OS 3차과제

(Designing a Virtual Memory Manager)

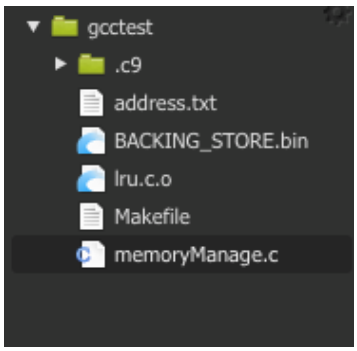
12114497 컴퓨터공학과 한정

Designing a Virtual Memory Manager

요구사항

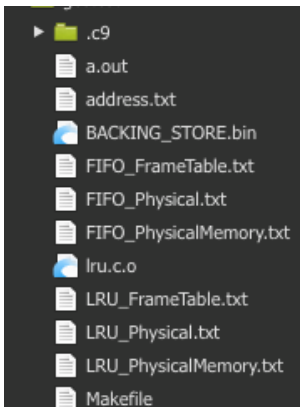
- TLB와 PageTable, FrameTable로 메모리를 관리합니다.
- TLB Entry의 갯수는 16, pagetable Entry의 갯수는 256, frametable Entry의 갯수는 128 개 입니다.
- Tlb table, LRU 알고리즘으로 구현되는 frameTable, FIFO 알고리즘으로 구현되는 frameTable의 Hit율을 출력해줍니다.
- 출력 파일은 Physical 주소가 담긴 파일, frameTable 내용이 담긴 파일, backingStore내용이 담긴 파일이 출력되야 합니다.

결과화면



- 코드와 address.txt 인풋 파일만 있는 프로젝트의 디렉토리 입니다.

- 인풋을 넣고 실행한 모습입니다. Hit율이 각각 55, 461, 462이 나오게 됩니다.



- 아웃풋 파일 총 6개가 생성 된 것을 볼 수 있습니다.

코드 및 설명

```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
/*Entry Size 정의*/
#define TlbTableSize 16
#define pageTableSize 256
#define frameTableSize 128
struct Node{
    int pageNumb;   //page #
    int frameNumb;  //frame #
    int validBit;   // 1: valid, 0: invalid
    struct Node *prev;  //prev Node
    struct Node *next;  //next Node
};
struct TableInfo{
    struct Node *firstNode; // LRU서 가장 최근에 사용한 것을 firstNode로 사용
    struct Node *lastNode;  // LRU서 full일 때 교체대상은 lastNode입니다.
    int size;   // table Entry Number
    int hit;    // table Hit ratio
};
int address, offset, pageNumber;
char addressInput[10];
unsigned char physicalMemory[128][256]; //page fault 떄 backingStore save
int frameTableResult[129][2];   //frame table 출력 시 사용
/*
 * 변수명 : pageTable
 * 내용 : 인덱스는 page #를 의미. 0은 frameNumb, 1은 validBit이다.
 */
int pageTable[256][2];
struct TableInfo frameTable, tlbTable;   //table선언
/* 함수 선언 부분, 뒤에서 자세히 설명하겠습니다. */
int insertNode(int,int);
int getFrameNumb();
int tlbTableLookUp(int);
void insertTlbNode(int, int);
void insertFrameNode(int, int);
void updateFrameNodeLRU(int);
void updateTlbNode(int);
void loadToFrame(int,int);
/*
 * 함수명 : loadToFrame(int pageNumb, int frameNumb)
 * 내용 : page fault상황에서 frame에 backingStore값을 로드합니다.
 *      pageNumb를 이용하여 read할 부분을 찾아 physicalMemory에
 *      backing store 값을 올려주게 됩니다.
 */
void loadToFrame(int pageNumb, int frameNumb) {
   FILE *backingStore = fopen("BACKING_STORE.bin", "rb");
   fseek(backingStore, pageNumb * 256, SEEK_SET);
   fread(physicalMemory[frameNumb], 256, 1, backingStore);
   fclose(backingStore);
}
/*
 * 함수명 : insertNode(int pageNumb, int type)
 * 내용 : type이 0이면 lru, type이 1이면 FIFO를 의미합나디.
```

```
 *      1. 먼저 tlbTableLookUp을 합니다. 있으면 frameNumb에 -1이 아닌
 *        다른 값이 들어가게 됩니다.
 *      2-1. tlbTable에 있는 경우 Frametable과 tlbTable을 업데이트 해줍니다.
 *          이 경우 tlbTable의 히트횟수를 증가 시킵니다.
 *      2-2. tlbTable에 없는 경우 먼저 pageTable을 확인 합니다.
 *      2-2-1. validBit가 1일 경우 이전에 참조 한 것이므로 Tlb table에 insert를
 *             한 후 FrameTable을 업데이트 합니다. 이 때 frameTable hit 횟수를
 *             증가시켜 줍니다.
 *      2-2-2. validBit가 0일 경우 FrameTable과 TLB테이블에 모두 insert를 합니다.
 *      FIFO일 경우 Framenode update를 할 필요가 없게 됩니다.
 */
int insertNode(int pageNumb, int type){
    int frameNumb = tlbTableLookUp(pageNumb);
    if(frameNumb != -1){
            updateTlbNode(pageNumb);
            if(type == 0){
                updateFrameNodeLRU(pageNumb);
            }
            tlbTable.hit++;
    } else{
        if(pageTable[pageNumb][1] == 1){
            frameNumb = pageTable[pageNumb][0];
            insertTlbNode(pageNumb,frameNumb);
            if(type == 0){
                updateFrameNodeLRU(pageNumb);
            }
            frameTable.hit++;
        }else{
            frameNumb = getFrameNumb();
            insertTlbNode(pageNumb,frameNumb);
            insertFrameNode(pageNumb,frameNumb);
        }
    }
    return frameNumb;
}
/*
 * 함수명 : getFrameNumb()
 * 내용 : frame table이 꽉 찼을 경우 victim인 lastNode의 frame number를 사용합니
다.
 *      꽉 차지 않았을 경우 frameTable의 size를 사용할 수 있습니다.
 */
int getFrameNumb(){
    if(frameTable.size == frameTableSize){
        return frameTable.lastNode->frameNumb;
    } else{
        return frameTable.size;
    }
}
/*
 * 함수명 : tlbTableLookUp(int pageNumb)
 * 내용 : tlbTable을 룩업하며 pageNumb를 가진 노드가 있는지 찾습니다.
 *      있을 경우 frameNumber를 없을경우 -1을 return 합니다.
 */
int tlbTableLookUp(int pageNumb){
    int frameNumb = -1;
```

```
    struct Node *findNode = tlbTable.firstNode;
    while(findNode != NULL){
        if(findNode->pageNumb == pageNumb){
            frameNumb = findNode->frameNumb;
            break;
        }
        findNode = findNode->prev;
    }
    return frameNumb;
}
/*
 * 함수명 : insertTlbNode(int pageNumb, int frameNumb)
 * 내용 : 1. Tlb table에 노드를 insert하는 함수입니다.
 *      1-1. tlbTable이 full일 경우 lastNode를 victim으로 하여 해제시켜 줍니다.
 *      2. 새로운 노드를 만들어 parameter로 넘겨받은 pageNumb와 frameNumb를
 *         노드에 입력시켜 줍니다. 그리고 새로 들어온 노드는 firstNode로 해주게 됩니다.
 *      2-1. 만약 처음 생성되는 경우 newNode는 firstNode이자 lastNode입니다.
 *      3. 생성 후 size를 증가 시킵니다.
 */
void insertTlbNode(int pageNumb, int frameNumb){
    if(tlbTable.size == TlbTableSize){
        struct Node *victim = tlbTable.lastNode;
        tlbTable.lastNode = victim->next;
        tlbTable.lastNode->prev = NULL;
        free(victim);
        tlbTable.size--;
    }
    struct Node *newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->pageNumb = pageNumb;
    newNode->frameNumb = frameNumb;
    newNode->validBit = 1;
    newNode->next = NULL;
    newNode->prev = tlbTable.firstNode;
    if(tlbTable.size > 0){
        tlbTable.firstNode->next = newNode;
    }
    if(frameTable.size == 0){
        tlbTable.lastNode = newNode;
    }
    tlbTable.firstNode = newNode;
    tlbTable.size++;
}
/*
 * 함수명 : insertFrameNode(int pageNumb, int frameNumb)
 * 내용 : 1. Frame table에 노드를 insert하는 함수입니다. 이 함수에 들어왔다면
 *          page fault가 발생한 경우입니다.
 *      1-1. FrameTable이 full일 경우 lastNode를 victim으로 하여 해제시켜 줍니다.
 *          또한 pageTable의 내용 또한 0으로 바꿔줍니다.
 *      2. loadToFrame을 이용하여 backing store내용을 로드해줍니다.
 *      3. pageTable내용을 새롭게 업데이트 한 후 가지고 있는 정보를 새로운 노드를 만들어
 *        넣어줍니다. 새로운 노드는 firstNode가 됩니다.
 *      3-1. 처음 들어온 노드일 경우 lastNode가 됩니다.
 *      4. 이후 size를 증가시켜줍니다.
 */
void insertFrameNode(int pageNumb, int frameNumb){
```

```
    if(frameTable.size == frameTableSize){
        struct Node *victim = frameTable.lastNode;
        frameTable.lastNode = victim->next;
        frameTable.lastNode->prev = NULL;
        int lastNodePageNumb = victim->pageNumb;
        free(victim);
        pageTable[lastNodePageNumb][0] = 0;
        pageTable[lastNodePageNumb][1] = 0;
        frameTable.size--;
    }
    loadToFrame(pageNumb, frameNumb);
    pageTable[pageNumb][0] = frameNumb;
    pageTable[pageNumb][1] = 1;
    struct Node *newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->pageNumb = pageNumb;
    newNode->frameNumb = frameNumb;
    newNode->next = NULL;
    newNode->validBit = 1;
    newNode->prev = frameTable.firstNode;
    if(frameTable.size > 0){
        frameTable.firstNode->next = newNode;
    }
    if(frameTable.size == 0){
        frameTable.lastNode = newNode;
    }
    frameTable.firstNode = newNode;
    frameTable.size++;
}
/*
 * 함수명 : updateFrameNodeLRU(int pageNumb)
 * 내용 : 1. FrameNode를 LRU 알고리즘을 사용하여 update하는 방식입니다.
 *      pageNumb를 이용하여 현재 노드의 위치를 먼저 찾습니다.
 *      2-1. 그 노드를 firstNode로 바꿔줍니다. 이미 firstNode일 경우
 *          바꿀 필요가 없어 return 하게 됩니다.
 *      2-2. lastNode일 경우 lastNode를 이전 노드의 next로 지정하고
 *          그 lastNode를 firstNode로 업데이트 합니다.
 *      2-3. lastNode, firstNode가 아닐 경우 이전과 이후 노드를 연결하고
 *          firstNode로 올라가게 됩니다.
 *      이를 통해 lastNode를 계속적으로 victim으로 쓸 수 있게 합니다.
 */
void updateFrameNodeLRU(int pageNumb){
    struct Node *findNode = frameTable.lastNode;
    while(findNode!=NULL){
        if(findNode->pageNumb == pageNumb){
            break;
        }
        findNode = findNode->next;
    }
    struct Node *firstNode = frameTable.firstNode;
    struct Node *lastNode = frameTable.lastNode;
    if(firstNode->pageNumb == findNode->pageNumb)
        return;
    if(lastNode->pageNumb == findNode->pageNumb){
        frameTable.lastNode = findNode->next;
        findNode->next->prev = NULL;
```

```
        findNode->next = NULL;
        findNode->prev = firstNode;
        firstNode->next = findNode;
        frameTable.firstNode = findNode;
    } else{
        struct Node *prevNode = findNode->prev;
        struct Node *nextNode = findNode->next;
        prevNode->next = nextNode;
        nextNode->prev = prevNode;
        firstNode->next = findNode;
        findNode->prev = firstNode;
        findNode->next = NULL;
        frameTable.firstNode = findNode;
    }
}
/*
 * 함수명 : updateTlbNode(int pageNumb)
 * 내용 : 1. TLB table을 LRU 알고리즘을 사용하여 update하는 방식입니다.
 *      pageNumb를 이용하여 현재 노드의 위치를 먼저 찾습니다.
 *      2-1. 그 노드를 firstNode로 바꿔줍니다. 이미 firstNode일 경우
 *          바꿀 필요가 없어 return 하게 됩니다.
 *      2-2. lastNode일 경우 lastNode를 이전 노드의 next로 지정하고
 *          그 lastNode를 firstNode로 업데이트 합니다.
 *      2-3. lastNode, firstNode가 아닐 경우 이전과 이후 노드를 연결하고
 *          firstNode로 올라가게 됩니다.
 *      이를 통해 lastNode를 계속적으로 victim으로 쓸 수 있게 합니다.
 */
void updateTlbNode(int pageNumb){
    struct Node *findNode = tlbTable.lastNode;
    while(findNode!=NULL){
        if(findNode->pageNumb == pageNumb){
            break;
        }
        findNode = findNode->next;
    }
    struct Node *firstNode = tlbTable.firstNode;
    struct Node *lastNode = tlbTable.lastNode;
    if(firstNode->pageNumb == findNode->pageNumb)
        return;
    if(lastNode->pageNumb == findNode->pageNumb){
        tlbTable.lastNode = findNode->next;
        findNode->next->prev = NULL;
        findNode->next = NULL;
        findNode->prev = firstNode;
        firstNode->next = findNode;
        tlbTable.firstNode = findNode;
    } else{
        struct Node *prevNode = findNode->prev;
        struct Node *nextNode = findNode->next;
        prevNode->next = nextNode;
        nextNode->prev = prevNode;
        firstNode->next = findNode;
        findNode->prev = firstNode;
        findNode->next = NULL;
        tlbTable.firstNode = findNode;
```

```
    }
}
int main(int argc, char* argv[]){
    FILE *inputFile;
    FILE *physical;
    FILE *PhysicalMemory;
    FILE *Frame;
    int i,j;
    /************************LRU************************/
    inputFile = fopen(argv[1], "r");
    physical = fopen("LRU_Physical.txt", "w");
    PhysicalMemory = fopen("LRU_PhysicalMemory.txt", "w");
    Frame = fopen("LRU_FrameTable.txt", "w");
    while (fscanf(inputFile, "%s\n", addressInput) != EOF){
        address = atoi(addressInput);   // string -> int type cast
        pageNumber = address >> 8; // 8-bit page number
        offset = address & ((1 << 8) - 1); // 8-bit page offset
        int frameNumber = insertNode(pageNumber,0);
        fprintf(physical, "Virtual address: %d Physical address: %d\n",
            256 * pageNumber + offset, 256 * frameNumber + offset);
    }
    /*
     * frame number대로 정리하여 작성합니다.
     */
    struct Node* idxNode = frameTable.firstNode;
    while (idxNode != NULL) {
        int addr = idxNode->pageNumb << 8;
        frameTableResult[idxNode->frameNumb][0] = idxNode->validBit;
        frameTableResult[idxNode->frameNumb][1] = addr;
        idxNode = idxNode->prev;
    }
    for(i =0; i<128; i++){
        fprintf(Frame, "%d %d %d\n", i+1 , frameTableResult[i][0], frame
TableResult[i][1]);
    }
    for (i = 0; i < 128; i++) {
        for (j = 0; j < 256; j++) {
            if (i != 0 || j != 0){
                if (j % 16 == 0){
                    fprintf(PhysicalMemory, "\n");
                }
            }
            fprintf(PhysicalMemory, "%02X ", physicalMemory[i][j]);
        }
    }

    fclose(inputFile);
    fclose(physical);
    fclose(PhysicalMemory);
    fclose(Frame);
    /*
     * tlbTable과 frameTable의 hit율을 출력합니다.
     */
    printf("TLB hit ratio : %d hits out of 1000\n", tlbTable.hit);
```

```
    printf("LRU hit ratio : %d hits out of 1000\n", frameTable.hit + tlb
Table.hit);

    /************************FIFO************************/
    /*
     * 기존에 생성되어 있던 frametable과 tlbTable에 대한 정보를 먼저 초기화 합니다.
     * 이후 사용되었던 배열들 또한 초기화 합니다.
     */
    frameTable.size = 0;
    frameTable.hit = 0;
    tlbTable.size = 0;
    tlbTable.hit = 0;
    frameTable.firstNode = NULL;
    frameTable.lastNode = NULL;
    tlbTable.firstNode = NULL;
    tlbTable.lastNode = NULL;
    for(i = 0; i<frameTableSize; i++){
        for(j = 0; j<pageTableSize; j++){
            physicalMemory[i][j] = 0;
        }
    }
    for(i = 0; i<pageTableSize; i++){
        pageTable[i][0] = 0;
        pageTable[i][1] = 0;
    }
    for(i =0; i<frameTableSize; i++){
        frameTableResult[i][0] = 0;
        frameTableResult[i][1] = 0;
    }
    inputFile = fopen(argv[1], "r");
    physical = fopen("FIFO_Physical.txt", "w");
    PhysicalMemory = fopen("FIFO_PhysicalMemory.txt", "w");
    Frame = fopen("FIFO_FrameTable.txt", "w");
    while (fscanf(inputFile, "%s\n", addressInput) != EOF){
        address = atoi(addressInput);
        pageNumber = address >> 8;
        offset = address & ((1 << 8) - 1);
        int frameNumber = insertNode(pageNumber,1);
        fprintf(physical, "Virtual address: %d Physical address: %d\n", 256
* pageNumber + offset, 256 * frameNumber + offset);
    }
    /*
     * frame number대로 정리하여 작성합니다.
     */
    struct Node* idxNode2 = frameTable.firstNode;
    while (idxNode2 != NULL) {
        int addr = idxNode2->pageNumb << 8;
        frameTableResult[idxNode2->frameNumb][0] = idxNode2->validBit;
        frameTableResult[idxNode2->frameNumb][1] = addr;
        idxNode2 = idxNode2->prev;
    }
    for(i =0; i<128; i++){
        fprintf(Frame, "%d %d %d\n", i+1 , frameTableResult[i][0], frame
TableResult[i][1]);
    }
```

```
    for (i = 0; i < 128; i++) {
        for (j = 0; j < 256; j++) {
            if (i != 0 || j != 0){
                if (j % 16 == 0){
                    fprintf(PhysicalMemory, "\n");
                }
            }
            fprintf(PhysicalMemory, "%02X ", physicalMemory[i][j]);
        }
    }
    fclose(inputFile);
    fclose(physical);
    fclose(PhysicalMemory);
    fclose(Frame);
    // FIFO hit율을 출력하는 부분입니다.
    printf("FIFO hit ratio : %d hits out of 1000\n",frameTable.hit + tlb
Table.hit);
    return 0;
}
```

테스트환경 및 어려웠던 점

- 클라우드 환경
  - testSite : https://c9.io
  - gcc --version : 4.8.4

```
hanjungv:~/workspace $ gcc --version
gcc (Ubuntu 4.8.4-2ubuntu1~14.04.3) 4.8.4
Copyright (C) 2013 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```

- 어려웠던 점
  - 모듈화를 하지 않고 코딩을 하려 했을 때 코드가 많이 복잡해지고 어려워졌었습니다.
  - backingStore.bin 파일의 내용을 어떻게 출력할지에 대해 고민했었습니다.