

# Embedded System Software [CSE4116]

## 과제 1

Department of Computer Science and Engineering, Sogang University, Seoul, South Korea

Data-Intensive Computing and System Laboratory

### 1. 목표

디바이스 컨트롤과 IPC를 이용하여 Simple Key-Value Store를 만든다.

### 2. Key-Value Store 아키텍처

Key-Value Store(KVS)는 NoSQL 데이터베이스에서 주로 사용되며 Key를 고유한 식별자로 Key-Value 쌍으로 데이터를 저장한다.

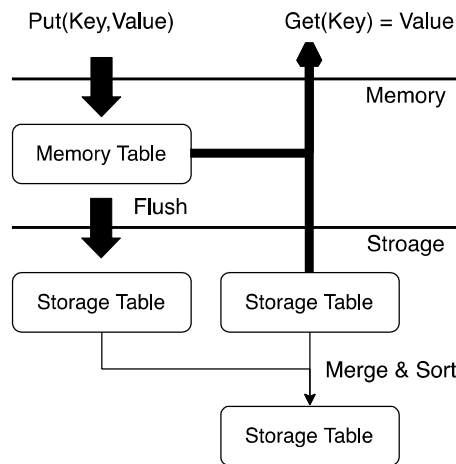


그림1. KVS Architecture

위 그림은 아주 간단한 KVS의 아키텍처를 나타낸다.

이번 과제에서 구현할 KVS의 구성요소와 Operation은 다음과 같다.

#### 2.1. 구성요소

- **Memory Table** : Put 으로부터 들어온 Key-Value 쌍이 Append-only manner 로 Memory 에 저장된다.
- **Storage Table** : Memory Table 크기가 지정된 한계에 도달하면 파일을 생성해 Storage 에 저장된다.

#### 2.2. Operation

- **Put** : (Key, Value) 쌍으로 데이터를 입력받는다.
- **Get** : Key 를 입력받아 현재 KVS 에 해당 Key 를 가진 Value 가 있으면 반환한다.
- **Flush** : Memory Table 크기가 지정된 한계에 도달한 경우 파일을 생성하여 Storage 에 저장한다.
- **Merge** : Storage Table 개수가 지정된 개수에 도달한 경우 전체 Storage Table 을 확인한다.

그리고 Merge 할 Storage Table 을 선택하여 그 중 중복된 Key 를 갖는 Value 를 제거하고 Key 를 기준으로 정렬하여 새로운 Storage Table 을 만들어 저장한다.

### 3. 구현

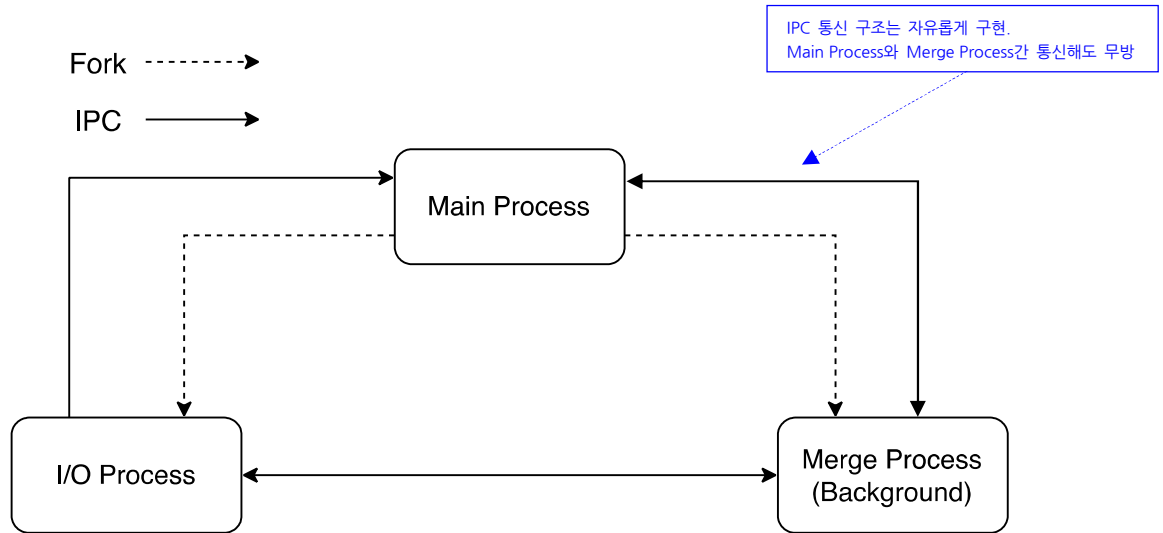


그림2. 과제1 프로그램 프로세스 구조

Main process는 처음 2개의 프로세스를 fork한다. 본 프로그램은 총 3개의 프로세스로 구성되고, 프로세스간 통신은 IPC를 사용한다. 이번 과제에서 IPC는 Message Queue 와 Shared Memory 방법을 모두 사용하여 선택적으로 적용해 구현한다. 둘 중 하나만 사용하면 감점. Semaphore 사용은 자유.

Ex) I/O Process ↔ Merge Process 간에 Shared Memory 사용,  
Main Process ↔ I/O Process 간에 Message Queue 사용      혹은 반대

IPC 구현 내용은 주석과 보고서로 상세히 명시해야 한다. 각 프로세스의 구성은 다음과 같다.  
(프로세스의 구성과 프로그램 기능 구조가 1대1로 일치하지 않으므로 차이를 잘 이해할 것.)

#### 3.1. Main Process

- I/O Process로부터 IPC 를 통해 전달받은 요청(Put, Get)을 처리한다.

#### 3.2. I/O Process

- Device Control 을 사용하여 입출력을 관리한다.
- 사용자로부터 요청된 Operation 을 IPC 를 통해 Main Process 로 전달한다.

#### 3.3. Merge Process (Background)

- I/O Process로부터 IPC 를 통해 전달받은 Merge 요청을 처리한다.
- 또는 Merge Mode 에서 Merge 를 수동으로 수행한다.

#### 3.4. Device Control

- LED 디바이스는 mmap()함수를 사용하고, 나머지 디바이스들은 디바이스 드라이버를 사용하여 프로그램을 구현한다. 이를 보고서에 상세히 명시해야 한다.

## 4. 기능

본 과제에서는 Put, Get, Merge 세가지 기능을 구현한다. 모드 1에서는 Put 기능을, 모드 2에서는 Get 기능을, 모드 3에서는 Merge 기능을 구현한다. 기본적으로 모드가 변경되면, 각 모드의 초기 상태가 된다. Text LCD의 첫번째 줄에는 항상 어떤 모드인지 표시한다. (PUT Mode, GET Mode, MERGE Mode)

각 모드에서 언급이 없는 나머지 디바이스의 초기 상태는 빈 화면, 혹은 불이 모두 꺼진 상태이다.

또한, 프로그램 시작 시 기본 모드(default mode)는 모드 1이다. 각 모드 변경 시, 현재 사용 중인 모드를 제외한 다른 모드에서 사용하는 디바이스들은 모두 초기화 시킨다. 전체 프로그램은 "BACK" 키를 입력하였을 때만 정상 종료되며 이 때 모든 device는 각자의 가장 초기 상태가 된다. (FND 0000, 나머지 device off)

### 4.1. READKEY

- BACK : 프로그램 종료
- PROG
- VOL+ : Mode 변경 Put → Get → Merge → Put
- VOL- : Mode 변경(역방향) Put → Merge → Get → Put

### 4.2. FPGA 모듈

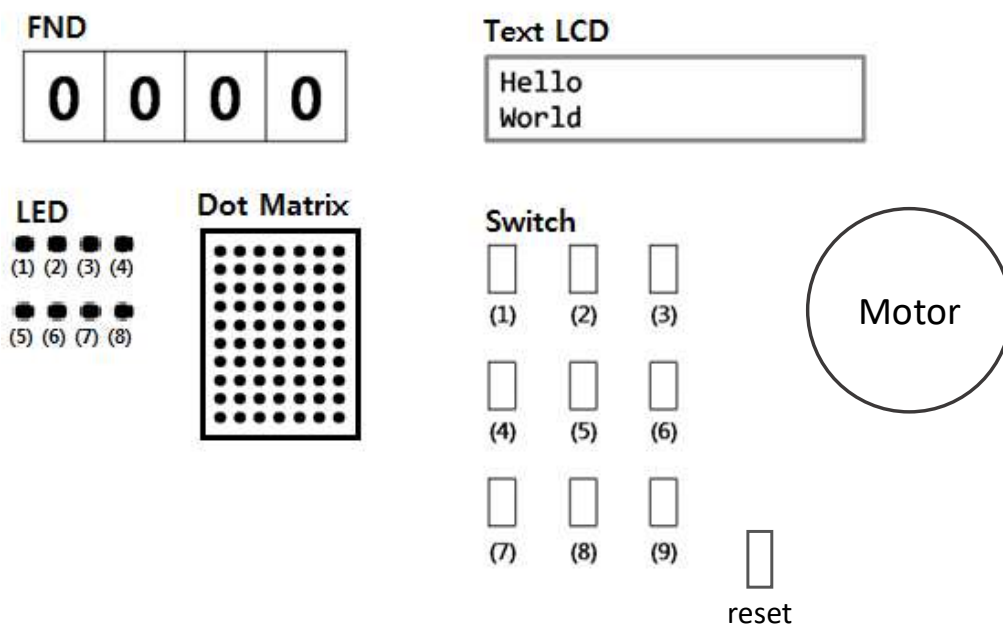


그림3. FPGA Module diagram

- 과제 1 사용

FND

Text LCD

LED

Reset

Switch

Motor

### 4.3. Put 모드

Key-Value 쌍을 설정하여 Put 요청을 전송한다. Key-Value쌍은 Memory Table에 저장된다. Memory Table 이 꽉 찬 경우 Main Process를 호출하여 Memory Table 공간을 확보하기 전까지(Flush동작 전까지) Put은 정지된다.

- FND : 설정할 Key 값을 출력한다.  
입력되는 순서에 따라 왼쪽에서부터 한 숫자씩 출력한다.  
Ex) 1234 가 출력 되는 과정  
1000 → 1200 → 1230 → 1234

숫자 0 입력 버튼이 없으므로 0 이 중간에 있는 테스트는 수행하지 않음. Ex) 2024, 7080  
Key 값은 고정적으로 4 자리 숫자이다.

- Text LCD : 첫번째 줄에는 현재 모드를, 두번째 줄에는 설정할 Value 값을 출력한다.  
Put 완료 시 KVS 에서 저장순서, Key, Value 를 순서대로 출력한다. (양쪽 괄호 포함)  
첫번째 저장하는 Key-Value 쌍이 (1000, A) 인 경우 다음과 같다.

```
PUT Mode
(1, 1000,A)
```

이후 두번째로 (2000, B) 를 저장(Put)하면 다음과 같다.

```
PUT Mode
(2, 2000,B)
```

- LED : Put Mode 의 초기 상태는 ①번 LED 에만 불이 들어온 상태이다.
  - Key 입력을 시작하면 ③번 ④번 LED 가 1 초에 하나씩 번갈아 가면서 불이 들어오게 한다.
  - Value 입력을 시작하면 ⑦번 ⑧번 LED 가 1 초에 하나씩 번갈아 가면서 불이 들어오게 한다.
  - Put 이 완료되면 전체 LED 에 불이 들어왔다가 이후 다시 ①번 LED 에만 불이 들어오게 한다.
- Reset : Key 와 Value 입력을 전환한다. 최초에는 Key 입력으로 시작한다.

- Swtich : 알파벳과 숫자를 입력 받는 버튼이다. 알파벳과 숫자를 입력해서 Key 와 Value 값을 설정한다.
- Key 입력시에는 FND 에 출력하고, Value 입력시에는 LCD 로 출력한다.

〈Switch 기능 상세〉

초기 상태는 숫자 입력이다.



그림4. Switch key mapping

새로운 버튼의 입력이 한 번 들어올 때마다 한 글자씩 출력한다. 한번 눌렀던 버튼을 다시 누를 때마다 해당 알파벳을 입력 횟수에 맞게 변경한다. 즉 ②번 버튼을 한 번 누르면 A 를 출력하고, ②번 버튼을 세 번 입력하면 C 를 출력한다. ④, ⑥번 버튼의 입력이 한꺼번에 들어오기 전까지 LCD 에 출력되는 string 은 항상 덧붙여 출력한다.

①번 버튼을 누르면 알파벳 입력에서 숫자 입력으로 바꾼다. LCD 에 출력되는 string 의 값은 변하지 않고, 기존에 입력된 text 에 덧붙여 입력한다. 숫자 입력의 경우 한 번 누를 때마다 해당하는 숫자를 출력한다. 같은 버튼을 여러 번 누르면 버튼을 누를 때마다 같은 숫자를 연달아 출력한다.

Ex ) ② ⑥ ⑨ ⑨ ② ② : AMXB

② ② ② ② ⑤ ⑤ ⑦ ⑦ ① ⑨ ⑨ : AKQ99

⑥ ⑥ ③ ③ ⑥ ⑥ ⑥ ① ⑦ ① ⑤ ⑤ : NEO7K

연달아 같은 버튼에 위치한 알파벳을 입력하는 테스트는 수행하지 않음. Ex) AA, ABC

~~Text LCD 에 최대 16 글자만 출력할 수 있으므로~~ Value 의 길이는 최대 5 글자이다.

5 글자가 넘는 입력은 받을 수 없도록 해야 한다.

특수키 기능

① : 영어 ⇄ 숫자 입력을 바꿔준다.

① 1 초 이상 입력 : 입력중인 Key 또는 Value 초기화. 이 입력이 들어오면 FND 또는 LCD 에 있던 값을 없애고 다시 빈 상태로 만들어준다.

버튼 두 개 동시 입력

④ + ⑥ : 설정된 Key, Value 쌍을 저장(Put 요청)한다.

- Key 입력시에는 FND 에 출력하므로 알파벳 입력이 불가능하다.
- 언급이 없는 나머지 디바이스는 빈 화면, 혹은 불이 모두 꺼진 상태를 유지한다.

#### 4.4. Get 모드

Key를 설정하여 Get 요청을 전송한다.

- FND : 입력할 Key 값을 출력한다.
- LCD : 첫번째 줄에는 현재 모드를, 두번째 줄에는 Get 결과 값을 출력한다.  
Get 성공 시 KVS 에서 저장순서, Key, Value 값을 다음과 같이 출력한다. (양쪽 괄호 포함)

```
GET Mode
(1, 1000,A)
```

요청한 Key 값이 없을 경우 Error 를 출력한다.

```
GET Mode
Error
```

- LED : 초기 상태는 ⑤번 LED 에만 불이 들어온 상태이다.
  - Key 입력을 시작하면 ③번 ④번 LED 가 1 초에 하나씩 번갈아 가면서 불이 들어오게 한다.
  - Get 이 완료되면 전체 LED 에 불이 들어왔다가 이후 다시 ⑤번 LED 에만 불이 들어오게 한다.
- Swtich : 숫자를 입력(Put 과 동일) 받는 버튼이다. 숫자를 입력해서 Key 값을 설정하고 FND 에 출력한다.
- Reset : Get 요청을 보낸다.

#### 4.5. Merge 모드 & Background Job

Merge는 이 모드에서 직접 수행하거나 I/O Process로부터 호출되는 **두 가지 경우가 가능하다.**

Merge 수행은 가장 오래된 2개의 Storage Table에서 <sup>1)</sup>중복된 Key를 갖는 Key-Value 쌍들을 가장 최근에 입력된 Key-Value 쌍 하나로 merge하고 <sup>2)</sup>Key를 기준으로 오름차순으로 정렬한 다음 <sup>3)</sup>새로운 Storage Table을 만들어 저장한다.

아래는 Merge 모드에서 직접 수행할 때의 기능이다.

**단, Motor는 I/O Process로부터 호출되어 Merge가 수행되는 경우에도 회전한다.**

- LCD : 첫번째 줄에는 현재 모드(MERGE Mode)를, 두번째 줄에는 Merge 수행 결과 값을 출력한다.  
Merge 성공 시 생성된 Storage Table 의 이름과 Key-Value 쌍의 개수를 출력한다.  
(두번째 줄 출력 형식 자유)
- Reset : Merge 요청을 보낸다.
- Motor : Merge 수행 시 1-2 초(적당히)간 회전한다.

## 5. Key-Value Store 상세 구현 설명

### 5.1. Memory Table

- 최대 Key-Value 쌍 저장 개수는 3 개이다.
- Key-Value 쌍이 3 개 이후 Put 이 발생하면 먼저 입력된 3 개의 Key-Value 를 Storage Table 로 저장한다.
- 반드시 메모리에 저장해야 한다 (파일 사용 x)
- 프로그램 종료 시 Memory Table 에 남아있는 데이터는 Key-Value 쌍 개수에 상관 없이 반드시 Storage Table 로 저장해야 한다.

### 5.2. Storage Table

- 확장자는 .st 를 사용한다.
- 1 개의 Key-Value 쌍은 파일에서 1 줄을 차지한다.
- Key-Value 쌍은 KVS 에서 순서, Key, Value 가 공백으로 구분되어 순서대로 저장된다.  
첫번째로 (1000,A), 두번째로 (2000,B), 세번째로 (3000,C) 가 저장된 Storage Table 의 경우 다음과 같다.

```
1 1000 A
2 2000 B
3 3000 C
```

### 5.3. Get/Put

- Get 은 반드시 가장 최근에 입력된 순서대로 탐색해야 한다 (Memory Table → Storage Table).

### 5.4. Merge

- Merge 가 시작되는 Storage Table 개수는 3 개이다.
- Merge 수행시에도 중복된 Key 가 없어 Key-Value 개수를 줄일 수 없는 경우 정렬만 하여 3 개 이상의 **Key-Value 쌍**을 갖는 Storage Table 을 생성한다.
- Merge 예시 1  
(수행 전)

2.st	1.st
4 3000 F	1 1000 C
5 4000 E	2 1000 B
6 3000 D	3 1000 A

(과정)

2.st	1.st
<del>4 3000 F</del> (중복제거)	<del>1 1000 C</del> (중복제거)
5 4000 E	<del>2 1000 B</del> (중복제거)
6 3000 D	3 1000 A

(수행 후)

3.st (파일명은 가장 마지막 순서)
1 1000 A
2 3000 D
3 4000 E

- Merge 예시 2  
(수행 전)

2.st	1.st
4 5000 E	1 4000 D
5 1000 A	2 2000 B
6 6000 F	3 3000 C

(수행 후)

3.st
1 1000 A
2 2000 B
3 3000 C
4 4000 D
5 5000 E
6 6000 F

## 5.5. 기타

- 프로그램 재실행 시 KVS 의 Storage Table 의 Database 는 유지되어야 한다. 구현(Metadata 관리 등)은 자유롭게 하되 명세서에 상세히 작성해야 한다.
- 명세서에 나와있지 않은 예외처리는 자유롭게 구현하되 명세서에 상세히 작성해야 한다.



## 6. 제출 방법

- 아래 제출 파일을 사이버 캠퍼스에 업로드
- [프로그램 제출란]
  - 소스코드, Makefile, Readme 파일을 포함한 [HW1]학번.tar.gz  
Ex) [HW1]20240000.tar.gz
  - 파일 압축 방법 : 학번 폴더를 생성하고, 그 안에 숙제 관련 파일을 저장한 뒤, 학번 폴더가 있는 위치에서 `tar -cvzf [HW1]학번.tar.gz ./학번폴더`
- [보고서 제출란]
  - [HW1]학번\_reprt.pdf  
Ex) [HW1]20240000\_report.pdf
  - 제공되는 보고서 형식 참고. 세부 목차/내용 추가 가능.
  - **보고서는 반드시 PDF로 제출**
- Due Date :   프로그램 - 2024 년 4 월 16 일 화요일 23:59  
                  보고서 - 2024 년 4 월 18 일 목요일 23:59

## 7. 평가내용

- 프로그램 100 점(주석 점수 포함), 보고서 20 점.
- Late : 하루에 10% 감점  
Due Date 기준 프로그램은 +5 일까지, 보고서는 +3 일까지.
- 제출 형식 틀린 경우 10% 감점.
- Copy 적발 시 0 점.
- Copy 의심자는 조교와 면담을 통해 소명해야 함.

## 8. 참고

Input process 작성 시 "readkey"를 다음과 같이 non-blocking 방식으로 사용 할 수 있습니다.  
실습 자료에 있는 readkey.c 의 코드에서 파일 open 하는 부분을 아래와 같이 수정하여 사용하시길 바랍니다.

```
char* device = "/dev/input/event0";  
if((fd = open (device, O_RDONLY|O_NONBLOCK)) == -1) {  
    printf ("%s is not a vaild device\\n", device);  
}
```