**Embedded 1st HW. Calculator on Board with IPC**

**(설계 프로젝트 수행 결과)**

**과목명: [CSE4116] 임베디드시스템소프트웨어**

**담당교수: 서강대학교 컴퓨터공학과 박 성 용**

**학번 및 이름: 20151543 민지우**

**개발기간: 2019. 04. 8. -2018. 04. 18.**

**최 종 보 고 서**

**I. 개발 목표**

- 디바이스 컨트롤과 IPC를 이용하여 주어진 Clock, Counter, Text editor, Draw board를구현한다.

**II. 개발 범위 및 내용**

- 자신들이 설계한 개발 목표를 달성하기 위하여 어떠한 내용의 개발을 수행할 지 그 범위와 개발 내용을 기술할 것.

**가. 개발 범위**

각 디바이스들을 컨트롤해서 동작하는 **응용프로그램**을 만든다.

**나. 개발 내용**

main, input, output process를 fork해 서로 ipc (메세지큐 이용 필수)를 이용해 통신 한 뒤, 주어진 모드에 따라서 디바이스를 컨트롤 한다. 이때 디바이스 컨트롤은 device driver와 mmap을 이용한다.

**III. 추진 일정 및 개발 방법**.

**가. 추진 일정**

10일(4.8~ 4.18)

**나. 개발 방법**

1. process끼리의 통신  
 1) main / input  
 signal + shared memory 사용  
 2) main / output

Message queue 사용

2. main: event handler 구조 사용

Input: device에 따라 여러 event들이 존재, 이를 main에 보내줌

Main: input에서 들어오는 event에 따라 mode 마다 등록해 놓은

event handler실행

3. function pointer을 이용해 mode table, event table, device table 생성  
 function pointer를 이용해, 메시지나 event에 따라 실행될 callback 함수들을 저장

4. output: 가변 인자 함수 사용

각 기기마다 필요한 parameter가 다르기 때문에

콜백함수 테이블에 저장할 때 가변인자 함수의 형태로 저장

보낼 때에는 해당 변수를 전부 character array에 넣어서 message로 보내고

받을 때에는 메시지를 받아서 기기에 따라 array를 해석하고

이를 가변인자 함수 table에서 기기에 맞는 함수를 실행

5. timer

각 모드마다 주기적으로 동작해야 하는 기능을 위해서, javascript의 setInterval() 와 비슷한 기능을 작동하는 함수를 제작

SIGALRM signal과 리눅스 itimer 기능을 사용  
사용자 입력으로 주어지는 시간 간격대로 입력으로 받는 callback 함수를 수행

6. log service

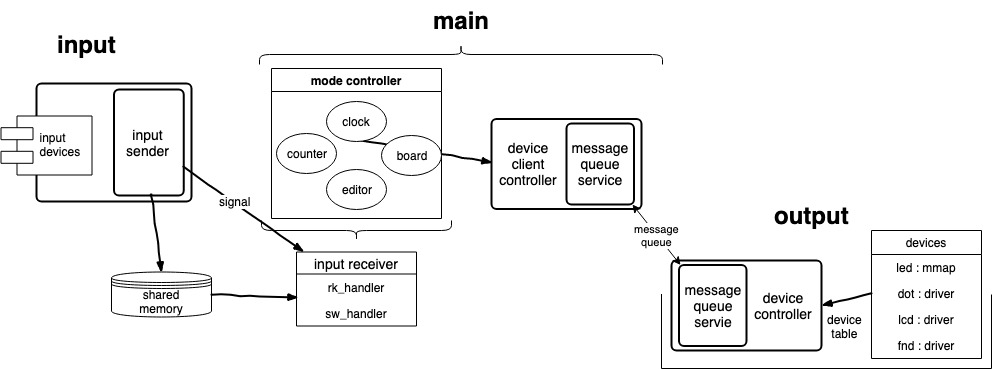
Log level에 따라서 log를 출력하는 service제작  
 macro 함수 및 가변 인자 함수를 이용해 로그를 출력

여러 모듈에서 사용할 수 있도록 함

**IV. 연구 결과**

- 최종 연구 개발 결과를 자유롭게 기술할 것.

**1. 합성 내용:**

****

**2. 제작 내용:** 개발 결과

- 구현한 컴포넌트들에 대한 역할 및 구현 방법에 관한 내용을 기술할 것. (예: 자료구조, 알고리즘).

1) **callback, callback\_no\_arg** type

Int (\*callback) (int, …): device table에 저장될 callback 함수, 가변인자 사용.  
 device들의 method를 wrapping하는 함수

Int (\*callback\_no\_arg) (void): device table에 저장될 callback 함수

Device init,exit 같은 인자 필요없는 함수들을 위한

2) **MESSAGE** 구조체

Output 과 main이 주고 받는 메시지 형식  
mq service에 정의  
device controller와 device client controller에서 사용

**Int device type**: output device type

**Int callback\_num**: 해당 디바이스의 callback # (해당 디바이스의 callback table의 index)

**Int arg\_cnt**: argument의 수

**Unsigned char data[MESSAGE\_SIZE]** : callback 함수에 넘겨줄 argument를 저장

3) **device\_itf** 구조체

Device controller에서 device를 제어하기 위한 device들의 interface 구조체

Device들을 제어할 수 있는 함수들을 저장

Device마다 하나씩 정의됨

**Callback\_no\_arg open**: 해당 device를 열때, initial하는 함수

**Callback\_no\_arg close**: 해당 device를 닫을 때 실행되는 함수

**Callback\* callbacks**: 해당 device를 제어할 수 있는 method들을 저장하는 table

4) **DEVICE\_TABLE**device controller가 제어할 수 있는 device들을 저장하는 table

Device\_itf를 저장

Device controller는 table에 등록된 device들만 다룰 수 있음

해당 table의 index가 device\_type

Message가 오면 message에서 device type에 따라 table에서 device\_itf를 가져옴  
message의 callback\_num에 따라 device\_itf.callbacks에 저장된 callback 함수를 실행  
이 때 message의 data를 인자로 넘겨줌

5) **mode** 구조체  
mode controller가 모드를 제어하기 위해 사용되는 구조체

각 모드마다 해당 모드로 바꼈을 때 수행해야하는 로직과, 종료 될 때 수행되는 로직들을 저장함.

Int num: 해당 모드의 index값

Int (\*init) (void): 해당 모드를 시작할 때 수행하는 함수, 보통 해당 모드의 변수들을 초기화하고, switch event에 대한 handler들을 해당 모드에 맞게 등록하고, 주기적으로 동작해야하는 로직들을 위해 timer를 설정하는 과정이 들어가 있음.

Int (\*exit)(void): 해당 모드를 종료할 때 수행하는 함수. 변수를 초기화하고, timer를 취소시키고, 등록한 handler들을 삭제하는 과정이 포함되어있음

6) **MODE\_TABLE**

Mode 구조체가 저장되어 있음.

Mode controller에서 제어하는 mode들을 저장하는 table  
여기에 등록된 mode들만 작동될 수 있음.

1. Mode controller

Rk\_handler로 mode controller의 method들이 등록됨.

Mode controller에서는 현재 모드를 저장하는 변수를 사용해, mode가 바뀔 때마다 MODE\_TABLE에서 이전 모드를 종료시키는 exit함수를 실행하고, 바뀐 모드에 대한 init함수를 실행시킴.

MODE\_TABLE에 선언된 함수들을 실행하는 주체

1. clock 모드

clock에 사용할 hour, minutes 선언해 사용

change모드에 들어오면 그 때 tmp\_hour, tmp\_minutes를 사용해 변경될 시간 정보를 저장.

시작할 때 timer로 1분 마다 분을 증가시키고 fnd를 갱신하는 로직을 등록하였음. 모드가 변경 될 때마다 timer를 취소하고 새로운 타이머를 등록하고, 각 모드의 함수들을 실행하였음.

변경모드일 경우, 1초마다 led를 번갈아 가면서 키도록하는 로직을 timer에 등록하고, fnd에 tmp\_hour, tmp\_minutes를 보여주도록 하였음.

1. count 모드

count값을 저장하는 변수를 선언해 사용  
사용하는 number system 구조체를 선언해 table로 만들어 저장하였음.

Number system마다 최대값과, light #를 저장하였음.

Count가 변해 그릴 때마다 현재 number system에 맞게 count를 숫자 문자열로 바꾸고 fnd에 출력하였음.

그리고 number system이 달라질 때마다 현재 system에서 사용하는 light#를 키도록 하였음.

1. text\_led모드

현재 text를 저장하고, 가장 최근에 눌린 버튼의 정보와 해당 버튼이 눌린 횟수를 저장함.

새로운 event가 들어올 때 이전의 버튼과 같은 버튼이면, text의 마지막 문자를 삭제하고, map alpha에서 다음 문자를 가져와 추가함.

Text의 길이가 buffer크기를 초과할 경우, temp를 사용해 앞글자를 제외한 문자열을 복사해 text위에 다시 쓰도록 하였음.

1. dot matrix 모드

bit operation을 사용  
현재 보드에 그려질 내용을 저장하는 배열을 선언하여

event가 들어오면 그때마다 수정된 배열을 다시 그려주었음

커서가 깜빡이면, 보드에 그려지는 내용과 보드의 논리적인 정보가 달라짐. (커서 위치가 현재는 채워져야하는 데 깜빡이게 되면 그려질 때 비게 그려져야할 수 있음) 그래서 현재 커서에 대한 정보를 저장해둠.

커서의 상태가 변화될 때마다 해당 정보를 갱신하고, 보드에 이전의 커서정보를 적용함.

1. Fnd count  
   text, draw board 모드에서 사용하는 fnd count는 동일한 기능을 수행하므로 모드에서 분리하였음.  
   count를 저장하는 변수를 선언하여, 모드가 바뀌면 초기화해주고, 각 모드에서 switch event에 해당하는 경우, inc\_count를 사용해 fnd값을 하나씩 늘려줌.

4)input 디바이스 읽기

Switch와 read key의 디바이스 파일을 연 후에, while문에서 계속 read를 하여 버튼 입력이 생기면, 읽어서 shared memory의 switch와 read key정보를 저장하는 위치에 저장하게 하였다.

그 뒤에 main에게 shared memory를 읽도록 SIGUSR을 보냄

5)output 읽기

Message queue에 메시지가 들어올 때까지 block되다가, message가 들어오면 그때 message를 읽어 message에 지정된 device의 callback함수를 수행한다.

**3. 시험 및 평가 내용:**

- 평가 방법에 대한 설명을 기술하라.(수행 sequence 등)

- 자신들의 결과물이 갖는 **보건 및 안정**, **생산성과 내구성**에 대하여 반드시 기술할 것.

평가 방법:

1. board에 main, input, output 실행 파일을 모두 옮김

2. 그 뒤 main을 실행

3. 모드에 따라 기능을 사용하면 됨

보건 및 안정성

Input의 경우, 버튼을 계속해서 누르고 있어도 이를 하나의 event로 파악  
여러 버튼을 누르고 있다가 중간에 하나를 떼어도 하나의 event로 파악  
그래서 두개를 동시에 정확하게 누르지 않아도 됨.

Output의 경우 message queue를 사용해, main에서 보내는 message의 순서에 따라서 기능을 수행하므로 안정성이 높음. 하나의 디바이스를 접근하는 중간에 다른 디바이스를 접근하는 것과 같은 행동을 하지 않음.

Main의 경우 SIGUSR1, SIGUSR2를 사용해, 해당 handler를 수행할 동안 중간에 해당 signal이 또 들어와도 중간에 멈추지 않고, 수행하고 있음. 다만 중간에 들어오는 signal들은 무시됨. 이는 안정성을 떨어트림.

Input은 shared memory에 write한 뒤 main에 signal을 보내고 main은 그때 memory에서 읽어오는데, 읽어오는 중간에 input이 write를 할 수 도있음. 이런 sync문제를 고려하지 못해 안정성이 떨어짐.

생산성

Input의 경우 user의 유효한 event가 들어올 경우에만 shared memory에 접근하고, main의 경우 input에서 signal이 들어올 때만 shared memory에 접근하기 때문에 불필요한 shared memory접근을 줄여 생산성을 높임

Output의 경우 message가 들어올 때까지 block되므로, 불필요한 cpu 자원을 줄여 생산성을 높였음

다만,

Input의 switch, read key에서 non\_block 모드를 사용해, 사용자의 input이 들어오지 않은 경우에도 수시로 check를 해, 불필요한 작업을 반복함  
이로인해 불필요한 cpu 연산이 수행됨

Main의 경우에도 signal이 들어올 때까지 block되어야하나, while문을 돌면서 프로세스가 죽지않도록 실행되고 있음. 결국 while문도 cpu의 불필요한 연산이므로 생산성을 떨어트림.

내구성

각 proccess마다 SIGINT가 들어올 시 의 handler를 등록해놓아, 중간에 interrupt가 들어와도 정상적으로 종료할 수 있음  
다만, segment fault와 같은 오류가 세 프로세스 중 하나라도 발생하면 다른 프로세스가 알지 못하고, 계속해서 프로세스가 실행됨. 직접 사용자가 프로세스를 꺼줘야함. 이런 부분에서는 내구성이 떨어짐.

**V. 기타**

- 기타 관련 내용을 기술할 것.

**1.** **연구 조원 기여도**: 민지우 100%

**2.** 기타 본 설계 프로젝트를 수행하면서 느낀 점을 요약하여 기술하라. 내용은 어떤 것이든 상관이 없으며, 본 프로젝트에 대한 문제점 제시 및 제안을 포함하여 자유롭게 기술할 것.

Fork라는 개념을 os에서 배웠을 때는 직접 써보는 과정이 없어서 이해도 잘 안되었었는데, 직접 사용해보니 fork라는 개념이 쉽게 이해 되었다.

시간이 부족할 거 같아, 무작정 시작을 해서 이번에 인풋을 읽는 부분에서 이상하게 되었다. Thread를 이용해 구현하였다면 read key를 읽을 thread는 사용자의 입력이 없을 동안에은 block되어 더 효율적으로 수행되었을 것임

Main도 timer에서 사용하는 SIGALRM때문에 wait하는 동안에 wait에서 깨어나, 프로세스가 의도와 다르게 종료되었음. 이를 while문을 사용해, sigalrm을 처리하여도 계속해서 프로세스가 죽지않도록, 강제로 수행되게끔 하였는데, 다만 이는 불필요한 cpu연산을 요함.그래서 block될 수 있는 방법을 찾지 못한 부분이 많이 아쉬움.