Programming Assignment #5

Due: 19th Dec. (Thu), 5:59 PM

1. Introduction

본 과제는 crash consistency를 보장하는 Key-value 데이터베이스를 제작하는 것으로, 데이터베이스에서 내 crash consistency 보장의 필요성 및 동작에 대한 전반적인 이해를 돕는다.

2. Problem Specification

Key-value 데이터베이스 API 및 역할

db_t *db_open(int size);

- 현재 디렉토리 위치에 파일을 저장한 "./db" 디렉토리를 생성한다. (이미 생성되어 있는 경우 해당 디렉토리를 사용)

- 해시 테이블의 엔트리 개수는 주어진 size로 만든다.

void db_close(db_t *db);

- 데이터베이스에서 사용했던 자원들을 모두 해제한다.

- 해시 테이블에 데이터가 남아 있을 경우, "./db" 디렉토리 밑의 하나의 파일로 저장한다.

void db_put(db_t *db, char *key, int keylen, char *val, int vallen);

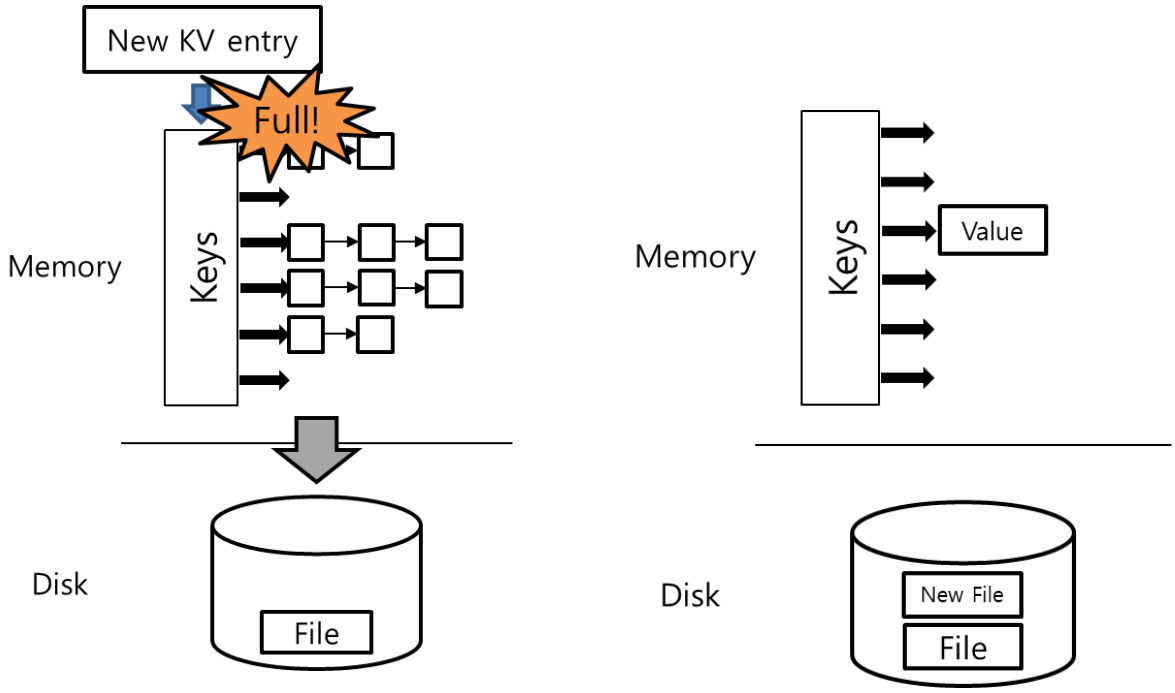
- 주어진 key와 value를 데이터베이스에 저장한다.

- 해시 테이블의 엔트리 개수가 꽉 찬 상태에서 메모리 상에 없는 엔트리가 추가될 때, 해시 테이블 내 모든 데이터를 "./db" 디렉토리 밑에 한 개의 파일로 저장한다. (저장된 해시 테이블 내 데이터는 모두 free)

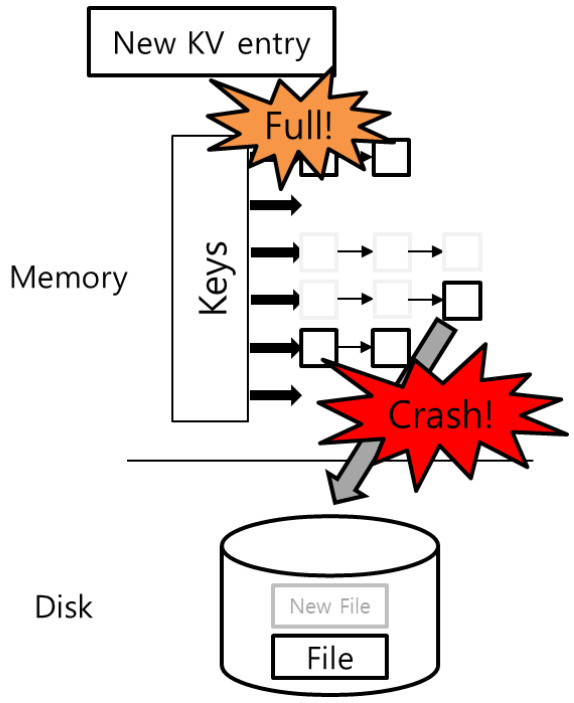
char *db_get(db_t *db, char *key, int keylen, int *vallen);

- 주어진 key가 메모리에 존재하지 않는다면 데이터들을 저장했던 파일에서 검색 및 value를 반환한다.

- 주어진 key가 존재하지 않는다면 NULL을 반환한다.



- Hash table이 full인 상태에서 hash table에 존재하지 않은 key에 대한 db_put을 수행할 경우, 모든 KV 쌍을 한 파일에 저장한 후, 메모리 상에서 free시킨다. 이후 hash table에 새로운 KV 쌍을 입력한다.



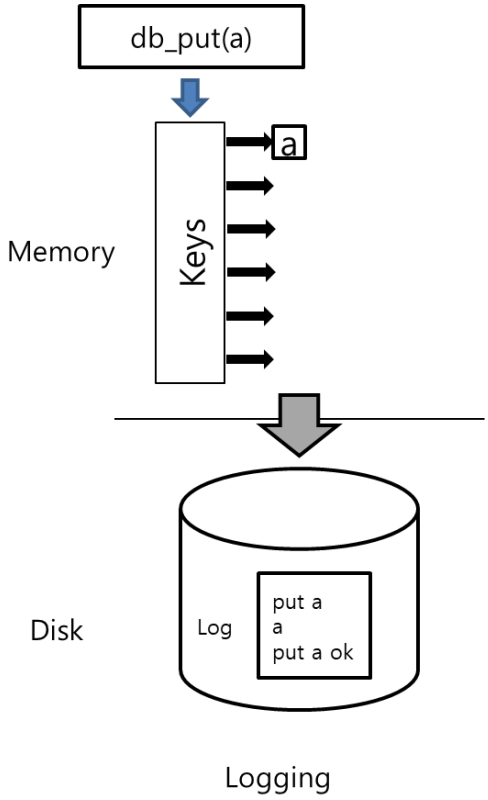
- Hash table 내 KV 정보를 disk에 저장하는 도중 crash가 발생하면(전원 off, 강제 종료 등) flush 동작이 완료되지 못한 채 종료된다.

- Crash 이후 다시 database를 open하고 사용할 경우, disk 내 불완전하게 생성된 database file을 참조하게 되는 문제가 발생한다.

- 1) Hash table에서 disk로의 data flush 진행 중에 crash가 발생하는 상황을 고려한 crash consistency 정책에 대해 database 동작을 구현한다. (ex. data flush 동작, db_open 시의 동동작 등)

- Hint) Data flush가 진행되면 disk 내 임시 파일을 생성하여 data를 기입할 수 있다. 기입 도중 crash가 발생할 경우 해당 임시 파일을 그냥 무시하는 것으로 해결될 수 있다.

- Hint) 임시 파일에 대한 data flush가 완료된 경우, 임시 파일에서 database 파일로의 변환을 위해 rename, 현재 디렉토리에 대한 fsync 과정이 수반된다. (syscall rename, 현재 directory에 대한 fsync 방법 숙지 필요)



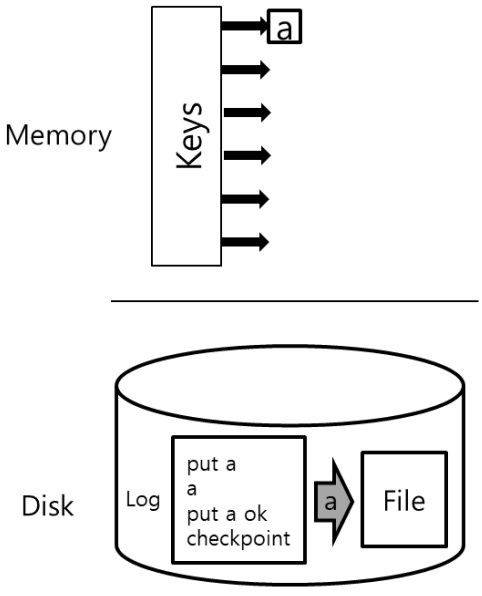
Logging

- 1) 과제는 crash로 인한 불완전한 file의 생성 및 불완전한 file에 대한 접근을 막을 수 있으나 잃게 되는 data 들에 대한 보장은 지원하지 못한다.

- 2) 과제에서는 "PUTOK"를 받은 data들이 반드시 disk 내에 존재하도록 하는 수준의 crash consistency를 구현한다.

- Logging은 database로의 다양한 작업에 대한 footprint 및 data를 남겨두는 기법으로, disk 내 database log 공간에 동작을 차례로 기입하며 database crash 등의 상황에서 log 참고를 통해 복원을 진행할 수 있다.

- Database를 open할 때, log 파일의 checkpointing 위치를 확인하여 복구 필요한 부분이 있는지 확인한다.



Checkpointing

- Checkpointing은 database log의 data를 disk 내 file로 옮기는 과정으로 log에 대한 checkpointing 과정 수행 후 log에 checkpointing이 해당 위치까지 완료되었음을 명시한다.

- Database crash 발생으로 log를 통한 복구 작업을 수행할 때, checkpointing log의 위치를 통해 복구를

수행할 위치를 결정할 수 있다.

- Checkpointing 시점은 hash table이 꽉 차 data flush 작업이 발생해야 하는 시점으로 한다.

$./wordcount 128 GET [EMMA] PUT [EMMA] [1] GET [EMMA] PUT [EMMA] [2] GET [JOHN] PUT [JOHN] [1] DB_CLOSE $./wordcount 128 GET [EMMA] PUT [EMMA] [3] DB_CLOSE	DB opened DB log file opened GETOK [EMMA] [NULL] PUTOK GETOK [EMMA] [1] PUTOK GETOK [JOHN] [NULL] PUTOK DB closed DB opened DB log file opened GETOK [EMMA] [2] PUTOK DB closed

4. Hand in Instructions

- **본 과제는 이전 과제를 기반으로 진행한다. (db.c db.h 파일 사용 가능, main.c 파일 새로 제공)**
  - **1) 까지 과제를 수행한 경우 만점, 2) 까지 과제를 수행한 경우 추가점수가 배점된다.**
  - **main.c 파일에는 db_log file 생성 코드가 첨부되어 있으며, 추가적으로 PA4에서와 같이 database가 db_get, db_put을 요청을 기다리는 형태로 구현한다.**
  - **(main.c 참고) Log file open 시 O_DIRECT flag가 추가되어있어 write 수행 시, buffer align을 맞춰 write를 수행해야 한다. (512Bytes의 배수)**
    - **posix_memalign(), memalign() 참조**
  - **Hash table size는 100으로 고정한다.**
  - **Crash test를 위해 data flush가 발생하는 중간에 ctrl+c signal을 통해 임의로 crash를 발생시킨다.**

◆ **따라서 ctrl+c에 대한 핸들러 구현을 금한다.**

- 제출 시 구현 코드와 보고서를 함께 압축하여 제출한다.
  - ■ 압축 파일은 Makefile / main.c / db.c / db.h / README.pdf 로 이루어져 있어야 하며, 압축 파일의 이름과 확장자는 "학번.tar.gz"로 한다.
  - ■ 구현 코드 상단에 이름 및 학번을 명시한다.
  - ■ 구현 코드와 별도로 구현에 대한 내용을 담은 보고서를 함께 제출한다. 보고서의 파일 형식은 pdf로 제한한다.
- 본 과제는 개인 과제이며 icampus의 과제 란에 제출한다.
- 과제 제출 시간은 icampus 제출 시각 기준으로 하며, 매 24시간마다 25%씩 감점된다.
- 과제 관련 주의 사항
  - ■ Copy 또는 미제출 2회 이상 시, F학점