보고서

20171015 남기석

1. 로컬 이진 카운터

   A. 18,27,22,6번 gpio를 output – led로 설정

   B. 25,12를 up / down 스위치로 설정

   C. 코드에서 처음 시작할 때 led를 모두 0으로 설정

   D. up스위치가 눌렸다면 25번 스위치가 눌린것이고 25번 스위치의 값을 if로 확인하여 현재 값인 count value를 ++1 해준다.

   E. down스위치인 12번이 눌리면 count –

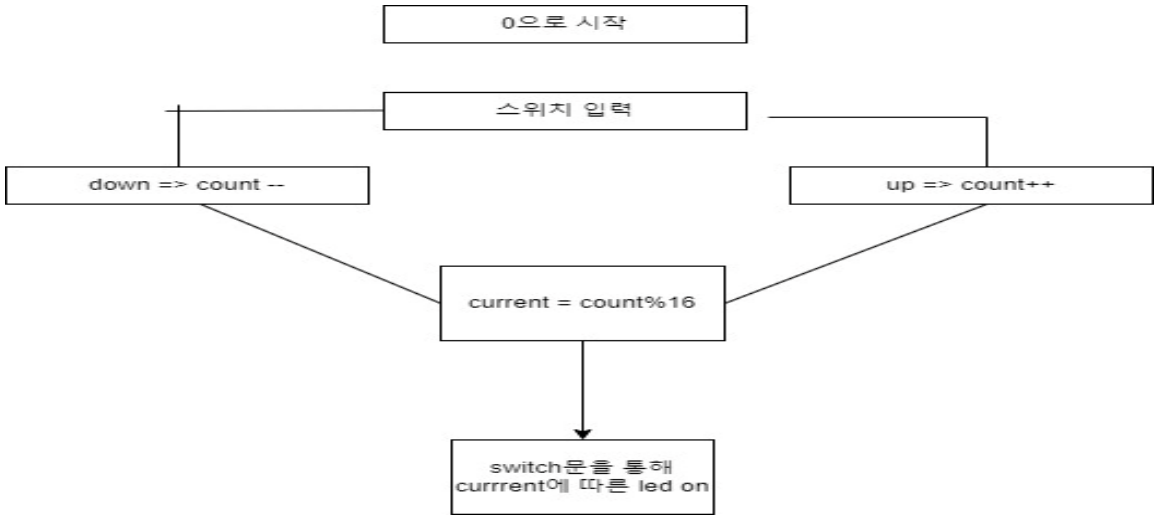
   F. 모듈러 연산을 위해 current = count % 16을 수행한 값

   G. 주의 : current값을 그냥 출력하니 너무 많은 출력이 있어서 이전 값과 다를 때만 출력을 하도록 설정했습니다.(past변수를 새로 만들어서 past!=current 일 때만 출력 의도)

   H. Current의 값에 따라 led가 이진 카운터로 동작하도록 switch문 활용

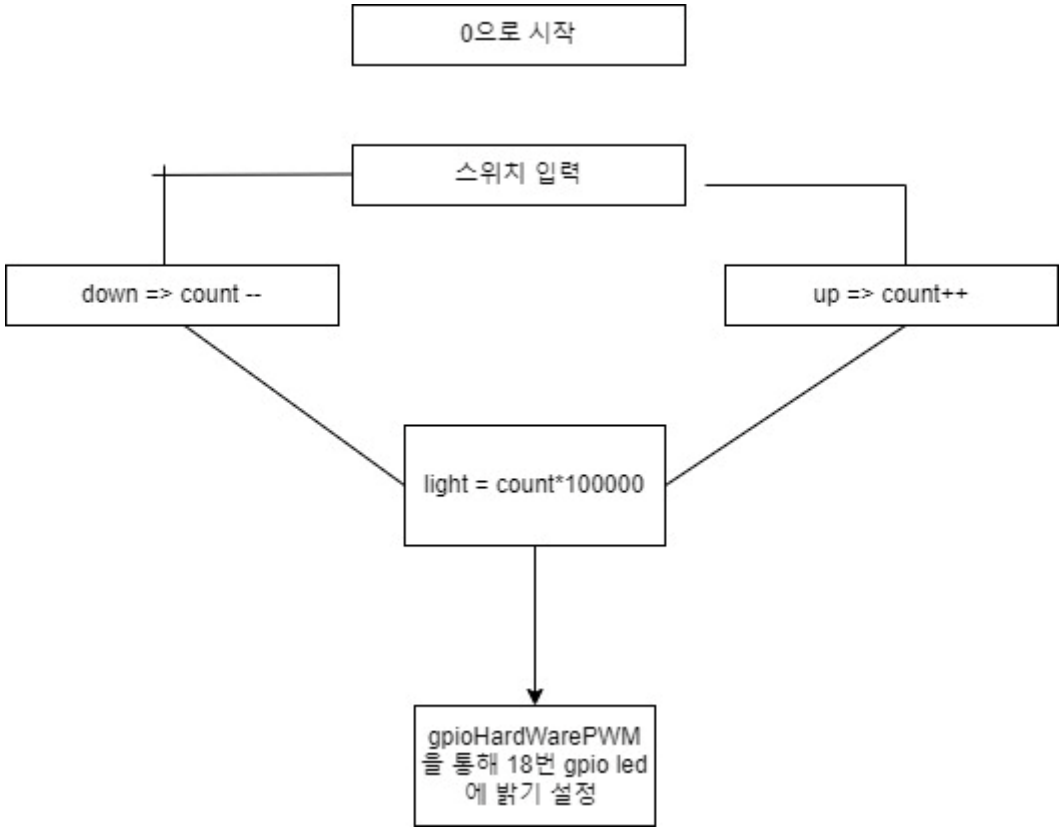
      i. 따라서 16 == 0 / 17 == 1 / 18 == 2 / ....로 동작

      ii. 음수의 경우 모두 0으로 처리된다.



2. 로컬 led 밝기 조절 스위치

- 1번과 동일한 회로를 활용
- 여기서 나머지 led는 제외하고 gpio 18번만 하나의 led를 위해 사용
- 25/12번은 그대로 up/down스위치로 활용
- 1~10단계로 적용
- 25번 스위치가 눌렸고 현재 count가 10보다 작다면 count ++
- 12번 스위치의 경우 count가 1보다 큰 경우에만 count—
  - 1에서 down한다면 아무 동작도 수행하지 않는다.
  - 마찬가지로 현재 카운트가 10이라면 up을 눌러도 아무 동작 수행 x
- 이 후 현재 count를 출력해주고
- Count * 100000으로 light를 설정한다
- gpioHardWarePWM을 통해 18번에 위 light값을 적용하여 밝기 표현



3. 원격 이진 카운터

- 1번 코드와 회로를 활용
- 원격을 수행하기 위해 pi는 server역할, 다른 프로세스는 클라이언트 역할을 수행하도록 설정
- 클라이언트의 경우 수업시간에 다룬 echo_client.c를 활용한(numClnt.c) 숫자를 입력받아 서버에 전달하는 역할을 수행

1 server 소캣 생성

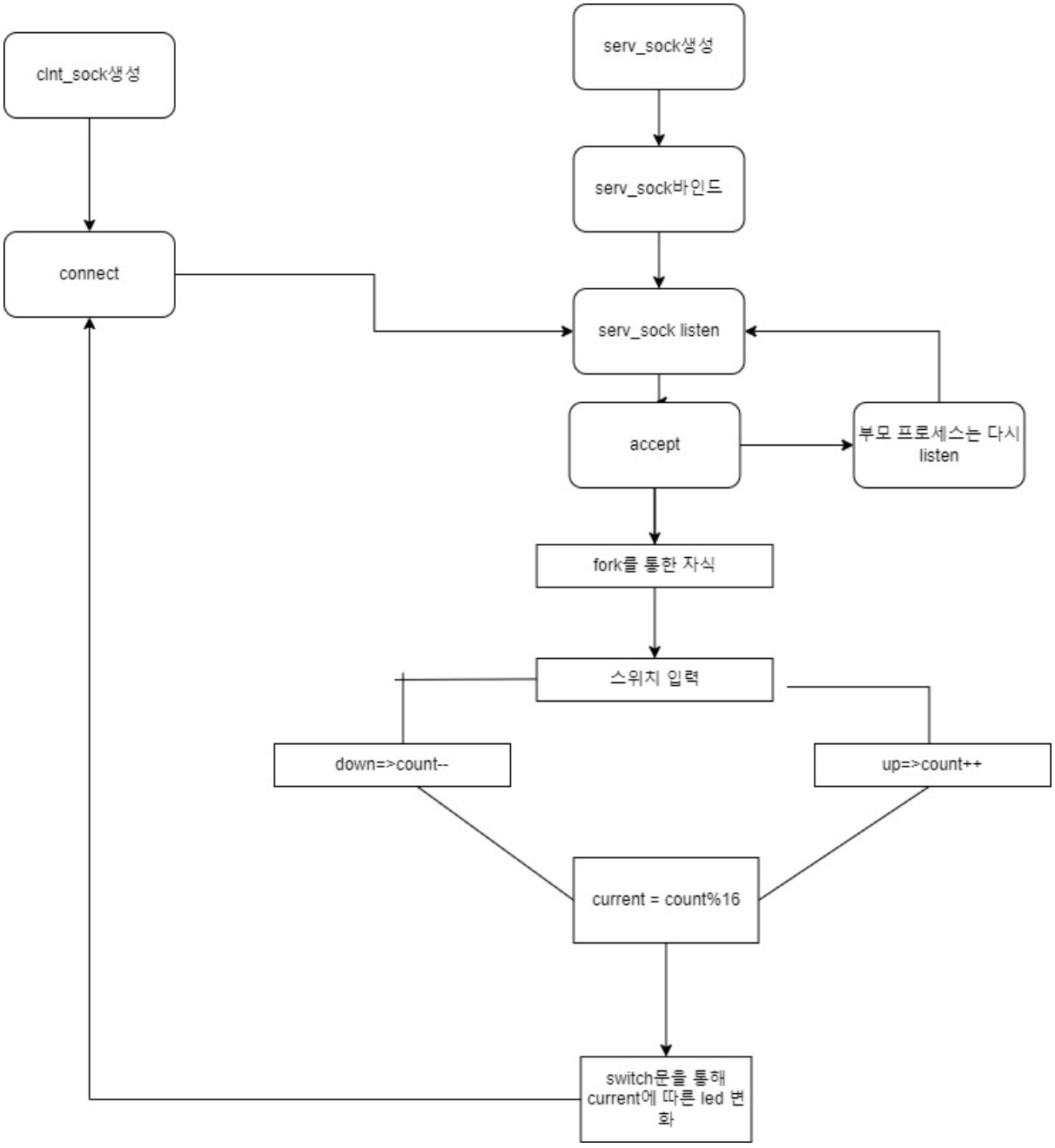
2 server소캣 bind(inaddrAny를 통해 현재 내 컴퓨터 주소를 활용)

3 server소캣 listen상태로

4 멀티 프로세스의 구조를 활용

- 먼저 자식의 죽음은 sigaction을 활용하여 sigchld를 waitpid를 통해 처리하도록 설정
- listen소캣이 클라이언트로부터 요청이오면 accept를 통해 새로운 소캣을 생성
- fork를 통해 자식 프로세스를 생성하고 클라이언트와의 통신은 자식이 담당
- 자식 프로세스는 listen소캣이 필요없으니 닫는다
- 자식 프로세스는 클라이언트로부터 받은 메시지를 message에 read한다
- Atoi를 통해 전달받은 숫자(message)를 int로 변환한다
- 이 후 1번과 마찬가지로 0에서 시작하여 클라이언트에게 받은 숫자만큼 count에 더해주고 모듈러 연산(current = count%16)을 수행한 결과인 current의 값에 따라 switch문을 돌며 동작
- 이진 카운터의 동작은 1번과 동일. 여기서 추가된 것은 원격을 위한 서버와 클라이언트의 통신
- 자식은 위 동작 수행 후 클라이언트에게 받은 숫자를 다시 돌려줘서 클라이언트에서 보낸 메시지가 잘 보내졌는지 확인할 수 있도록 동작

5 부모는 accept소캣이 필요없으니 닫고 다시 listen동작을 수행



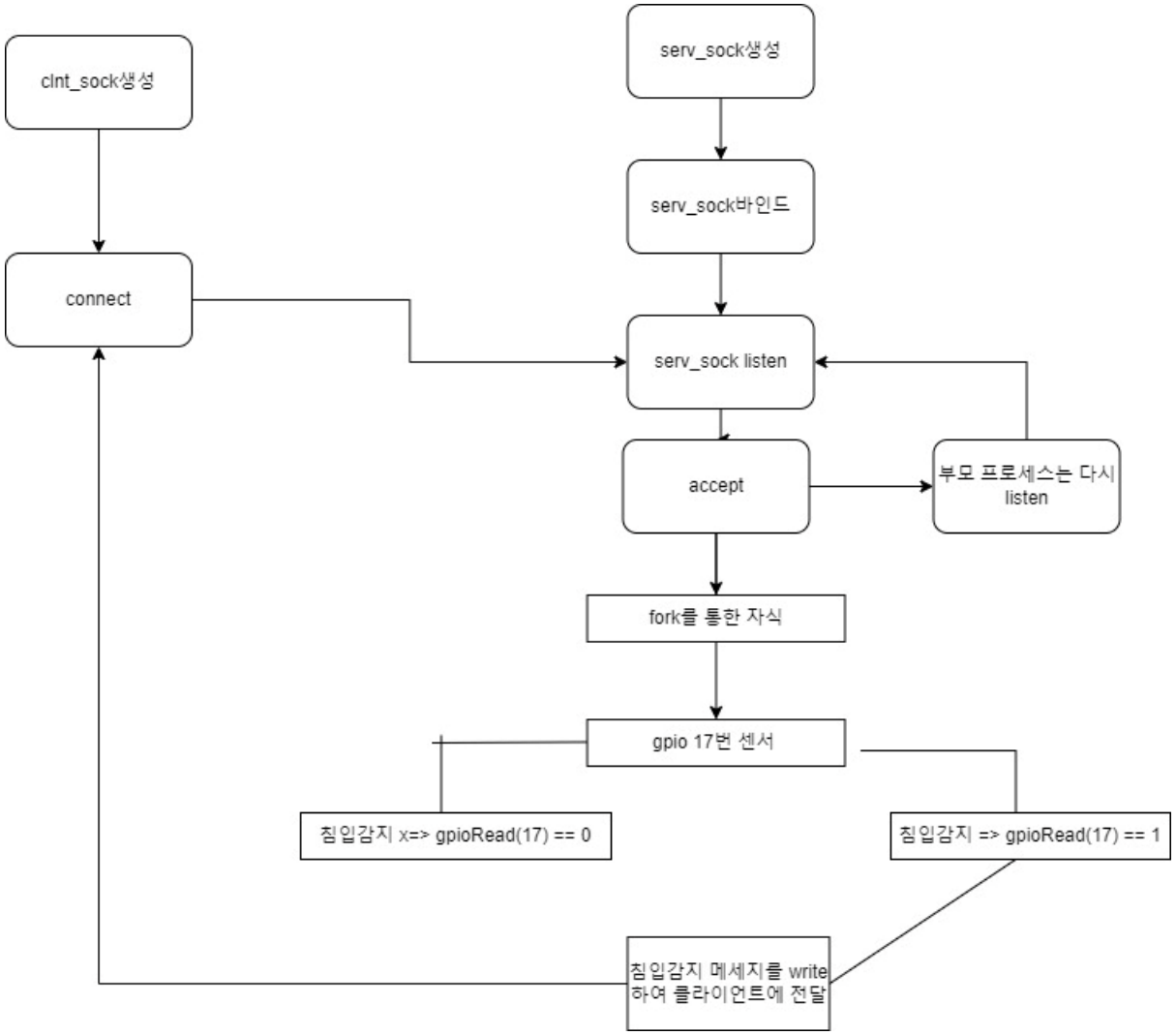
주의 : 클라이언트는 실행 시 ex) ./clnt 서버ip port

주의 : 서버는 실행 시 ex) ./serv 포트

4번 원격 침입 알리미

- 3번과 마찬가지로 server와 client의 소통을 통해 원격을 다룬다.
- 주의 : 서버에서 서버의 ip를 보내주면 0.0.0.0으로 나와 클라이언트에서 서버로부터 침입감지 메시지가 오면 서버 ip + 받아온 message + 현재시간을 출력해주는 식으로 진행했습니다.
- 클라이언트는3번과 마찬가지로 echo_client를 활용한 제출파일인 echo_client.c로 수행
- 3번과 동일하게 serv / clnt 연결 + 멀티프로세스 구조
- 마찬가지로 서버에서 accept 후 fork를 통해 생성된 자식 프로세스에서 클라이언트와 소통
- 3번과 다른점은 라즈베리파이의 동작방식

1. Gpio17번을 센서로 설정
2. 센서를 통해 무언가가 감지되면 gpioRead(17) == 1이고 이 경우 서버에서 accept로 생성한 클라이언트 소캣에 "침입감지"라는 메시지를 write하여 전송
3. "침입감지"메시지가 클라이언트에게 전달되었으면 클라이언트는 읽을게 있으니 read를 수행하여 메시지를 message버퍼에 저장 후 서버의 주소(serv_adr.sin_addr) + 서버에게 전달받은 메시지 + 현재 시간을 출력해준다


clnt_sock생성
serv_sock생성
serv_sock바인드
connect
serv_sock listen
accept
부모 프로세스는 다시 listen
fork를 통한 자식
gpio 17번 센서
침입감지 x=> gpioRead(17) == 0
침입감지 => gpioRead(17) == 1
침입감지 메세지를 write 하여 클라이언트에 전달