System Software Experiment 2

SSE 2 lab4 report

Thread를 이용한 database 구축

이번 lab4의 목표는 thread를 사용하여 data의 병렬적 처리를 목적으로 한다. 따라서 병렬적 처리가 필요한 부분에서 데이터를 나누어 각 thread가 처리할 수 있도록 만들었다.

함수

main.c

```c
#include "db.h"
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

int main(int argc, char* argv[]) {
    db_t* DB;
    char key[MAX_KEYLEN];
    int size, val, th_num;
    int ret, keylen, offset;
    if(argc < 3) {
        printf("Usage : %s size\n", argv[0]);
        return -1;
    }
    size = atoi(argv[1]);
    th_num = atoi(argv[2]);
    DB = db_open(size, th_num);
    if(DB ==NULL) {
        printf("DB not opend\n");
        return -1;
    }
    printf("DB opened\n");
    while((ret =scanf("%s", key)) != -1) {
        keylen = strlen(key);
        offset = -1;
        val = db_get(DB, key, keylen, &offset);
        if(val == 0)
            printf("GET [%s] [NULL]\n", key);
        else
            printf("GET [%s] [%d]\n", key, val);
        db_put(DB, key, keylen, val+1, offset);
        printf("PUT [%s] [%d]\n", key, val+1);
    }
    db_close(DB);
    printf("DB closed\n");
    return 0;
}
```

main 함수에서는 기존의 함수 형태를 유지하여 데이터를 직접적으로 처리하게 한다.

dbopen 함수에서는 미리 MAX_FILE수만큼 파일을 미리 만들어놓고 메모리도 이 파일의 개수만큼의 데이터베이스를 생성한다.

```
db_t* db_open(int size, int t_num) {
    int i, fd;
    char dir[MAX_DIR] = "./db";
    db_t* db = NULL;
    th_n = t_num;
    db_s = size;
    kv_s = 0;
    if(opendir(dir) == NULL){
        mkdir(dir, 0755);
        for(i=0;i<MAX_FILE;i++){
            sprintf(dir, "./db/%d.txt", i);
            fd = open(dir, O_CREAT|O_WRONLY, 0755);
            close(fd);
        }
    }
    db = (db_t*)malloc(sizeof(db_t)*t_num);
    for(i=0;i<t_num;i++) db[i].head = NULL;
    return db;
}
```

dbclose 함수에서는 thread를 생성하여 현재 메모리에 존재하는 모든 정보를 파일로 옮기고 데이터베이스의 자원을 해제한 후 종료한다.

```
void db_close(db_t* db) {
    int i;
    pthread_t* tid = (pthread_t*)malloc(sizeof(pthread_t)*th_n);
    for(i=0;i<th_n;i++)
        pthread_create(&tid[i], NULL, th_file_put, (void*)db[i].head);
    for(i=0;i<th_n;i++)
        pthread_join(tid[i], NULL);
    for(i=0;i<th_n;i++)
        free(db[i].head);
    free(db);
}
```

dbget 함수에서는 기존의 파일 개수를 정해놓았기 때문에 메모리에서 찾을 때에도, 파일에서 찾을 때에도 하나의 메모리 공간, 또는 하나의 파일만 탐색하면 되기 때문에 매우 빠르게 탐색이 가능하다. 따라서 thread의 사용없이 메모리 또는 파일에서 탐색을 한다.

```
int db_get(db_t* db, char* key, int keylen, int* offset){
    int fd, wtp, keysize, value;
    int hashed = hash_func(key, db_s), fhash = hash_func(key, MAX_FILE), thash;
    char dir[MAX_FILE];
    char buf[MAX_KEYLEN];
    node *temp, *temp_n;
    thash = fhash%th_n;
    if(thash < 0) thash += th_n;
    if(db[thash].head != NULL){
        temp = db[thash].head;
        if(temp[hashed].key != NULL){
            temp_n = &temp[hashed];
            while(temp_n != NULL){
                if(!strcmp(key, temp_n->key)){
                    *offset = temp_n->offset;
                    return temp_n->value;
                }
                temp_n = temp_n->next;
            }
        }
    }
    kv_s++;
    value = 0;
    sprintf(dir, "./db/%d.txt", fhash);
    fd = open(dir, O_RDONLY);
    while((wtp = read(fd, &keysize, sizeof(int))) > 0){
        memset(buf, 0, MAX_KEYLEN);
        wtp = read(fd, buf, keysize);
        buf[keysize] = '\0';
        if(!strcmp(buf, key)){
            *offset = lseek(fd, 0, SEEK_CUR);
            wtp = read(fd, &value, sizeof(int));
            break;
        }
        lseek(fd, sizeof(int), SEEK_CUR);
    }
    close(fd);
    return value;
}
```

dbput 함수에서는 메모리에 데이터를 저장할 때는 hash 값에 따라 필요한 공간만 탐색해 저장을 하고 key-value 쌍이 db size보다 커지는 순간 쓰레드를 이용하여 이미 쓰레드 개수만큼 나뉜 메모리 공간을 인자로 pthread_create를 하고 각 쓰레드가 본인의 database 파일을 파일로 내린다.

```
    if(kv_s < db_s) return;
    kv_s = 0;
    tid = (pthread_t*)malloc(sizeof(pthread_t)*th_n);
    for(i=0;i<th_n;i++)
        pthread_create(&tid[i], NULL, th_file_put, (void*)db[i].head);
    for(i=0;i<th_n;i++)
        pthread_join(tid[i], NULL);
    for(i=0;i<th_n;i++){
        free(db[i].head);
        db[i].head = NULL;
    }
    free(tid);
    return;
}
```

thread로 실행되는 이 함수에서는 db의 한 공간에 저장되어 있는 내용을 파일로 내리는 역할을 한다.

```c
void* th_file_put(void* arg){
    int fd, keysize, wtp;
    int i;
    char dir[MAX_DIR];
    node *head = (node*)arg;
    node *temp, *temp_p;
    if(head == NULL) pthread_exit(NULL);
    for(i=0;i<db_s;i++){
        if(head[i].key == NULL) continue;
        temp = &head[i];
        while(temp != NULL){
            temp_p = temp->next;
            sprintf(dir, "./db/%d.txt", hash_func(temp->key, MAX_FILE));
            fd = open(dir, O_CREAT | O_WRONLY, 0755);
            if(temp->offset == -1){
                lseek(fd, 0, SEEK_END);
                keysize = strlen(temp->key);
                wtp = write(fd, &keysize, sizeof(int));
                wtp = write(fd, temp->key, keysize);
            }else lseek(fd, temp->offset, SEEK_SET);
            wtp = write(fd, &temp->value, sizeof(int));
            free(temp->key);
            if(temp_p != head[i].next) free(temp);
            close(fd);
            temp = temp_p;
        }
    }
    wtp = wtp;
    pthread_exit(NULL);
}
```

그리고 자동으로 메모리 공간을 초기화하게 된다.

메모리에서 저장된 데이터들을 파일로 내릴 때 쓰레드를 사용하여 병렬적으로 내리고 일부러 메모리 공간의 데이터들이 파일 개수에 따른 해쉬에 영향을 주지 않게 했기 때문에 thread_safe하고 파일에서 데이터를 찾은 경우에는 offset을 메모리에 임시 저장해놓아서 다시 파일로 옮길 때 새로운 공간이 아닌 원래 적혀있던 공간에 썼기 때문에 data compaction이 자동으로 이루어진다. 파일에는 아예 사용하지 않는 공간이 있기는 하나 이 파일이 사라진다고 해도 다시 생성하고 찾기도 용이하다. 그리고 파일에 내릴 때에는 기존 database size에 영향을 받지 않기 때문에 새로 database size를 다르게 주는 것도 영향이 없어서 신경쓰지 않아도 파일에서 찾을 수 있다.