**Embedded System Software**

과제1 수행 결과 보고서

과목명: [CSE4116] 임베디드시스템 소프트웨어

담당교수: 서강대학교 컴퓨터공학과 박성용

학번 및 이름 : 20171696 채벼리

개발기간: 2020.04.13 ~ 2020.04.26

최종보고서

1. 개발 목표

* mmap() 함수를 사용하거나 Device Driver를 사용하여 Device Control을 수행하는 방식에 대해 이해한다.
* IPC를 사용한 프로세스간의 통신 방법을 이해한다.
* Clock, Counter, Text editor, Draw board를 주어진 조건에 맞게 구현한다.

1. 개발 범위 및 내용

* Back 버튼 : 프로그램 종료
* VOL+/VOL-버튼 : 기능 모드 변경 Clock -> Counter -> Text editor -> Draw board
* Clock 기능

1. FND : 보드의 시간을 출력한다. 앞 두자리는 시간(00~24시) 뒤 두자리는 분(00~60분)
2. LED : 보드 시간 변경 중에는 (3), (4)번 LED 불이 1초씩 번갈아 가면서 들어오고, 변경 완료 중에는 (1)번 LED에만 불이 들어온다.
3. 스위치 버튼

* SW(1) : 보드의 시간을 변경할 수 있도록 하고, 변경된 시간으로 저장하는 버튼
* SW(2) : 시간 변경 시 보드의 시간으로 reset 시킨다.
* SW(3) : 1시간 증가 버튼
* SW(4) : 1분 증가 버튼
* Counter 기능

1. FND : 현재 숫자를 진수에 맞게 출력한다. 999를 넘어갈 경우, 천단위를 생략한다.
2. LED : 10진수는 (2)번, 8진수는 (3)번, 4진수는 (4)번, 2진수는 (1)번 LED가 켜진다.
3. 스위치 버튼

* SW(1) : 10진수 -> 8진수 -> 4진수 -> 2진수 순으로 진수 변환
* SW(2) : 백의 자리 숫자 + 1
* SW(3) : 십의 자리 숫자 + 1
* SW(4) : 일의 자리 숫자 + 1
* Text editor 기능

1. FND : 현재 모드에서 스위치를 누른 횟수를 출력한다. 9999가 넘어갈 경우, 만단위를 생략한다.
2. 스위치 버튼

* 알파벳 또는 숫자 입력을 받는다.

|  |  |  |
| --- | --- | --- |
| (1)  .QZ | (2)  ABC | (3)  DEF |
| (4)  GHI | (5)  JKL | (6)  MNO |
| (7)  PRS | (8)  TUV | (9)  WXY |

* (2)(3)번 동시에 누를 경우 : text LCD에 있던 값을 없애고 빈 상태로 만든다.
* (5)(6)번 동시에 누를 경우 : 영어 <-> 숫자로 입력을 바꾼다.
* (8)(9)번 동시에 누를 경우 : 한 칸 띄어쓴다.

1. Text LCD : text값을 출력한다. 최대 출력 범위는 8이다.
2. Dot Matrix : 현재 입력이 알파벳이면 A, 숫자면 1을 출력한다.

* Draw board 기능

1. FND : 현재 모드에서 스위치를 누른 횟수를 출력한다. 9999가 넘어갈 경우, 만단위를 생략한다.
2. 스위치 버튼

* SW(4), (6), (2). (8) : 방향키로 커서를 각각 좌우상하로 이동한다.
* SW(1) : 현재 그리고 있는 그림을 지우고 초기 상태로 리셋한다.
* SW(3) : 커서 표시 <-> 커서 숨기기
* SW(5) : 현재 위치 선택
* SW(7) : 현재 그리고 있는 그림을 지우나 커서 상태가 리셋되지는 않는다.
* SW(9) : 현재 그림을 반전시킨다.

1. Dot Matrix : 그림을 출력한다.
2. 추진 일정 및 개발 방법
3. 추진 일정

2020/04/18 : input\_process, output\_process, main 함수 구축, 모드 변경 가능하도록, FND출력 가능하도록 기능 구현한다.

2020/04/19 : receive\_msg 함수 구축, 스위치 입력 처리, 기본 CLOCK\_MODE, COUNTER\_MODE, TEXT\_MODE 기능 구현한다.

2020/04/20 : receive\_msg 함수 수정, 기능 구체화

2020/04/21 : DOT matrix, text LCD 출력 기능 추가, 기능별 파일 분리

2020/04/22 : 2개의 스위치 동시 입력 시 TEXT\_MODE 기능 추가

2020/04/24 : DRAW\_MODE 구현 시작

2020/04/25 : 커서 깜빡이기 기능 추가, LED 출력 기능 추가

2020/04/28 : shared memory 방법을 통한 프로세스간 통신 방법 구축

2020/04/29~30 : 세부 조건 수정, 완성, 주석 추가, 보고서 작성

1. 개발 방법

Host PC : Linux(Ubuntu 16.04) System on x64 architecture

Target Board : Embedded linux system on ARM architecture

1. 프로세스 간의 통신 : shared memory를 사용하여 IPC를 수행한다.
2. Device Control 문제 : Device driver 또는 mmap방법을 사용한다.
3. 연구 결과

실행 방법 : make를 통해 생성된 app.out파일을 실행한다.

* + 1. **Inter-Process Communication에 대한 이해**

IPC(Inter-Process Communication)은 프로그래머가 동시에 수행된 개별 프로그램을 다룰 수 있도록 해주는 인터페이스 집합이다. IPC 인터페이스는 프로세스간 통신을 가능케 해준다.

메모리공간은 메모리를 요청한 프로세스만 접근 가능하도록 되어있다. 하지만 공유메모리 영역을 생성하면 커널에 의해 공유메모리가 관리되고, 공유 메모리공간을 삭제하지 않는 한 공유메모리는 프로세스가 없어져도 지속적으로 유지된다.

* shmget

공유메모리는 int shmid = shmget(key\_ t key, size\_t size, int permflags); 호출로 생성된다. Key는 영역을 식별하기 위한 키 값이고 size는 메모리 영역에 필요한 크기(바이트), permflags는 메모리 영역에 대한 허가를 나타낸다.

* shmat

shmget()을 통하여 공유메모리 영역을 생성하면, 식별자 shmid를 갖게된다. 이 식별자를 사용하여 공유메모리를 접근 가능한 유효한 주소를 shmat()을 통하여 얻는다.

char\* shmaddr = (char\*)shmat(int shmid, const void\* daddr, int shmflags);

shmid에 shmget을 이용하여 리턴된 식별자 번호를 명시하고, shmflags를 통해 읽기/쓰기 가능 여부를 명시할 수 있다.

* 공유메모리 제거

Shmdt를 통해 프로세스가 공유메모리를 더 이상 사용할 필요가 없을 경우 프로세스와 공유메모리를 분리시킨다. 그리고 shmctl을 사용하여 공유메모리를 제거할 수 있는데, shmctl은 공유메모리 제거뿐만 아니라 권한을 변경하거나 잠금하는 등 공유메모리 제어에도 사용할 수 있다.

**ii) Device Control 하는 두 가지 방법**

* Mmap

메모리 공간에 직접 매핑하여 해당 공간에 데이터를 써서 device를 직접 제어한다.

* Device driver

Device file을 사용하여 device를 제어한다. Device driver에 명령을 내리면 driver가 명령을 해석해서 device를 제어한다. 따라서 세부적은 mechanism을 몰라도 명령을 내릴 수 있다.

**iii) 네 가지 기능에 대한 구현**

* 파일

header.h : 사용 라이브러리 명시, 매크로 정의, 함수 선언, 전역 변수 선언

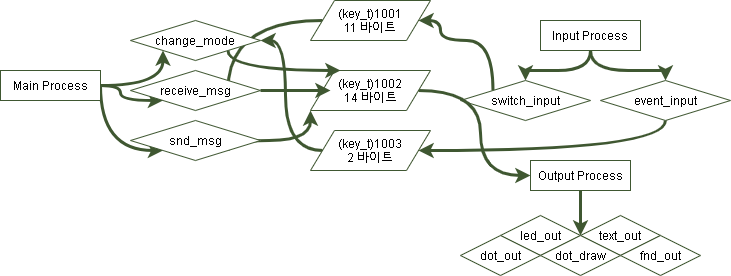
main.c : 받은 input을 처리하여 output process에 메시지를 전달한다.

intput.c : 스위치 또는 이벤트 입력을 받고 해당 정보를 main process에 전달한다.

output.c : 받은 메시지에 따라 알맞은 기기에 값을Device control을 사용하여 출력한다.

* Flow Chart

함수 호출 순서와 공유 메모리 접근에 대한 간략한 Flow Chart를 그렸다.



* 공유메모리 구조

1. int mid1 = shmget((key\_t)1001, 11, IPC\_CREAT|0666);

(key\_t) 1001 : 11바이트의 공간

shmaddr[0] type : '\*' 스위치 입력 기다리는 중, SWITCH 스위치 입력 후

shmaddr[1] n : SWITCH가 동시에 눌려지는 개수 (1일경우 하나의 스위치가, 2일 경우 두개의 스위치가 눌려짐을 뜻한다.)

shmaddr[2] ~ shmaddr[10] : 9개의 switch에 대한 값, 0은 안눌려짐, 1은 눌려짐을 뜻한다.

1. int mid2 = shmget((key\_t)1002, 14, IPC\_CREAT|0666);

(key\_t) 1002 : 14바이트의 공간

shmaddr[0] type : '\*' OUTPUT 출력끝나고 기다리는 중, '#' 출력 중, 메시지 전달을 원할 경우 매크로값FND, FND\_WITH\_BASE, DOT, LED를 타입별로 설정한다.

shmaddr[1] shmaddr[2] : 최대 4자리 숫자를 shmaddr[1]에 하위 2자리, shmaddr[2]에 상위 2자리 나누어 담는다.

shmaddr[3] ~ shmaddr[12] : text lcd 출력을 위한 정보 또는 dot matrix 출력을 위한 정보를 담고 있다.

shmaddr[13] base : type이 FND\_WITH\_BASE일 경우 base정보를 담고 있다.

1. int mid3 = shmget((key\_t)1003, 2, IPC\_CREAT|0666);

(key\_t) 1003 : 2바이트의 공간

shmaddr[0] type : '\*' 이벤트 입력 기다리는 중, EVENT 이벤트 입력 후

shmaddr[1] : 이벤트 버튼에 대한 정보를 담고 있다.

* 함수에 대한 설명

receive\_msg()

스위치 입력을 받으면 input process에서 받은 메시지를 처리하는 함수이다. 현재의 모드(CLOCK\_MODE, COUNTER\_MODE, TEXT\_MODE, DRAW\_MODE)에 따라서 알맞은 처리를 하고 출력을 하기 위해 output process에 메시지를 전달한다.

change\_mode()

이벤트 버튼이 눌려진 경우 input process에서 받은 메시지를 처리하는 함수이다. 현재의 모드를 바꾸고 각 모드에 따라 보드를 초기 상태로 바꾸어준다. 보드의 상태를 바꾸기 위하여 output process에 메시지를 전달한다.

snd\_msg()

커서가 보일 경우(isCursor == 1), CLOCK\_MODE 수정 중(flag == 1)일 경우 1초마다 출력이 바뀐다. 커서가 보일 경우 현재 커서의 위치에서 1초마다 깜빡거리도록, flag == 1일 경우 3번 4번 LED에 1초마다 번갈아 불이 들어오도록 1초마다 output process에 메시지를 전달한다.

event\_input()

이벤트 입력을 받으면 main process에 입력 정보를 전달해준다. 버튼이 눌릴 경우 type을 \*에서 EVENT로 변경하고(다시 \*로 바뀔때까지) 처리 완료될 때까지 대기한다.

switch\_input()

스위치 입력을 받으면 main process에 입력 정보를 전달해준다. 스위치가 눌릴 경우 몇 개의 스위치가 눌렸는 지, 몇 번 스위치가 눌렸는 지 확인한다. type을 \*에서 SWITCH로 변경하고 처리 완료될 때까지(다시 \*로 바뀔때까지) 대기한다.

output\_process()

메시지가 전달되면 각 메시지에 따라 알맞은 FND, Dot Matrix, Text LCD, LED 출력을 할 수 있게 한다. FND, Dot Matrix, Text LCD는 Device driver 제어를 통해, LED 출력은 mmap을 통하여 가능케했다. 각 출력은 fnd\_out, dot\_out, dot\_draw, text\_out, led\_out 함수를 사용한다.

1. 기타

이론에서 배웠던 것을 실제로 실습하면서 프로그램을 작성하니, 어려웠던 개념을 정확히 이해할 수 있었다.