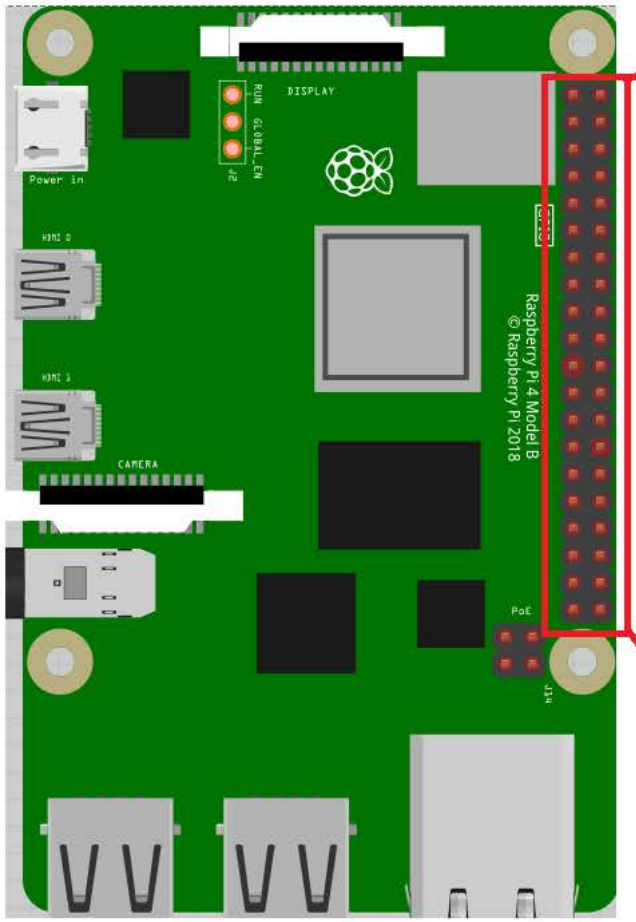
임베디드 보드 실습과 응용 프로그램

Chapter 4.

- Threading, 메시지 큐 통신

최영근
010-5898-3202

GPIO



wPi	BCM	Pin	No	No	Pin	BCM	wPi
		3.3V	1	2	5V		
GPIO 08	GPIO 02	SDA1	3	4	5V		
GPIO 09	GPIO 03	SCL1	5	6	GND		
GPIO 07	GPIO 04		7	8	TXD	GPIO 14	GPIO 15
		GND	9	10	RXD	GPIO 15	GPIO 16
GPIO 00	GPIO 17		11	12		GPIO 18	GPIO 01
GPIO 02	GPIO 27		13	14	GND		
GPIO 03	GPIO 22		15	16		GPIO 23	GPIO 04
		3.3V	17	18		GPIO 24	GPIO 05
GPIO 12	GPIO 10	MOSI	19	20	GND		
GPIO 13	GPIO 09	MISO	21	22		GPIO 25	GPIO 06
GPIO 14	GPIO 11	SCLK	23	24	CE0	GPIO 08	GPIO 10
		GND	25	26	CE1	GPIO 07	GPIO 11
GPIO 30	GPIO 00	SDA0	27	28	SCL0	GPIO 01	GPIO 31
GPIO 21	GPIO 05		29	30	GND		
GPIO 22	GPIO 06		31	32		GPIO 12	GPIO 26
GPIO 23	GPIO 13		33	34	GND		
GPIO 24	GPIO 19		35	36		GPIO 16	GPIO 27
GPIO 25	GPIO 26		37	38		GPIO 20	GPIO 28
		GND	39	40		GPIO 21	GPIO 29

GPIO extension board & GPIO cable

GPIO extension board & GPIO cable

- 브레드보드에 장착하기

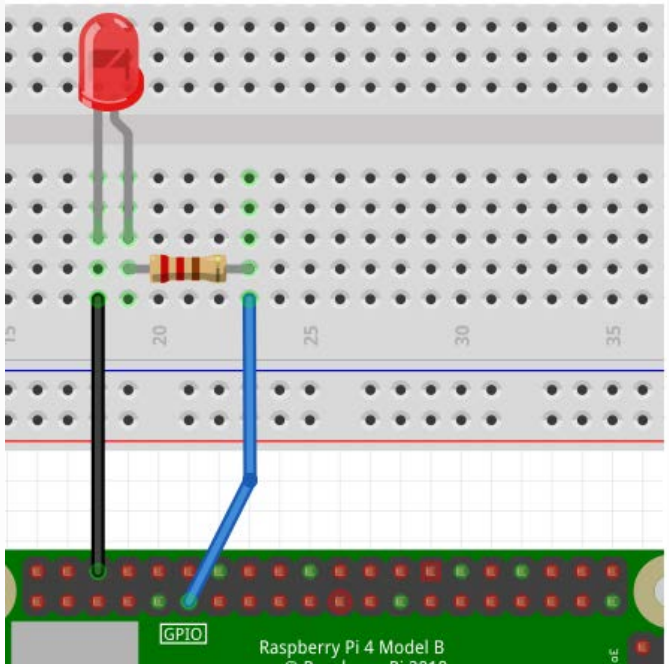
input 함수

- 키보드 입력 받기

```
try:
        while True:
                user_input = input()     # string 입력받음
                print(user_input)        # 입력받은 string을 출력

except KeyboardInterrupt:
        pass
```

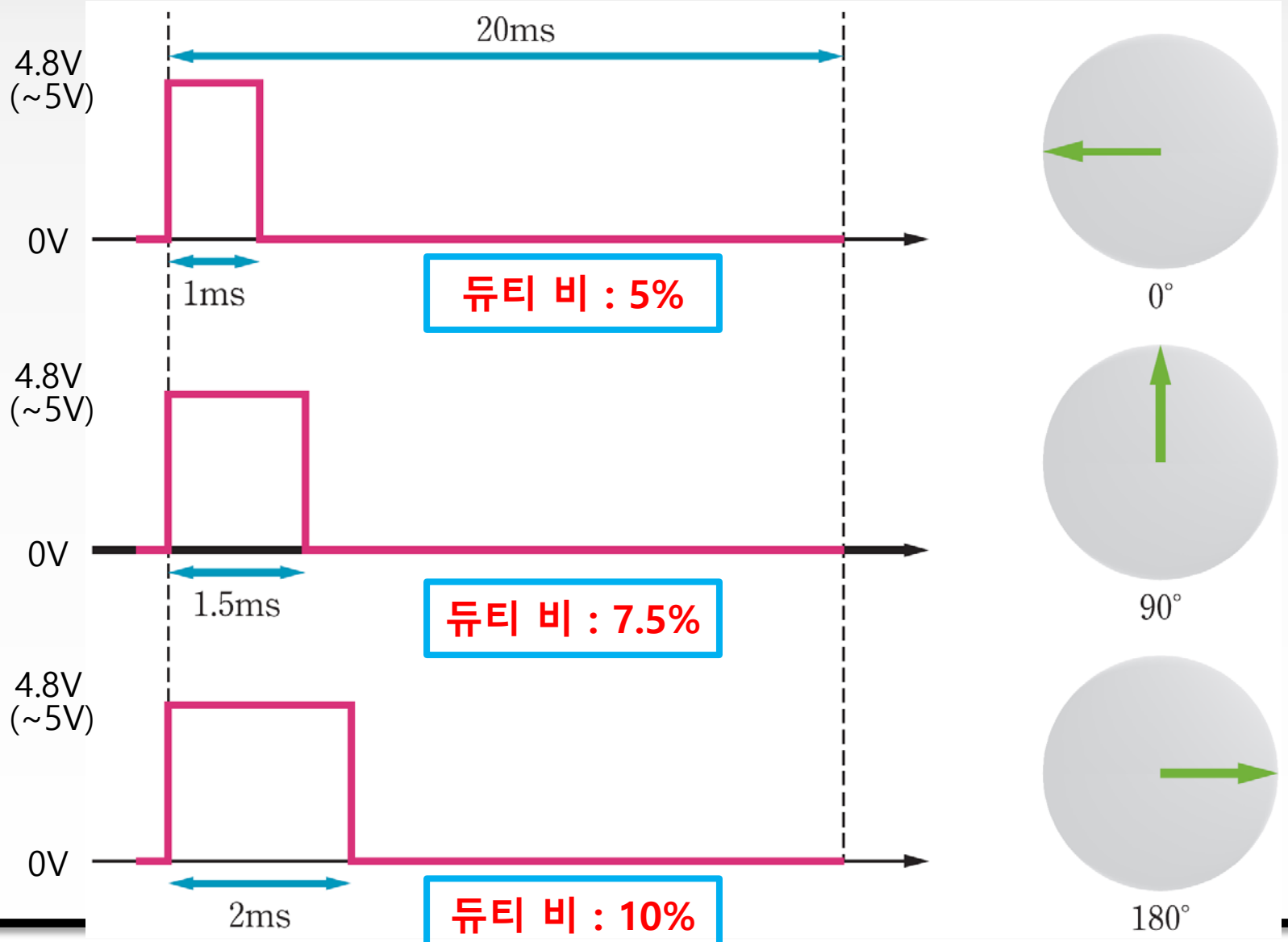
input 함수

- **키보드 입력 받기**
  - GPIO BCM 17번 핀에 연결된 LED 회로를 구성
  - 키보드 'n' 키를 입력하면 LED가 on
  - 키보드 'f' 키를 입력하면 LED가 off



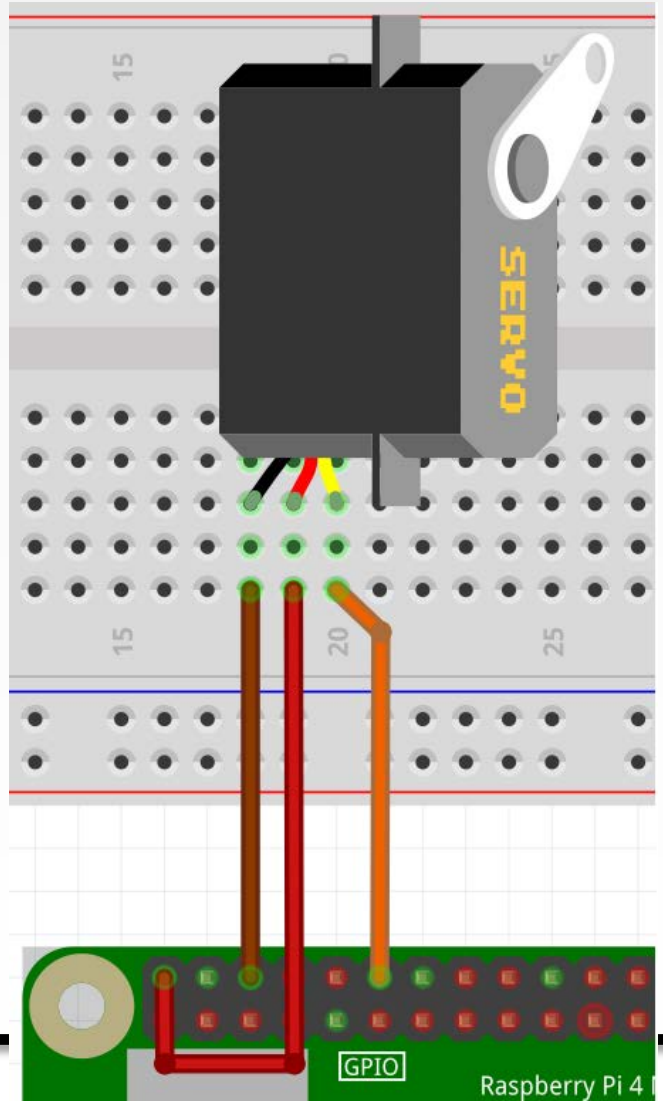

```
import RPi.GPIO as gpio

led_pin =17

gpio.setmode(gpio.BCM)
gpio.setup(led_pin, gpio.OUT)
gpio.output(led_pin, False)

try:
        while True:
                userInput = input()
                print(userInput)
                if userInput == "n":
                        gpio.output(led_pin, True)
                elif userInput == "f":
                        gpio.output(led_pin, False)
except KeyboardInterrupt:
        pass

gpio.cleanup()
```

서보 모터 제어


20ms
4.8V
(~5V)
0V
1ms
듀티 비 : 5%
0°
4.8V
(~5V)
0V
1.5ms
듀티 비 : 7.5%
90°
4.8V
(~5V)
0V
2ms
듀티 비 : 10%
180°

서보 모터 제어

- **SG90 서보 모터의 결선**
  - 3.3V 핀에 연결할 경우에는 라즈베리 파이가 리셋되어 재부팅될 수 있음
  - 간단한 실습을 위해서는 5V 핀에 연결하는 것으로 충분하지만
  - 서보 모터에 부하가 크게 걸릴 경우에는 라즈베리 파이 보드의 5V 핀에 연결하지 말고 외부 전원을 사용 (라즈베리 파이 보드의 영구적인 손상이 발생할 수 있음)



서보 모터 제어

•0도 위치로 회전

- pwmstart(3.0)으로 Run했을 때 서보 모터의 진동이 발생할 경우 3.1이나 3.2로

```
import RPi.GPIO as gpio
import time

servo_pin =18

gpio.setmode(gpio.BCM)

gpio.setup(servo_pin, gpio.OUT)

pwm = gpio.PWM(servo_pin, 50)    # 50Hz
pwm.start(3.0)                   # 0.6ms
# 원칙적으로 서보 모터 제어를 위한 펄스 폭은 1ms(0도) ~ 2ms(180도)
# 저가형 서보 모터인 SG90의 경우 약 0.7ms(0도) ~ 약 2.3ms(180도)
# 20ms X 0.03 = 0.666666ms
# 20ms X 0.12 = 2.4ms

time.sleep(2.0)
pwm.ChangeDutyCycle(0.0)     # 서보 모터 정지

pwm.stop()
gpio.cleanup()
```

서보 모터 제어

- **0도 → 180도로 회전**

- pwmstart(12.0)으로 Run했을 때 서보 모터의 진동이 발생할 경우 11.9이나 11.8로

```
import RPi.GPIO as gpio
import time

servo_pin =18

gpio.setmode(gpio.BCM)

gpio.setup(servo_pin, gpio.OUT)

pwm = gpio.PWM(servo_pin, 50)
pwm.start(3.0)

for cnt in range(0,3):
        pwm.ChangeDutyCycle(3.0)          # 0.6ms == 0도
        time.sleep(1.0)
        pwm.ChangeDutyCycle(12.0)         # 2.4ms == 180도
        time.sleep(1.0)

pwm.ChangeDutyCycle(0.0)

pwm.stop()
gpio.cleanup()
```

서보 모터 제어

- 0도 → 180도로 회전

```
import RPi.GPIO as gpio
import time

servo_pin =18

gpio.setmode(gpio.BCM)

gpio.setup(servo_pin, gpio.OUT)

pwm = gpio.PWM(servo_pin, 50)
pwm.start(3.0)                                  # 초기 상태는 0도

for t_high in range(30,120):
        pwm.ChangeDutyCycle(t_high/10.0) # 듀티 비를 3.0 ~ 12.0까지 0.1씩 증가
        time.sleep(0.02)                        # 0.02sec delay

pwm.ChangeDutyCycle(3.0)                        # 동작이 끝나면 다시 0도로 회전
time.sleep(1.0)
pwm.ChangeDutyCycle(0.0)

pwm.stop()
gpio.cleanup()
```

서보 모터 제어

- 0도 → 180도로 회전한 후 180도 → 0도로 회전

```
import RPi.GPIO as gpio
import time

servo_pin           = 18
SERVO_MAX_DUTY    = 12
SERVO_MIN_DUTY    = 3

gpio.setmode(gpio.BCM)
gpio.setup(servo_pin, gpio.OUT)

servo = gpio.PWM(servo_pin, 50)
servo.start(0)          # 서보 모터는 정지 상태로 시작

def servo_control(degree, delay):
  if degree > 180:    # 180도를 초과하지 못하도록
    degree = 180

  duty = SERVO_MIN_DUTY+(degree*(SERVO_MAX_DUTY-SERVO_MIN_DUTY)/180.0)
    # 각도 값을 듀티 값으로 변환
  print("Degree: {} to {}(Duty)".format(degree, duty))
    # shell 에 각도와 듀티의 값을 출력
  servo.ChangeDutyCycle(duty)
  time.sleep(delay)

for i in range(1, 180, 10):  # 정회전
        servo_control(i, 0.1)

time.sleep(1.0)

for i in range(180, 1, -10): # 역회전
        servo_control(i, 0.1)
except KeyboardInterrupt:
    pass

servo.stop()
gpio.cleanup()
```

다중 입력, 다중 출력 제어

- **키보드 입력에 의해 LED와 서보 모터 회전**
  - 첨부한 Ch04_13.txt 파일에서 다음과 같이 제어되도록 수정할 부분을 찾아 보시오.

  - GPIO BCM 22번 핀에는 버튼을 연결
  - GPIO BCM 17번 핀에는 red LED, 27번 핀에는 green LED, 23번 핀에는 blue LED, GPIO BCM 18번 핀에는 서보 모터를 연결

  - 키보드 'q' 키를 입력하면 서보 모터가 0도 회전, 키보드 'w' 키를 입력하면 서보 모터가 90도 회전, 키보드 'e' 키를 입력하면 서보 모터가 180도 회전

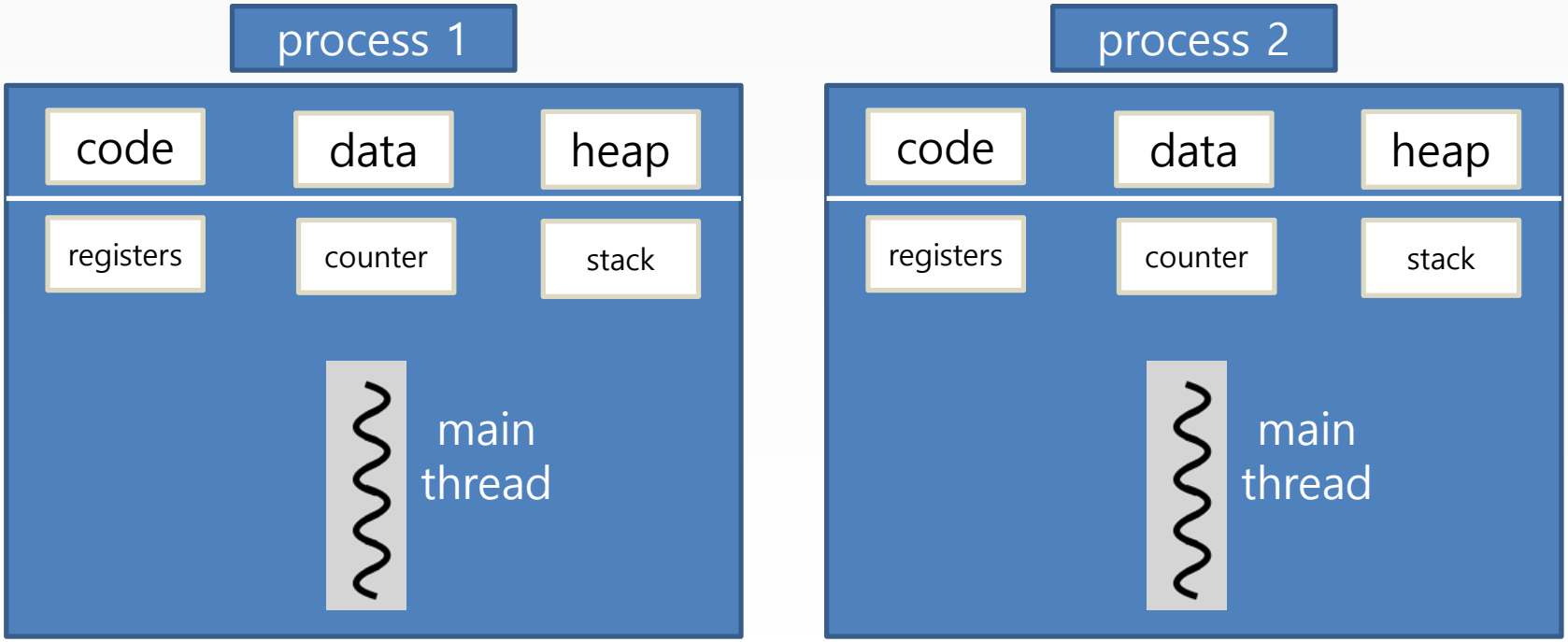
  - 버튼을 첫 번째로 누르면 red LED on
  - 버튼을 두 번째로 누르면 green LED on
  - 버튼을 세 번째로 누르면 blue LED on
  - 버튼을 네 번째로 누르면 모든 LED off

메모리 영역

- **code**
  - 소스 코드 자체. 함수, 제어문, 상수 등
- **data**
  - 전역 변수, 정적 변수, 배열 등
- **stack**
  - 지역 변수, 매개 변수, 리턴 값 등 임시 데이터
- **heap**
  - 필요에 의해 동적으로 메모리를 할당 하고자 할 때 위치하는 메모리 영역
  - 파이썬에서는 Python Memery Manager에 의해 자동 할당
- **register**
  - CPU 내의 임시 저장 장치. 읽어 들인 명령어, 메모리 주소, 메모리와 주고받을 값 등
- **counter**
  - 읽어 들일 명령어의 주소

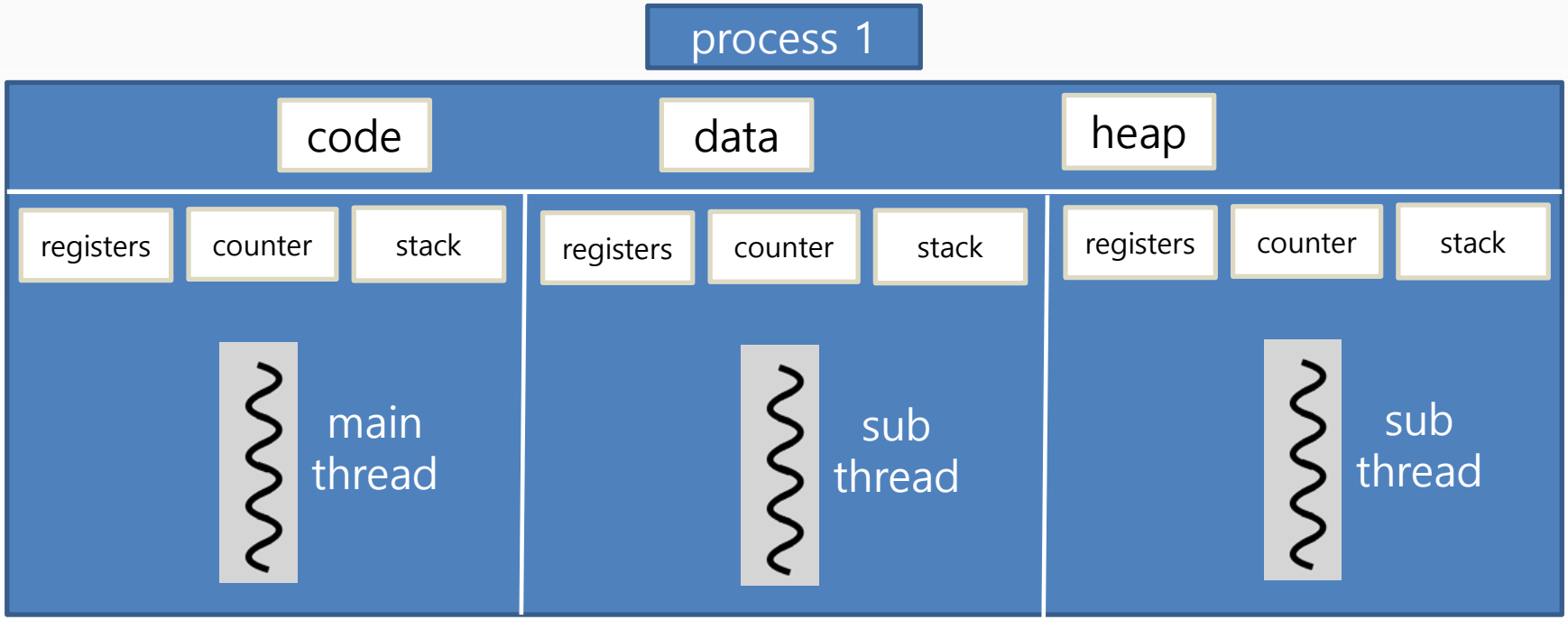
thread

- **thread**
  - 지금까지 작성한 소스 코드는 1개의 process(CPU가 수행하는 작업의 단위) 상에서 수행되었음
  - 1개의 process는 기본적으로 1개의 thread(process 내에서 실행되는 흐름의 단위)를 가짐



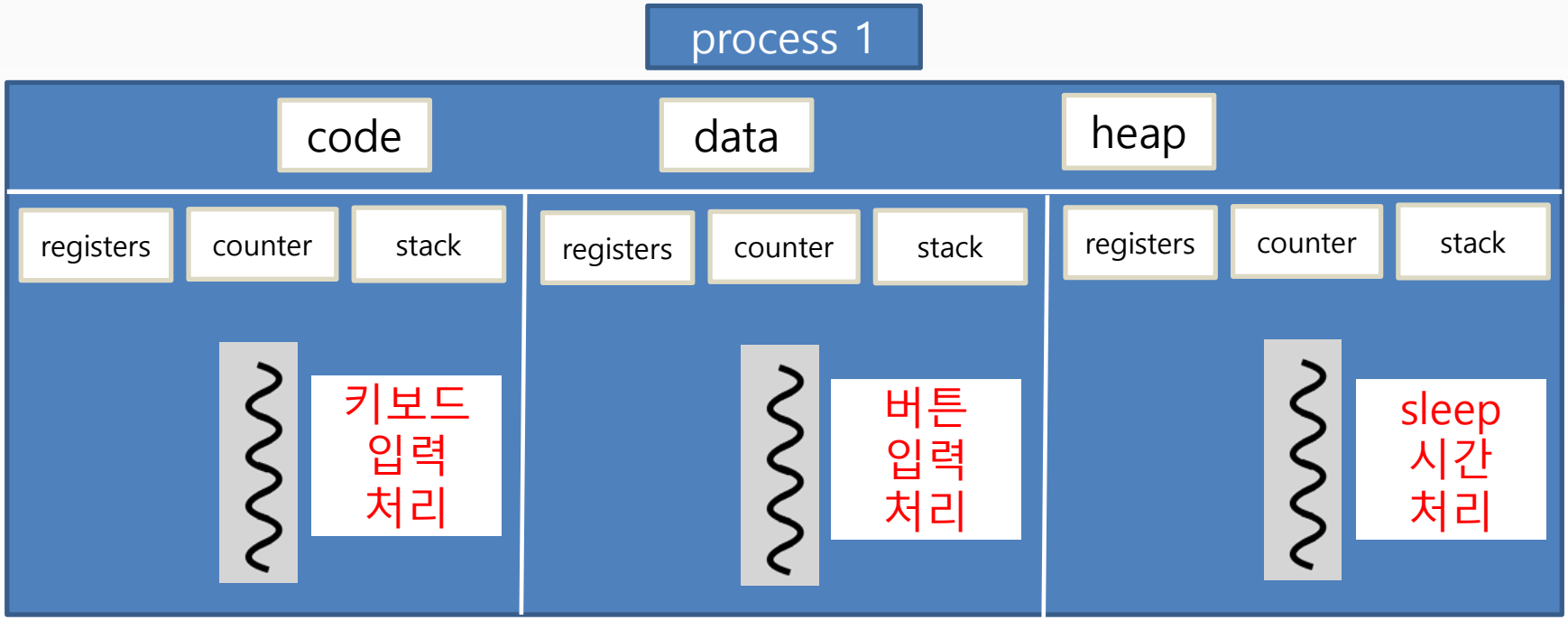
thread

- **multi-thread**
  - 프로세스 내에 여러 개의 thread를 생성할 수 있음
  - 한 process 내의 thread들은 code, data, files를 서로 공유하며 register, counter, stack은 따로 할당 받음



thread

- **multi-thread**
  - 하나의 프로그램에서 여러 가지 입력을 받아 처리해야 할 경우 등에 필요



thread

- thread 생성

```
import threading                        # threading 모듈 import
import time

flag_exit = False                       # flag_exit가 True가 되면 thread 종료
def t1_main():                          # thread가 수행할 함수
        while True:
                print("₩tt1")           # 실행 확인을 위해 "Tab t1" 을 출력
                time.sleep(0.5)
                if flag_exit: break     # flag_exit가 True가 되면 thread 종료

t1 = threading.Thread(target=t1_main) # t10이라는 이름의 thread 생성
t1.start()                              # start 함수에 의해 thread를 실행 가능하도록 함

try:
        while True:                     # main routine. 1sec 간격으로 "main"을 출력
                print("main")
                time.sleep(1.0);

except KeyboardInterrupt:               # 키보드 인터럽트가 발생하면
        pass

flag_exit = True                        # thread 종료
t1.join()                               # thread가 종료되기를 기다림
```

thread

- thread 추가 생성

```
import threading
import time

flag_exit = False
def t1_main():
        while True:
                print("₩tt1")
                time.sleep(0.5)
                if flag_exit: break

def t2_main():                                  # thread를 추가 생성
        while True:
                print("₩t₩tt2")
                time.sleep(0.2)
                if flag_exit: break

t1 = threading.Thread(target=t1_main)
t1.start()

t2 = threading.Thread(target=t2_main)
t2.start()

try:
        while True:                             # 키보드 입력을 출력
                userInput = input()
                print(userInput)

except KeyboardInterrupt:
        pass

flag_exit = True
t1.join()

t2.join()
```

Ch.3 p.8 LED 제어

- **LED flicker**
  - 주파수 1Hz로 점멸

```
import RPi.GPIO as gpio
import time

led_pin =17

gpio.setmode(gpio.BCM)

gpio.setup(led_pin, gpio.OUT)

try:
        while True:
                gpio.output(led_pin, True)
                time.sleep(0.5)                 # LED on 후 0.5sec delay
                gpio.output(led_pin, False)
                time.sleep(0.5)                 # LED off 후 0.5sec delay
except KeyboardInterrupt:
        pass

gpio.cleanup()
```

LED 제어

- thread를 이용한 LED flicker 및 문자열 출력

```
import threading
import time
import RPi.GPIO as gpio

led_pin =17

flag_exit = False
def blink_led():
        while True:
                gpio.output(led_pin, True)
                time.sleep(0.5)
                gpio.output(led_pin, False)
                time.sleep(0.5)

                if flag_exit: break

gpio.setmode(gpio.BCM)
gpio.setup(led_pin, gpio.OUT)

thread_blink = threading.Thread(target=blink_led)
thread_blink.start()

try:
        while True:
                print("main")
                time.sleep(1.0);

except KeyboardInterrupt:
        pass

flag_exit = True
thread_blink.join()
```

다중 주기 작업 처리

- **서로 다른 주기로 점멸하는 3개의 LED 제어**
  - 첨부한 Ch04_22.txt 파일을 테스트해 보시오.

  - GPIO BCM 17번 핀에는 red LED, 27번 핀에는 green LED, 23번 핀에는 blue LED를 연결

  - red LED는 0.7sec on – 0.7sec off를 반복
  - green LED는 1.3sec on – 1.3sec off를 반복
  - blue LED는 1.7sec on – 1.7sec off를 반복

  - 메인 루틴에서는 red LED 제어, thread t1은 green LED 제어, thread t2는 blue LED 제어

Ch.3 p.20 LED 제어

- **RPi.GPIO.PWM 모듈**
  - 0.1초 간격으로 LED 밝기가 0% ~ 100% 로 1%씩 증가 후 1%씩 감소해서 0%가 됨

```
import RPi.GPIO as gpio
import time

led_pin =18

gpio.setmode(gpio.BCM)

gpio.setup(led_pin, gpio.OUT)

pwm = gpio.PWM(led_pin, 1000.0) # 1000.0Hz
pwm.start(0.0) # 0.0% 밝기로 시작

try:
        while True:
                for t_high in range(0,101):
                        pwm.ChangeDutyCycle(t_high)
                        time.sleep(0.1)
                for t_high in range(100,-1,-1):
                        pwm.ChangeDutyCycle(t_high)
                        time.sleep(0.1)
except KeyboardInterrupt:
        pass

pwm.stop()
gpio.cleanup()
```

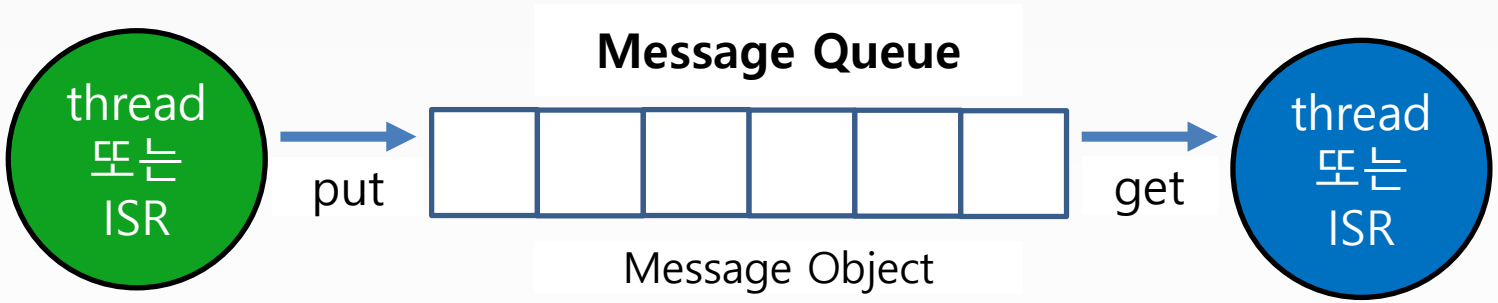
LED 제어

- thread를 이용한 LED 밝기 조절 및 문자열 출력

```
import threading
import time
import RPi.GPIO as gpio

led_pin =18
gpio.setmode(gpio.BCM)
gpio.setup(led_pin, gpio.OUT)
pwm = gpio.PWM(led_pin, 1000.0)
pwm.start(0)

flag_exit = False
def fading_led():
        while True:
                for t_high in range(0,101):
                        pwm.ChangeDutyCycle(t_high)
                        time.sleep(0.01)
                for t_high in range(100,-1,-1):
                        pwm.ChangeDutyCycle(t_high)
                        time.sleep(0.01)

                if flag_exit: break

thread_ledPWM = threading.Thread(target=fading_led)
thread_ledPWM.start()

try:
        while True:
                print("main")
                time.sleep(1.0);

except KeyboardInterrupt:
        pass

flag_exit = True
thread_ledPWM.join()

pwm.stop()
gpio.cleanup()
```

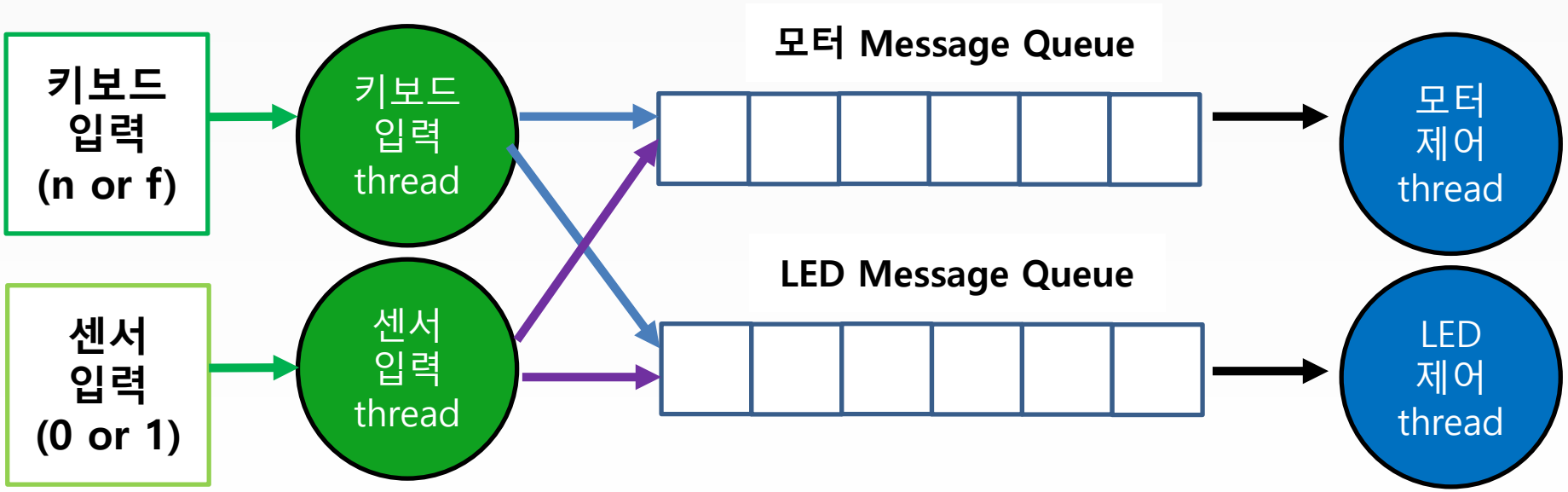
메시지 큐(Message Queue)

- **메시지 큐 통신**
  - thread와 다른 thread 간에, 또는 인터럽트 처리 함수와 thread 간에 데이터를 주고 받을 때 사용



메시지 큐(Message Queue)

- **메시지 큐 통신**
  - 가령, 키보드 입력 n에 의해 DC 모터가 회전하고 LED가 on되며, 키보드 입력 f에 의해 DC 모터가 정지하고 LED가 off. 또한 센서가 on되면 DC 모터가 정지하고 LED가 점멸되는 시스템
  - 하나의 출력에 대해 2개 이상의 thread가 접근 가능할 경우 출력을 담당하는 thread 2개를 추가하면 편리하게 구현 가능



메시지 큐(Message Queue)

- 메인 루틴과 thread 간의 메시지 큐 통신

```
import threading
import queue                                # queue 모듈 import
import time

HOW_MANY_MESSAGES =10                       # 메시지 큐에 저장할 수 있는 메시지의 최대 개수
mq = queue.Queue(HOW_MANY_MESSAGES)         # 메시지 큐 생성. 파라미터는 최대 개수

flag_exit = False
def t1_main():
        value_put = 0

        while True:
                value_put = value_put +1
                mq.put(value_put)           # 메시지 큐에 value의 값을 put
                time.sleep(0.5)

                if flag_exit: break

thread_MQ = threading.Thread(target=t1_main)
thread_MQ.start()

try:
        while True:
                value_get = mq.get()        # 메시지 큐에 저장되어 있는 메시지를 변수 value_get에 저장
                                            # 만약 get할 메시지가 없으면 thread는 메시지를 기다림
                print("Read Data %d" %value_get)

except KeyboardInterrupt:
        pass

flag_exit = True
thread_MQ.join()
```

thread와 message queue를 이용한 다중 입력/다중 출력 제어

- 첨부한 Ch04_28.txt 파일을 테스트해 보고 수정하시오.

- GPIO BCM 22번 핀에는 버튼을 연결
- GPIO BCM 17번 핀에는 red LED, 27번 핀에는 green LED, 23번 핀에는 blue LED, 24번 핀에는 white LED, 18번 핀에는 yellow LED 를 연결
- GPIO BCM 19번 핀에는 서보 모터를 연결

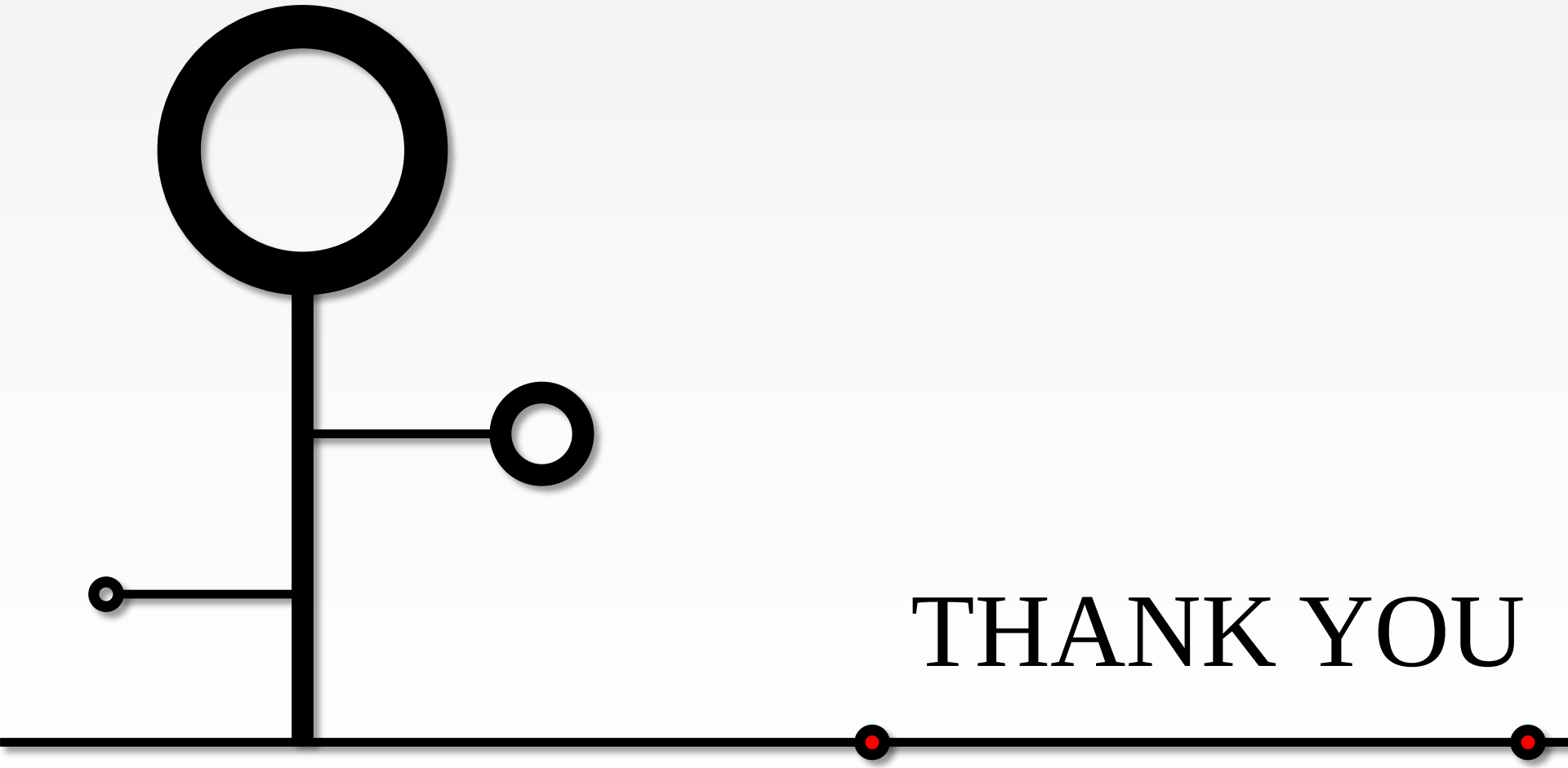
(다음 페이지로)

thread와 message queue를 이용한 다중 입력/다중 출력 제어

- 키보드 'q' 키를 입력하면 서보 모터가 0도 회전, 키보드 'w' 키를 입력하면 서보 모터가 90도 회전, 키보드 'e' 키를 입력하면 서보 모터가 180도 회전

- white LED는 2.4sec 주기로 on – off를 반복
- yellow LED는 2.048sec 주기로 1024단계로 밝아지고 어두워지기를 반복
- 버튼을 첫 번째로 누르면 red LED on
- 버튼을 두 번째로 누르면 green LED on
- 버튼을 세 번째로 누르면 blue LED on
- 버튼을 네 번째로 누르면 모든 LED off

- ISR에서는 red, green, blue, LED off 메시지를 main thread로 전송
- thread t1_main은 white LED 제어
- thread t2_main은 yellow LED 제어
- thread t3_main은 서보 모터 제어



THANK YOU