|  |  |
| --- | --- |
| 년도-학기 | 2021년 1학기 |
| 과목명 | 임베디드시스템설계 |

|  |  |
| --- | --- |
| **LAB번호** | **제목** |
| 3 | Lab3: Device Drivers |

|  |  |
| --- | --- |
| 실험 일자 | 2021년 3월 19일 |
| 제출자 이름 | 강\*\* |
| 제출자 학번 | 201803\*\*\*\* |
| 팀원 이름 |  |
|  |  |

**Chapter 1. 프로그램의 동작 방식 설명**

본 과제는 두 가지 방법으로 구현되었다. 방법(1)은 Thread를 사용하지 않고 하나의 프로그램으로 구현한 방법, 방법(2)는 2개의 thread를 사용한 방법을 의미한다. 방법(1)과 방법(2)로 구분해 설명한다.

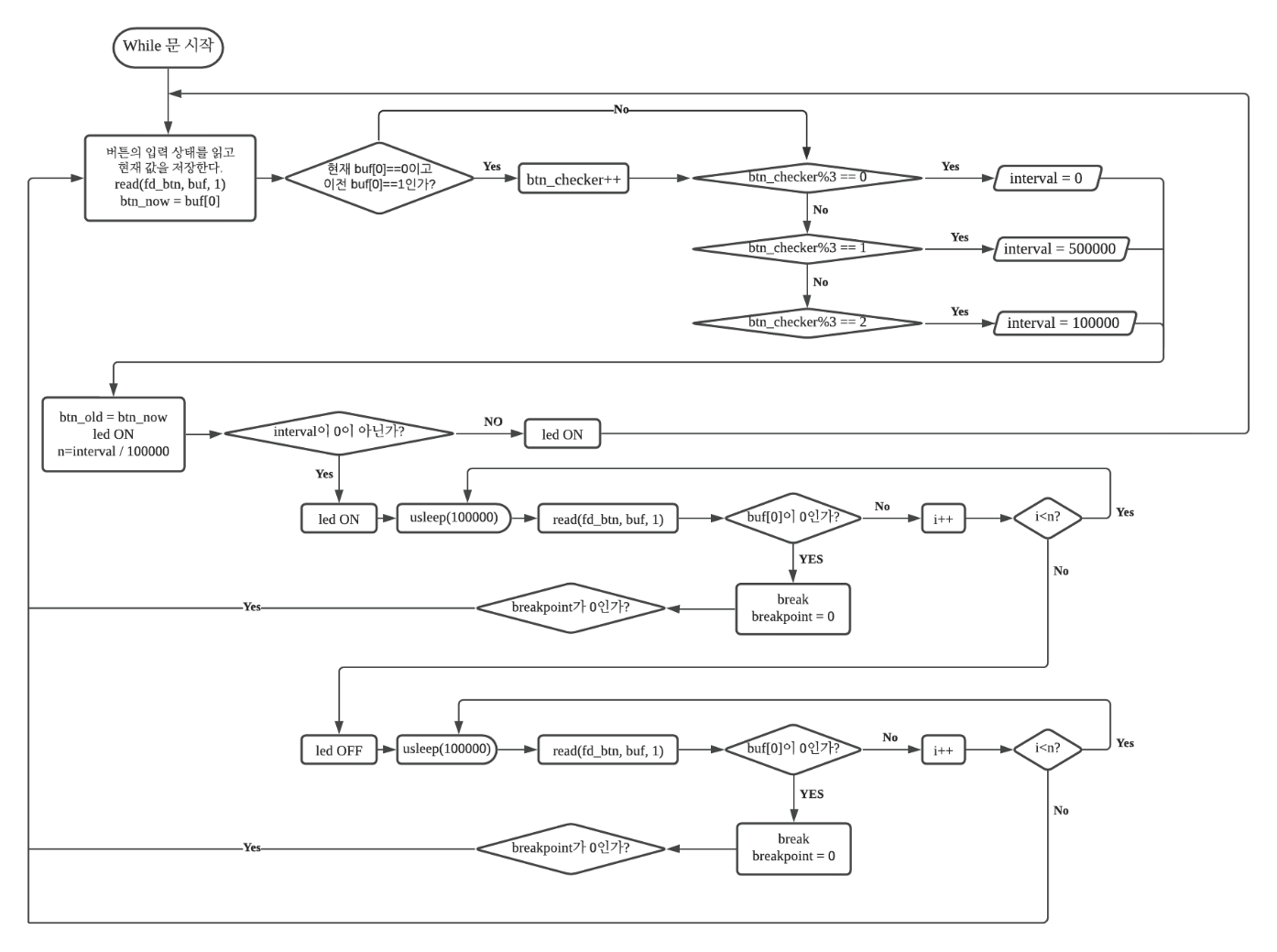
**(1) Thread를 사용하지 않고 하나의 프로그램으로 구현**

우선 main 함수에서 led와 button driver를 open하고 문제가 있으면 에러임을 출력하며 종료한다. While(1)으로 무한 반복문을 생성하고, read(fd\_btn, buf, 1)을 통해 현재 버튼 상태를 읽는다. 읽은 buf[0] 값을 btn\_now 변수에 저장한다.

만약, btn\_now = 0이고 btn\_old(초기값은 1로 설정한다.) = 1이면 버튼이 입력되었음을 의미하며, btn\_checker를 1 증가시킨다. btn\_checker를 3으로 나눈 나머지를 구해 led의 3가지 동작 모드에 해당하도록 경우를 나눈다. 나머지가 0이면 interval = 0, 나머지가 1이면 interval = 500000, 나머지가 2이면 interval = 100000으로 둔다. (Interval은 led의 깜박거림을 위한 usleep 함수에 들어갈 시간이다.)

만약 interval 값이 0이 아니면, led를 켜고 난 뒤 usleep(100000)을 for 문을 통해 n(= interval/100000)번 반복한다. usleep(100000)을 한 이유는 앞선 driver 예제에서 button 상태를 read 할 때 usleep(100000) 간격으로 확인했음을 이용했기 때문이다. 반복문 안에서 read(fd\_btn, buf, 1)을 통해 버튼 입력의 유무를 확인한다. 버튼이 눌렸다면 buf[0] = 0이 된다. 이 때, breakpoint = 0으로 설정하며 break를 통해 usleep을 반복하는 for문을 탈출한다.

이 반복문 뒤에는 led를 끄고 난 뒤 usleep(100000)을 n번 반복하는 같은 구조의 for문이 또 존재하므로, 두 for문 사이에 breakpoint = 0일 때 continue 하는 조건문을 추가한다. 그러면 while문의 처음으로 돌아가 led가 켜지고 꺼지는 그 사이에서도 버튼 입력을 감지하여 모드를 변경할 수 있게 된다. 버튼 입력이 없었다면 led가 켜지고 n번의 usleep(100000)반복, led가 꺼지고 n번의 usleep(100000)이 반복된 후 다시 처음으로 돌아간다. 그리고 interval 값이 0일 때는 led가 켜지고, 처음으로 돌아간다. 이 과정을 무한 반복한다. 아래는 동작을 flow chart로 표현한 것이다. While문 안에서의 상황을 보여준다.

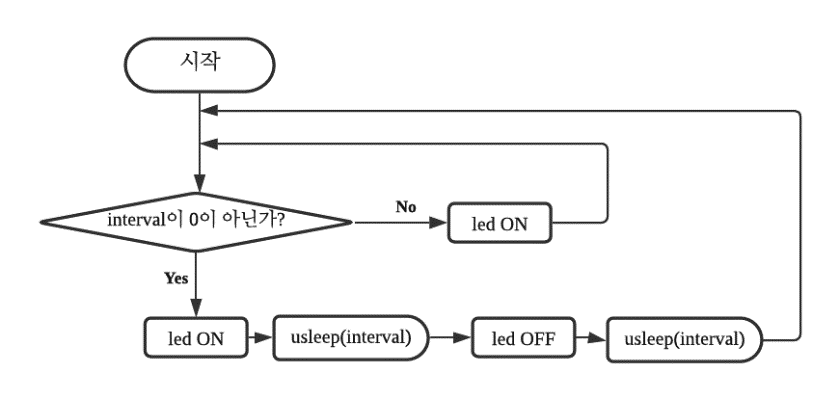


**2) 두 개의 Thread를 사용하여 구현**

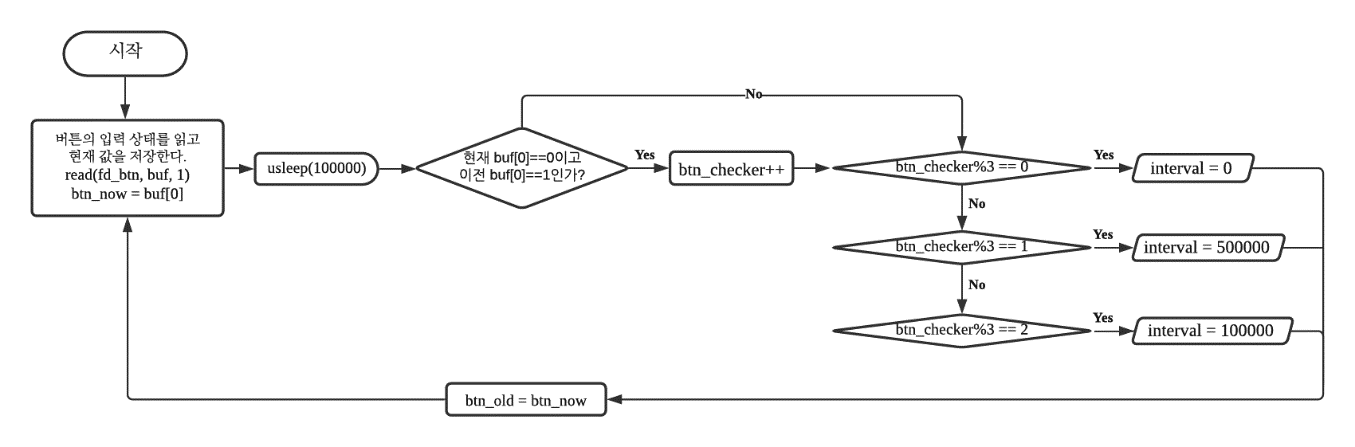
우선 main 함수에서 led와 button driver를 open하고 문제가 있으면 에러임을 출력하며 종료한다.

Thread를 생성하기 위해 먼저 led의 3가지 모드에 대한(interval에 대한) 동작을 구현한 ledmode 함수와 버튼 입력을 감지하고 그 횟수에 따라 led 점멸 간격(interval)을 정하는 button 함수를 만들었다.

ledmode 함수는 무한 루프로 button 함수에서 구한 interval을 이용해 interval이 0이면 led를 켜고, interval이 0이 아니면 led를 켠 뒤 usleep(interval)만큼 멈추고, led를 끈 뒤 usleep(interval)만큼 멈춰 깜박거림을 보인다. 이를 무한 반복해 계속 깜박거리거나 계속 켜질 수 있도록 한다. 아래는 ledmode함수의 flow chart이다.



button함수 또한 무한 루프로, 버튼의 상태를 읽고 그 값(buf[0])을 btn\_now 변수에 저장해 현재 상태를 저장한다. 이전 상태를 의미하는 btn\_old 값이 1이고 btn\_now값이 0이면 버튼을 누른 것이므로 이 때 btn\_checker를 1 증가시킨다. btn\_checker를 3으로 나눈 나머지를 구하고 나머지가 0이면 interval = 0, 나머지가 1이면 interval = 500000, 나머지가 2이면 interval = 100000으로 둔다.



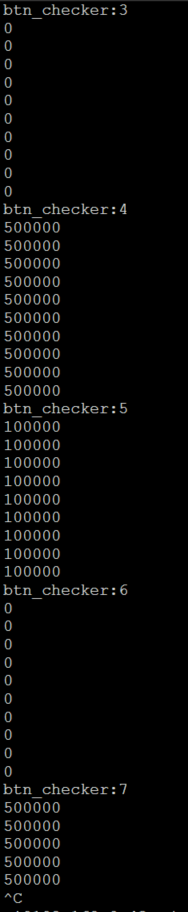
다음 pthread\_create로 ledmode 함수를 사용하는 led thread와 button 함수를 사용하는 btn thread를 생성했다.

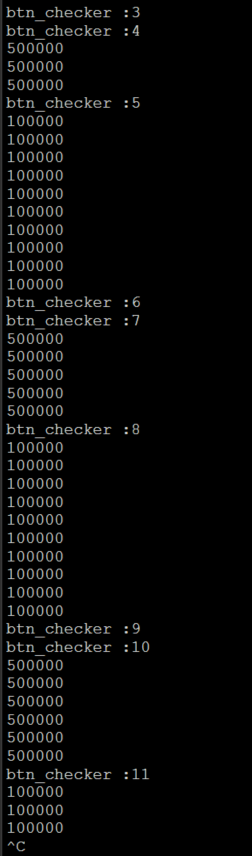
int \*p = &interval;

pthread\_create(&btn,NULL,button,NULL);

pthread\_create(&led,NULL,ledmode,p);

그리고 pthread\_join 함수를 통해 main 함수가 끝나더라도 두 thread가 끝날 때까지 계속 동작하도록 하였다.

**Chapter 2. 결과**



좌측은 thread 없이 구현한 방법의 사진이고 우측은 thread를 2개 이용한 방법의 사진이다.

사진으로는 led 모드의 변화를 담을 수 없기 때문에 버튼 입력을 감지하고 이를 카운트하는 변수와 모드가 변경되었을 때의 led 깜박거림 간격을 출력해 실습이 성공적임을 보였다. 버튼이 입력되면 btn\_checker가 올라가고, led 모드가 변경을 위해 바뀐 interval 값이 출력되는 것을 확인할 수 있다.

Led 모드는 총 3가지로 첫번째로 계속 켜져 있는 모드는 interval이 0인 상태, 3으로 나눴을 때 나머지가 0인 경우로 설정하였다. 두번째로 1초마다 깜박이는 모드는 interval이 500000인 상태, 3으로 나눴을 때 나머지가 1인 경우로 설정하였다. 마지막으로, 0.2초마다 깜박이는 모드는 interval이 100000인 상태, 3으로 나눴을 때 나머지가 2인 경우로 설정했다.

그림을 보면, btn\_checker와 각 경우에 맞는 interval 값이 출력됨을 볼 수 있다. 나머지가 0인 경우에는 매우 빨리 반복하기 때문에 한눈에 결과를 보기 어려워 interval 출력은 생략하였다.

**Chapter 3. 결론 및 Discussion**

일단, 두 가지 방법 모두 정상적으로 작동되었다.

thread 없이 구현한 첫 번째 시도는 led를 켜거나 끄고 sleep(usleep)함수를 실행하는 동안에는 버튼 입력을 인식하지 못한다는 단점이 있었다. 그래서 깜박이는 간격이 짧을 때는 문제가 없어 보였지만 조금만 느리게 깜박이면 여러 번 버튼을 눌러야 모드가 변경되었다. 이걸 해결하는 것이 이번 과제에서 가장 어려웠던 점이였고, 교수님께 조언을 구해 해결할 수 있었다. 해결방안은 sleep해야 하는 총 시간을 버튼 입력 상태를 확인하는 간격에 맞춰 여러 번에 나눠 진행하는 것이었다.

앞선 수업에서 버튼 드라이버 예제로 버튼 입력 상태를 usleep(100000) 간격으로 출력하는 코드를 실행했던 것을 확인하였다. 그래서 led 모드마다 주어진 깜박거림 속도(interval)을 100000값으로 나눠 반복해야할 횟수를 구하고 for문을 만들었다. for문 안에는 usleep(100000)을 하고 read(fd\_btn, buf,1)을 통해 버튼 입력 유무를 확인했다. 입력이 되면 반복문을 탈출해 처음으로 돌아가 위의 문제를 해결할 수 있었다.

더 정확하게 문제를 해결하려면 usleep 시간을 더 짧게 하고 반복문을 더 많이 돌려 버튼 입력 상태를 더 자주 확인해보면 될 것 같다는 생각을 했다. 하지만 너무 짧게 한다면, 반복문이 많이 돌아가 실행 속도가 늦어지는 문제가 발생할 것이라고 생각했다. 따라서 버튼 상태를 read하는 속도에 맞추는 것이 적당할 것 같다.

두 번째 방법인 thread를 이용한 방법은 방법(1)을 구현하는 것에 비해 비교적 쉽게 구현할 수 있었다. 전체 코드를 크게 led의 기능과 button의 기능으로 나누어 생각해보았다. Interval에 맞게 led를 동작하는 부분을 함수로 구현하고, button을 입력을 확인해 버튼이 눌린 누적 횟수를 통해 interval을 구하는 부분을 함수로 구현하여 각 함수를 사용하는 thread를 사용했더니 작동이 잘 되었다.

thread의 사용이 입력에 대한 변화를 감지하고 이에 따라 출력에 변화를 줘야할 때 매우 유용한 것 같았다. sleep함수를 여러 번 나눠서 반복해야 할 이유도 사라졌기 때문에 코드의 작성도 매우 간단했다. 또한 많은 조건문과 반복문을 탈출하기 위한 break와 continue 사용도 필요 없어서 프로그램의 더 빠른 실행이 가능할 것 같다.

**부록**

**<방법 1: thread 사용하지 않은 하나의 프로그램>**

#include <fcntl.h>

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/stat.h>

#include <string.h>

int main()

{

int fd\_btn;

int fd\_led;

char buf[30];

char wbuf[30];

int btn\_old = 1;

int btn\_now;

int btn\_checker = 0;

int interval = 0;

int n;

fd\_btn=open("/dev/gpio\_button",O\_RDWR);

fd\_led=open("/dev/gpio\_led",O\_RDWR);

if (fd\_btn < 0) {

printf("Device open error : %s\n","/dev/gpio\_button");

exit(1);

}

if (fd\_led < 0) {

printf("Device open error : %s\n","/dev/gpio\_led");

exit(1);

}

while(1) {

int breakpoint = 1;

read(fd\_btn, buf, 1); //read button status

btn\_now = buf[0];

if( btn\_now == 0 && btn\_old == 1 ) {

btn\_checker++;

}

if (btn\_checker%3 == 0) {

interval = 0;

}

else if (btn\_checker%3 == 1) {

interval = 500000;

}

else if(btn\_checker%3 == 2) {

interval = 100000;

}

btn\_old = btn\_now;

n = interval / 100000;

if(interval!=0){

wbuf[0]=1;

write(fd\_led, wbuf, 1);

for(int i=0; i<n ; i++){

usleep(100000);

read(fd\_btn, buf, 1);

if(buf[0]==0){

breakpoint = 0;

break;

}

}

if(breakpoint == 0)

continue;

wbuf[0]=0;

write(fd\_led,wbuf,1);

for(int i=0;i<n;i++){

usleep(100000);

read(fd\_btn, buf, 1);

if(buf[0]==0){

breakpoint = 0;

break;

}

}

}

else{

wbuf[0]=1;

write(fd\_led,wbuf,1);

}

}

close(fd\_btn);

close(fd\_led);

}

**<방법 2: 2개의 thread를 사용>**

#include <fcntl.h>

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/stat.h>

#include <string.h>

#include <pthread.h>

#include <time.h>

int fd\_btn;

int fd\_led;

char buf[30];

char wbuf[30];

int btn\_old = 1;

int btn\_now;

int btn\_checker = 0;

int interval = 0;

void\* ledmode(void\* data)

{

while(1){

if(interval!=0)

{

wbuf[0]=1;

write(fd\_led,wbuf,1);

usleep(interval);

wbuf[0]=0;

write(fd\_led,wbuf,1);

usleep(interval);

}

else

{

wbuf[0]=1;

write(fd\_led,wbuf,1);

}

}

}

void\* button(void\* data)

{

while(1){

read(fd\_btn, buf, 1); //read button status

btn\_now = buf[0];

if( btn\_now == 0 && btn\_old == 1 ) {

//btn click

btn\_checker++;

}

if (btn\_checker%3 == 0) {

interval = 0;

}

else if (btn\_checker%3 == 1) {

interval = 500000;

}

else if(btn\_checker%3 == 2) {

interval = 100000;

}

btn\_old = btn\_now;

}

}

int main(){

fd\_btn=open("/dev/gpio\_button",O\_RDWR);

fd\_led=open("/dev/gpio\_led",O\_RDWR);

pthread\_t led,btn;

int \*p = &interval;

pthread\_create(&btn,NULL,button,NULL);

pthread\_create(&led,NULL,ledmode,p);

pthread\_join(led,NULL);

pthread\_join(btn,NULL);

}