Embedded System Software HW #1

due date: 2019. 4. 15.

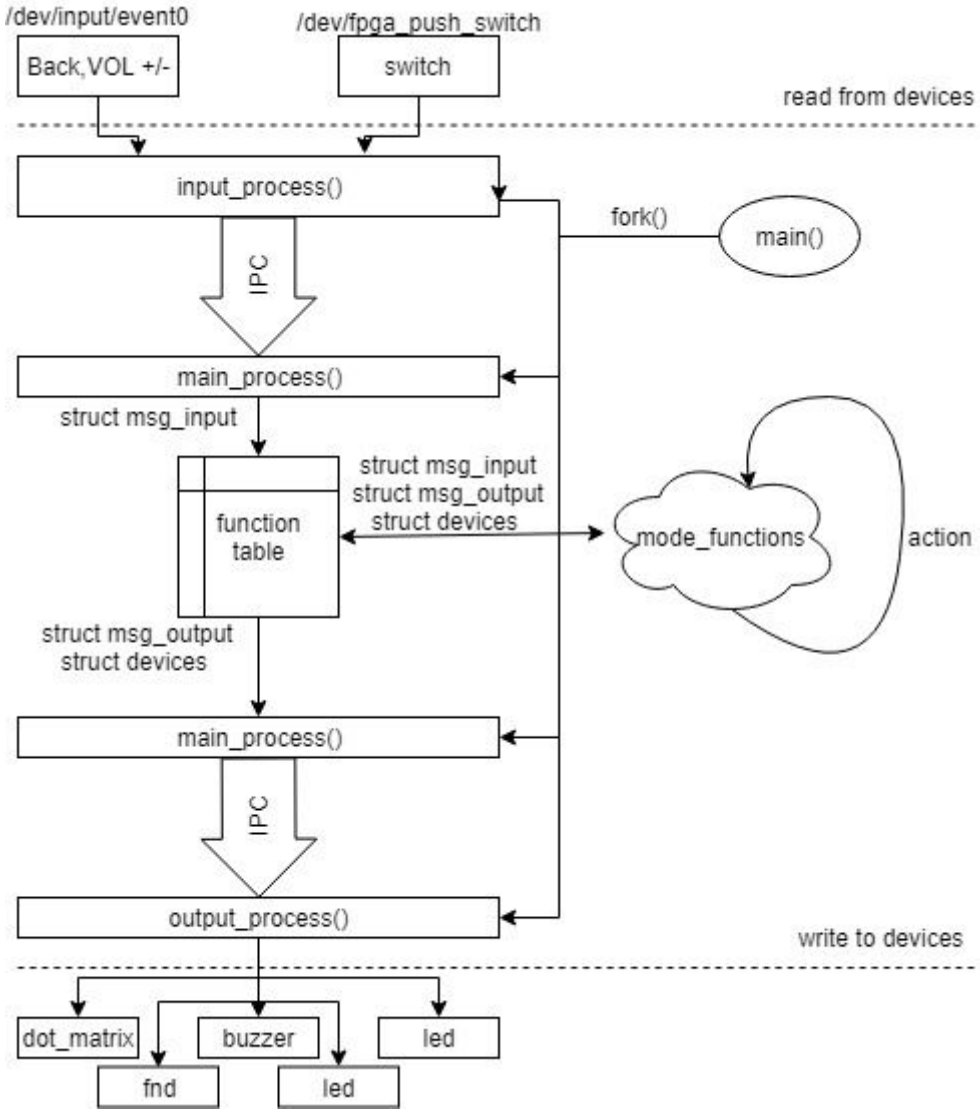
student name: 윤제형
student id: 20151575

목차

1. 목표

- 임베디드 환경인 주어진 보드에서 사용 가능한 device들을 이용하여 주어진 여러 mode (clock, counter, text editor, draw board)를 구현한다. 이 과정에서 device driver와 mmap을 이용한 device 제어를 적용해 보고 unix환경 에서의 다중 프로세스 생성 을 위한 fork()와 프로세스 간의 통신을 위한 IPC인 message queue기법을 사용 해 본다.

2. 순서도



- main()에 의해서 생성된 input_process, output_process와 main_process는 서로 ipc(메세지 큐)를 통해 소통한다. input_process 는 device들에서 push와 release를 감지하여 키 입력으로 간주된 경우 msg를 만들어 main_process로 보낸다 main_process는 입력받은 msg의 정보를 통해 해당하는 mode_function을 호출한다. mode_function은 현재 main에 임시로 저장된 device의 정보와 새로 입력된 device의 정보를 이용하여 output_process로 전달할 msg를 생성하고 이를 main_process로 전달한다. 이후 main_process는 output_process에 ipc를 통해 이를 전달하고 이를 받은 output_process는 그대로 device에 write하여 출력하거나 flag를 새워 시간마다 정해진 행동을 한다.

3. 제약 사항

1) main,input,output process fork

```c
int main(){
    pid_t pid_input=0, pid_output =0;

    pid_input = fork();
    if(pid_input == 0){
        input_process();
        return 0;
    }
    else if(pid_input < 0){
        printf("fail!\n");
        return 0;
    }

    pid_output = fork();
    if(pid_output == 0){
        output_process();
        return 0;
    }
    else if(pid_output < 0){
        waitpid(pid_input,NULL,0);
        printf("fail!\n");
        return 0;
    }
    main_process(pid_input,pid_output);

    return 0;
```

RMAL main.c c

- fork를 통해 input_process, output_process를 분리하고 main은 main_process를 실행시켰다.

2) IPC(message queue)

```
key_id = msgget((key_t)875,IPC_CREAT|0666);
if(key_id == -1){
    perror("error!!\n");
    exit(-1);
}
```
input_process.c c utf-8[unix]

```
    else if(msg.msgtype != 0){
        int a = msgsnd(key_id,&msg,sizeof(msg),0);
        if(a == -1){
            perror("error!!\n");
            EXIT_HANDLING_INPUT;
        }
    }
}
```
input_process.c c utf-8[unix] 83% ≡

- input process에 message queue를 사용하여 지속적으로 main에 msg를 send하였다.

```
key_input_id = msgget((key_t)875,IPC_CREAT|0666);
key_output_id = msgget((key_t)5975,IPC_CREAT|0666);

if(key_input_id == -1 || key_output_id == -1)
    EXIT_HANDLING(ERR);
```
main.c c utf-8[unix] 51% ≡

```
    n = msgrcv( key_input_id, &imsg, sizeof(imsg), 0, IPC_NOWAIT);
    if (n != -1)
```
main.c c utf-8[unix] 76% ≡ 103/135 □ :

```
    if(msgsnd(key_output_id,&oms
        EXIT_HANDLING(ERR);
}
```
main.c c

- main process에서는 message queue를 사용하여 지속적으로 input process에서 msg를 receive 하고 output process로 msg를 send한다.

```
key_id = msgget((key_t)5975,IPC_CREAT|0666);
```
output_process.c c utf-8[unix]

```
    temp = msgrcv( key_id, &msg, sizeof(msg), 0,IPC_NOWAIT) ;
    if(temp != -1){
```
output_process.c c utf-8[unix] 49% ≡ 75/151

- output process에 message queue를 사용하여 지속적으로 main으로부터 msg를 receive 하였다.

3) device driver, mmap

```
cse20151575@cspro9:~/EmbeSysSW/proj1$ grep -r write
global.h: * this structure has data ASAP write to real device
macros.h:    write(dev_dot,devices.dot_matrix,sizeof(devices.dot_matrix));\
output_process.c: * and Using data of msg, write to the devices
output_process.c:            write(dev_fnd,&msg.fnd_data,4);
output_process.c:                write(dev_text,msg.data.mode3.text_data,LEN_TEXT);
output_process.c:                write(dev_dot,devices.dot_matrix,sizeof(devices.dot_matrix));
output_process.c:                write(dev_dot,devices.dot_matrix,sizeof(devices.dot_matrix));
output_process.c:                        write(dev_buzzer,&msg.data.mode5.buzzer,1);
output_process.c:                write(dev_dot,devices.dot_matrix,sizeof(devices.dot_matrix));
output_process.c:                write(dev_dot,devices.dot_matrix,sizeof(devices.dot_matrix));
cse20151575@cspro9:~/EmbeSysSW/proj1$
```

```
fpga_addr = (unsigned long *)mmap(NULL, 4096, PROT_READ | PROT_WRITE, MAP_SHARED,dev_led, FPGA_BASE_ADDRESS);
led_addr=(unsigned char*)((void*)fpga_addr+LED_ADDR);

if (fpga_addr == MAP_FAILED) {
    printf("error\n");
    EXIT_HANDLING_OUTPUT;
}
 +0 ~0 -0  develop   output_process.c                    c   utf-8[unix]   39% ☰   59/151 ㏑  : 13  ☰ [68]trailing [1
```

- fnd,dot,buzzer,text 는 device driver를 이용하여 출력하고 led는 mmap을 통해 구현하였다.

4. 추가 구현

1) mode_game

- **use device : switch, buzzer, led(mmap), dot matrix, fnd**
- 목표: dot matrix에 나타난 9개의 블럭을 모두 채워라
- 이 게임은 기본적으로 dot matrix의 일부분만을 사용한다 dot matrix는 7x10 픽셀이므로 2x3 개의 픽셀을 한 블럭으로 하여 9개의 블럭을 사용 하고 있다.
  ex)
  bb|bb|bb0
  bb|bb|bb0
  bb|bb|bb0
  ---
  bb|bb|bb0
  bb|bb|bb0
  bb|bb|bb0
  ---
  bb|bb|bb0
  bb|bb|bb0
  bb|bb|bb0
  00|00|000
  b라고 적힌 블럭은 각각 스위치에 매핑된다.
  스위치를 누르면 해당 스위치 블럭과 그 블럭의 왼쪽 오른쪽 위 아래 블럭 모두 반전을 시킨다. 즉 누른 스위치의 십자모양의 주위를 1을 0으로 0을 1로 반전 시킨다. 해당 스위치의 위나 왼쪽 혹은 오른쪽 이나 아래가 없다면 그 부분을 제외하고 나머지를 반전 시킨다.
  mode가 시작되면 led에 불이 들어 오는데 이는 해당 mode의 level을 나타낸다. dot matrix에는 해당 level에 해당하는 문제가 출제 되어있고 이를 switch를 눌러 전체 블럭에 들어오게 한다면 buzzer가 울리고 다음 level의 led와 문제가 출력된다. 또한 이 game중 switch input의 개수를 fnd에 출력한다.