입력을 판단하기 위한 이전 입력값, 현재 입력중인 문자의 인덱스로 정의하였다.

```
// text_editor 모드에서 사용할 값들
typedef struct _text_editor_values {
    int count;
    char is_letter_mode;
    char string[100];
    char prev_value;
    int editing_index;
} text_editor_values;
```

main 프로세스에서 현재 입력된 버튼에 따라 다른 동작을 구현하였다.

```
val->string[0] = '\0';
val->prev_value = -1;
val->editing_index = -1;
```

```
val->is_letter_mode = 1 - val->is_letter_mode;
val->prev_value = -1;
```

좌측과 같은 데이터 변경을 통해 초기화, 우측과 같은 데이터 변경을 통해 입력모드 변경하였다.

```
// 현재 입력중인 문자의 다음 값과 다른 버튼을 눌렀을 때 나올 첫 값에 대한 설정
start_text_map[0] = '.';
next_value_map['.'] = 'Q';
next_value_map['Q'] = 'Z';
next_value_map['Z'] = '.';
```

위와 같은 방식으로 배열들을 선언하여, 1번을 눌렀을 때 첫 값은 . 이고, 연속 입력 시 온점은 Q가 Q는 Z가 Z는 온점이 된다는 것을 저장해두었다.

위에 선언한 배열을 사용해서, 이전 입력한 값과 같은지 판별하고 같다면 next_value_map의 값으로 변경하고, 아니라면 입력 인덱스를 증가시키고 새로운 값을 쓰도록 하여 구현하였다.

```
if (val->prev_value == i)
{
    // 이전에 눌렀던 스위치를 누를 경우 마지막 값을 변경
    char now_editing_char = val->string[val->editing_index];
    val->string[val->editing_index] = next_value_map[now_editing_char];
    val->string[val->editing_index + 1] = '\0';
}
else
{
    // 이전에 눌렀던 스위치와 다른 스위치 입력시 새로운 문자 추가
    val->editing_index++;
    val->string[val->editing_index] = start_text_map[i];
    val->string[val->editing_index + 1] = '\0';
}
val->prev_value = i;
```

입력 영역 (32자) 초과 시 strcpy를 통해 한칸씩 앞으로 복사하고 editing_index를 감소시키도록 구현하였다.

output 프로세스에서는 저장된 스트링을 불러와 lcd에 표시하고, 카운트를 fnd에 표시하였다.

```
// text_editor_process에서 변경된 텍스트를 읽어와 lcd에 표시
strcpy(buffer, val->string);

set_lcd_text(buffer);
set_fnd(val->count);
```

dot matrix에 숫자, 문자 모드를 표기하기 위해 아래와 같은 배열을 만들어 표기하였다.

```
// dot matrix에 입력모드 표시
unsigned char num_matrix[10];
if (val->is_letter_mode == True)
{
    // A
    num_matrix[0] = 0b0011100;
    num_matrix[1] = 0b0110110;
    num_matrix[2] = 0b1100011;
    num_matrix[3] = 0b1100011;
    num_matrix[4] = 0b1100011;
    num_matrix[5] = 0b1111111;
    num_matrix[6] = 0b1111111;
    num_matrix[7] = 0b1100011;
    num_matrix[8] = 0b1100011;
    num_matrix[9] = 0b1100011;
}
else {...}
set_dot_matrix(num_matrix);
```

5. Draw board 구현

Draw board를 표시하기 위해서는, 입력 횟수, 현재 커서 위치, 해당 좌표의 on/off여부를 판별하기 위한 배열, 커서 표시여부가 필요하였다.

```
typedef struct _draw_board_values {
    int count;
    int cursor_point;
    unsigned char board[BOARD_ROW * BOARD_COL];
    char is_show_cursor;
} draw_board_values;
```

위와 같은 struct를 선언하여 cursor_point를 현재 커서 위치, board를 해당 좌표의 on/off 여부로 사용하였다. 이때 값과 인덱스는 ROW 값으로 나눈 값이 수직 좌표(row), ROW로 나눈 나머지가 수평 좌표(col)로 사용되었다.

main 프로세스에서의 연산을 설명하면, 모든 정보 초기화는 아래와 같이 구현하였고, 그림만 초기화는 board의 값만 초기화하는 것으로 구현하였다.

```
memset(val->board, False, BOARD_ROW * BOARD_COL);
val->count = 0;
val->cursor_point = 0;
val->is_show_cursor = True;
```

커서 표기 변경은 is_show_cursor 값 변경을 통해, 그림 반전은 board 값을 변경하는 것으로 구현하였다. 커서 위치 이동은 cursor_point의 값 변경으로 구현하였다. 커서에 점을 찍는 기능은 board의 커서 위치를 인덱스로 하는 값을 변경시켜 구현하였다.

```
    // 현재 커서위치의 값 변경
 else if (button_addr[4] == True)
 {
     val->board[val->cursor_point] = 1 - val->board[val->cursor_point];
 }
```

output 프로세스에서는 우선 char[row*col]로 저장된 정보를 device 컨트롤에 사용할 수 있는 값으로 변환한다.

```
// char[row*col] 로 저장된 정보를 unsigned char[row] 로 변환
int i, j;
unsigned char num_matrix[10];
for (i = 0; i < BOARD_ROW; i++)
{
    unsigned char temp = 0;
    for (j = 0; j < BOARD_COL; ++j)
    {
        temp += val->board[i * BOARD_COL + j] << (BOARD_COL - 1 - j);
    }
    num_matrix[i] = temp;
}
```

그 후, 커서 표시 모드라면 현재 커서 위치의 led를 1초마다 변경하기 위해 비트 연산을 통해 구현하였다.

```
// 1초마다 커서 표시
if (val->is_show_cursor == True)
{
    int row, col;
    unsigned char bit;
    time_t now = time(NULL);

    row = val->cursor_point / BOARD_COL;
    col = val->cursor_point % BOARD_COL;
    bit = 1 << (BOARD_COL - 1 - col);
    // 커서 위치 led on
    if (now % 2)
    {
        num_matrix[row] |= bit;
    }
    else // 커서 위치 led off
    {
        num_matrix[row] &= (0b1111111 - bit);
    }
}
```

변환 전에 임시 배열로 복사 후 수정하고 변환하는 편이 훨씬 쉽지만, 메모리 사용량과 cpu 사용량에서 이 방법이 낫다고 생각하여 그대로 두었다.

6. 종료 구현

fork할 때 자식의 pid를 저장해두었다가, input에서 back키 입력시 모드를 EXIT 모드로 변경하여 main에서 해당 모드를 인식하고 저장된 자식 프로세스를 종료하도록 구현하였다.

```
// 종료 입력시 프로그램 종료
if (mode_addr[0] == EXIT)
{
    kill(input_process_pid,SIGTERM);
    kill(output_process_pid,SIGTERM);
    exit(0);
}
```

V. 기타

- 리눅스 환경이 제대로 설정되지 않은 IDE 코드작성 후 가상머신으로 코드를 이동시킨 후 컴파일하고, 프로그램을 다시 임베디드 보드로 옮겨서 실행하면서 작성했기 때문에 개발 중에 낭비되는 시간이 많았다.
- 과제 명세서에 명시되어 있지 않거나 헷갈릴만한 사항이 몇 개 존재했다. '보드 시간'은 시스템 시간인지, 하드웨어 시간인지 헷갈렸고, Counter 기능은 10진수만 3자리로 출력하는지, 나머지도 3자리인지가 명시되어 있지 않았다. 이 경우 질문 게시판의 내용을 바탕으로 10진수만 3자리로 가정하고 개발하였다. 버튼을 2개 동시에 누를 때, FND count는 1이 증가할지 2가 증가할지 명시되어 있지 않았다. 이 경우 2가 증가한다고 가정하였다.
- 과제의 크기가 매우 크다고 생각하였다. 중복되는 부분이 적도록 작성했다고 생각하는데도 헤더파일까지 총 1000줄의 코드가 나왔으며 3일간 20시간이 넘는 시간이 걸렸다.