운영체제

과목명	운영체제
교수명	김철홍
학 과	컴퓨터학부
학 번	20192393
이 름	김현우
제출일	2023.12.04

1 . 제출하는 소스코드 파일 리스트

- Makefile
- simulation.c

2. 작업 설명

가상 주소 길이, 페이지 크기, 물리 메모리 크기, 알고리즘 종류를 입력 받는다.

Input.in 또는 입력 받은 파일에 있는 가상 메모리 값들을 가져와서 가상 메모리 값들로 Page No 값을 구한다.

입력 받은 알고리즘 종류에 따라 Frame No 를 구해 물리 주소와 page fault 유무를 확인한다.

simulation.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <string.h>
#include <time.h>
#include <math.h>

#define MAX_PATH 1024

void makeInputFile(int maxVirtualAddress); // "input.in" file 자동 생성하는 함수
void getVirtualAddress(); // 가상 메모리 값 받는 함수
int calculatePageNo(int num, int startBit, int numBits); // page number 구하는 함수
void optimal(int* frameTable, int frameEntryNum, int virtualAddressLenght, int
pageSize); // optimal 함수
int isPageFault(int* frameTable, int referenceString, int frameEntryNum); //
optimal 에서 pagefault 확인하는 함수
int optimalReplace(int* frameTable, int* timeStamp, int now, int frameEntryNum); //
optimal 에서 frameTable 바꾸는 함수
void fifo(int* frameTable, int frameEntryNum); // fifo 함수
void LRU(int* frameTable, int frameEntryNum); // LRU 함수
void second_chance(int* frameTable, int frameEntryNum); // second_chance 함수
void printNumber(int num, char *number); // 숫자 형식 맞춰주기

int virtualAddressLenght;
int maxVirtualAddress;
int pageSizeNum;
int pageSize; // offset
int physicalMemorySizeNum;
int physicalMemorySize;
int algo;
int inputStyle;
char* input_path;
char number[20];

int frameEntryNum;
int virtualAddress[5000];
```

```c
int referenceString[5000];

int pageNo;
int frameNo;
int physicalAddress;

int frameTableP = 0; // frameTablePointer
int pagefaultCnt = 0; // pagefault 횟수 저장 변수


int main()
{
    printf("A. Simulation 에 사용할 가상주소 길이를 선택하시오 (1. 18bits   2. 19bits
3. 20bits): ");
    scanf("%d", &virtualAddressLenght);

    printf("\nB. Simulation 에 사용할 페이지(프레임)의 크기를 선택하시오 (1. 1KB   2. 2KB
3. 4KB): ");
    scanf("%d", &pageSizeNum);

    printf("\nC. Simulation 에 사용할 물리메모리의 크기를 선택하시오 (1. 32KB   2. 64KB):
");
    scanf("%d", &physicalMemorySizeNum);

    printf("\nD. Simulation 에 적용할 Page Replacement 알고리즘을 선택하시오\n");
    printf("(1. Optimal   2. FIFO   3. LRU   4. Second-Chance): ");
    scanf("%d", &algo);

    printf("\nE. 가상주소 스트링 입력방식을 선택하시오\n");
    printf("(1. input.in 자동 생성   2. 기존 파일 사용): ");
    scanf("%d", &inputStyle);

    switch (virtualAddressLenght)
    {
        case 1:
            maxVirtualAddress = 262143;
            virtualAddressLenght = 18;
            break;

        case 2:
            maxVirtualAddress = 524287;
            virtualAddressLenght = 19;
            break;

        case 3:
            maxVirtualAddress = 1048575;
            virtualAddressLenght = 20;
            break;

        default:
```

```
            break;
    }

    switch (pageSizeNum)
    {
        case 1:
            pageSize = 10; // 1KB
            break;

        case 2:
            pageSize = 11; // 2KB
            break;

        case 3:
            pageSize = 12; // 4KB
            break;

        default:
            break;
    }

    switch (physicalMemorySizeNum)
    {
        case 1:
            physicalMemorySize = 15; // 32KB
            break;

        case 2:
            physicalMemorySize =16; // 64KB
            break;

        default:
            break;
    }

    frameEntryNum = (int)pow(2, physicalMemorySize - pageSize); // frameTable 크기
저장 변수
    int frameTable[frameEntryNum];
    for(int i = 0 ; i < frameEntryNum ; i++)
    {
        frameTable[i] = -1; // 초기화
    }


    switch (inputStyle)
    {
        case 1:
            makeInputFile(maxVirtualAddress);
            input_path = (char*) malloc(sizeof(char) * MAX_PATH);
            strcpy(input_path, "input.in");
            break;
```

```
        case 2:
            getchar();
            printf("\nF. 입력 파일 이름을 입력하시오: ");
            input_path = (char *) malloc(sizeof(char) * MAX_PATH);

            if (fgets(input_path, MAX_PATH, stdin) != NULL)
            {
                size_t len = strlen(input_path);
                if (len > 0 && input_path[len - 1] == '\n')
                {
                    input_path[len - 1] = '\0';
                }
            }
            break;

        default:
            break;
    }


    switch (algo)
    {
        case 1:
            optimal(frameTable, frameEntryNum, virtualAddressLenght, pageSize);
            break;

        case 2:
            fifo(frameTable, frameEntryNum);
            break;

        case 3:
            LRU(frameTable, frameEntryNum);
            break;

        case 4:
            second_chance(frameTable, frameEntryNum);
            break;

        default:
            break;
    }

}

// "input.in" file 자동 생성하는 함수
void makeInputFile(int maxVirtualAddress) {
    // 파일 열기
    FILE *file = fopen("input.in", "w");

    if (file == NULL) {
```

```
        printf("파일을 열 수 없습니다.\n");
    }

    // 난수 초기화
    srand(time(NULL));

    for (int i = 0; i < 5000; i++) {
        int random_value = rand() % (maxVirtualAddress + 1);
        fprintf(file, "%d\n", random_value); // 파일에 값 저장
    }

    // 파일 닫기
    fclose(file);
}

// 가상 메모리 값 받는 함수
void getVirtualAddress() {

    FILE *file = fopen(input_path, "r");

    int count = 0;
    while (fscanf(file, "%d", &virtualAddress[count]) != EOF) {
        count++;
    }

    fclose(file);
}

// page number 구하는 함수
int calculatePageNo(int num, int startBit, int numBits) {
    // 추출할 비트 범위를 마스킹하여 원하는 비트를 추출
    int mask = ((1 << numBits) - 1) << (startBit - 1);
    int result = (num & mask) >> (startBit - 1);

    return result;
}

// physical address 구하는 함수
int cacluatePhysicalAddress(int num, int startBit, int numBits, int pageNo) {
    // 기존 값을 clear
    int mask = ~(((1 << numBits) - 1) << (startBit - 1));
    num &= mask;

    // 새로운 값을 set
    num |= (pageNo << (startBit - 1));

    return num;
}

// 숫자 형식 맞춰주기
```

```c
void printNumber(int num, char *number) {
    char temp[20];
    sprintf(temp, "%d", num); // 숫자를 문자열로 변환

    int length = strlen(temp);
    int comma_count = (length - 1) / 3; // 삽입할 쉼표의 수 계산

    int j = 0;
    for (int i = 0; i < length; i++) {
        number[j++] = temp[i];
        if ((length - i - 1) % 3 == 0 && (length - i - 1) != 0) {
            number[j++] = ','; // 쉼표 삽입
        }
    }
    number[j] = '\0'; // 문자열 끝에 null 문자 추가
}

// optimal 함수
void optimal(int* frameTable, int frameEntryNum, int virtualAddressLenght, int
pageSize) {

    // 파일 열기
    FILE *file = fopen("output.opt", "w");

    if (file == NULL) {
        printf("파일을 열 수 없습니다.\n");
    }

    getVirtualAddress();
    fprintf(file, "%-10s  %-10s  %-10s  %-10s  %-10s  %-10s\n", "No.", "V.A", "Page
No.", "Frame No.", "P.A", "Page Fault\n");


    int pagefaultCnt = 0;
    int maxPageNumber = (int)pow(2, virtualAddressLenght - pageSize);
    int pagefault = 1;
    int timeStamp[frameEntryNum]; // frameTable 에 들어온 순서 저장 변수
    int pageTable[maxPageNumber]; //page -> frameNumber

    for(int i = 0 ; i < 5000 ; i++)
    {
        referenceString[i] = calculatePageNo(virtualAddress[i], pageSize + 1,
virtualAddressLenght - pageSize); // reference string 저장하기
    }

    for(int i = 0; i < frameEntryNum; i++)
    {
        timeStamp[i] = -1; // timestamp 초기화
    }
```

```
    for(int i = 0; i < 5000; i++)
    {
        if(!isPageFault(frameTable, referenceString[i], frameEntryNum)) // page
fault 난 경우
        {
            int replaceNo = optimalReplace(frameTable, timeStamp, i + 1,
frameEntryNum);
            frameTable[replaceNo] = referenceString[i];
            timeStamp[replaceNo] = i;
            frameNo = replaceNo; // frameNo 저장하기
            pageTable[referenceString[i]] = replaceNo;

            pagefaultCnt++;
            pagefault = 1;
        }
        else
        {
            pagefault = 0;
            frameNo = pageTable[referenceString[i]]; // frameNo 저장하기
        }

        physicalAddress = cacluatePhysicalAddress(virtualAddress[i], pageSize + 1,
virtualAddressLenght - pageSize, frameNo);

        fprintf(file, "%-10d  %-10d  %-10d  %-10d  %-10d  ", i + 1,
virtualAddress[i], referenceString[i], frameNo, physicalAddress);

        if(pagefault)
            fprintf(file, "%-10s\n", "F");
        else
            fprintf(file, "%-10s\n", "H");
    }

    printNumber(pagefaultCnt, number);

    fprintf(file, "Total Number of Page Faults: %s\n", number);

    // 파일 닫기
    fclose(file);
}

// optimal 에서 pagefault 확인하는 함수
int isPageFault(int* frameTable, int referenceString, int frameEntryNum) {
    for(int i = 0; i < frameEntryNum; i++)
    {
        if(frameTable[i] == referenceString)
            return 1;
    }
    return 0;
}
```

```
// optimal 에서 frameTable 바꾸는 함수
int optimalReplace(int* frameTable, int* timeStamp, int now, int frameEntryNum) {

    int maxDist = -1;
    int replaceNo = -1;
    int oldStamp = 10000;
    bool check = true;

    for(int i = 0; i < frameEntryNum; i++)
    {
        int j;
        for(j = now; j < 5000; j++)
        {
            if(frameTable[i] == referenceString[j]) // 현 시점 사용하는 pageNo 와
같은게 있을 경우
            {
                if(j > maxDist && check)
                {
                    maxDist = j;
                    replaceNo = i;
                }
                break;
            }
        }
        if(j == 5000) // 끝까지 돌았을 경우 다시는 안 나오는 값인 경우
        {
            if(timeStamp[i] == -1)
                return i;
            if(timeStamp[i] < oldStamp) // 가장 오래된 값 찾기
            {
                oldStamp = timeStamp[i];
                replaceNo = i;
                check = false;
            }
        }
    }
    return replaceNo;
}

// fifo 함수
void fifo(int* frameTable, int frameEntryNum) {

    // 파일 열기
    FILE *file = fopen("output.fifo", "w");

    if (file == NULL) {
        printf("파일을 열 수 없습니다.\n");
    }
```

```
    getVirtualAddress();
    fprintf(file, "%-10s  %-10s  %-10s  %-10s  %-10s  %-10s\n", "No.", "V.A", "Page
No.", "Frame No.", "P.A", "Page Fault\n");

    for(int i = 0 ; i < 5000 ; i ++)
    {
        int pagefault = 1;
        pageNo = calculatePageNo(virtualAddress[i], pageSize + 1,
virtualAddressLenght - pageSize);

        for(int j = 0 ; j < frameEntryNum ; j++)
        {
            if(pageNo == frameTable[j]) // hit 한 경우
            {
                pagefault = 0;
                frameNo = j;
                break;
            }
        }

        if(pagefault) // page fault 난 경우
        {
            frameTable[frameTableP] = pageNo;
            frameNo = frameTableP;
            frameTableP = (frameTableP + 1) % frameEntryNum; // frameTablePointer
값 증가
            pagefaultCnt++;
        }

        physicalAddress = cacluatePhysicalAddress(virtualAddress[i], pageSize + 1,
virtualAddressLenght - pageSize, frameNo);

        fprintf(file, "%-10d  %-10d  %-10d  %-10d  %-10d  ", i + 1,
virtualAddress[i], pageNo, frameNo, physicalAddress);
        if(pagefault)
            fprintf(file, "%-10s\n", "F");
        else
            fprintf(file, "%-10s\n", "H");
    }

    printNumber(pagefaultCnt, number);

    fprintf(file, "Total Number of Page Faults: %s\n", number);

    // 파일 닫기
    fclose(file);
}

// LRU 함수
void LRU(int* frameTable, int frameEntryNum) {
```

```
    // 파일 열기
    FILE *file = fopen("output.lru", "w");

    if (file == NULL) {
        printf("파일을 열 수 없습니다.\n");
    }

    getVirtualAddress();
    fprintf(file, "%-10s  %-10s  %-10s  %-10s  %-10s  %-10s\n", "No.", "V.A", "Page No.", "Frame No.", "P.A", "Page Fault\n");

    int lruarr[frameEntryNum];
    for (int k = 0; k < frameEntryNum; k++)
    {
        lruarr[k] = -1; // 초기화
    }

    for(int i = 0 ; i < 5000 ; i ++)
    {
        int pagefault = 1;
        int isFullFrame = 1;
        pageNo = calculatePageNo(virtualAddress[i], pageSize + 1, virtualAddressLenght - pageSize);

        for(int j = 0 ; j < frameEntryNum ; j++)
        {
            lruarr[j]++;
        }

        for(int j = 0 ; j < frameEntryNum ; j++)
        {
            if(frameTable[j] == -1) // frameTable 이 가득차지 않은 경우
                isFullFrame = 0;
            if(pageNo == frameTable[j])  // hit 한 경우
            {
                pagefault = 0;
                frameNo = j;
                lruarr[j] = 0;
                break;
            }
        }

        if(pagefault) // page fault 난 경우
        {
            if(isFullFrame) // frameTable 이 가득찬 경우
            {
                for(int k = 0 ; k < frameEntryNum ; k++)
                {
                    if(lruarr[k] > lruarr[frameTableP]) // 사용한지 가장 오래된 값 찾기
```

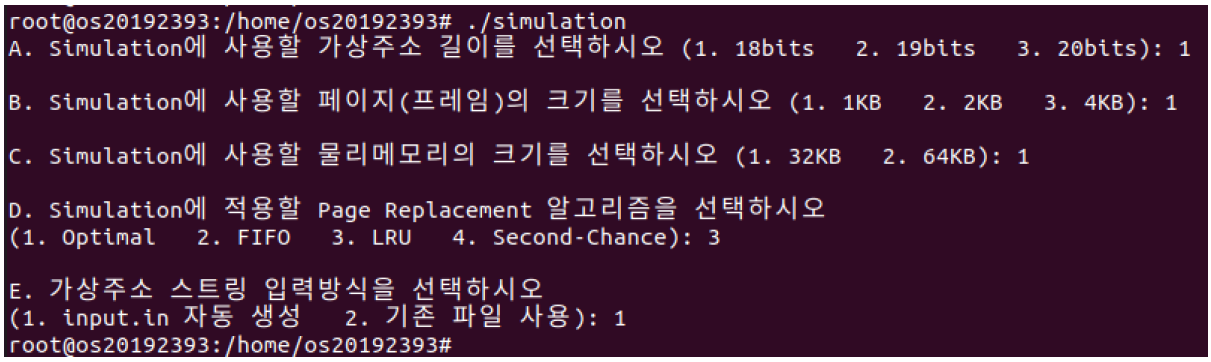
```
                        frameTableP = k;
                }
            }

            frameTable[frameTableP] = pageNo;
            lruarr[frameTableP] = 0; // 새로 들어와서 0 으로 저장하기
            frameNo = frameTableP;
            frameTableP = (frameTableP + 1) % frameEntryNum;
            pagefaultCnt++;
        }

        physicalAddress = cacluatePhysicalAddress(virtualAddress[i], pageSize + 1,
virtualAddressLenght - pageSize, frameNo);

        fprintf(file, "%-10d  %-10d  %-10d  %-10d  %-10d  ", i + 1,
virtualAddress[i], pageNo, frameNo, physicalAddress);
        if(pagefault)
            fprintf(file, "%-10s\n", "F");
        else
            fprintf(file, "%-10s\n", "H");
    }

    printNumber(pagefaultCnt, number);

    fprintf(file, "Total Number of Page Faults: %s\n", number);

    // 파일 닫기
    fclose(file);
}

// second_chance 함수
void second_chance(int* frameTable, int frameEntryNum) {

    // 파일 열기
    FILE *file = fopen("output.sc", "w");

    if (file == NULL) {
        printf("파일을 열 수 없습니다.\n");
    }

    getVirtualAddress();
    fprintf(file, "%-10s  %-10s  %-10s  %-10s  %-10s  %-10s\n", "No.", "V.A", "Page
No.", "Frame No.", "P.A", "Page Fault\n");

    int scarr[frameEntryNum]; // second_chance 확인하는 변수
    for (int k = 0; k < frameEntryNum; k++)
    {
        scarr[k] = 0; // 초기화
    }
```

```
    for (int i = 0 ; i < 5000 ; i++)
    {
        int pagefault = 1;
        int isFullFrame = 1;
        pageNo = calculatePageNo(virtualAddress[i], pageSize + 1,
virtualAddressLenght - pageSize);

        for(int j = 0 ; j < frameEntryNum ; j++)
        {
            if(frameTable[j] == -1) // frameTable 이 가득차지 않은 경우
                isFullFrame = 0;
            if(pageNo == frameTable[j]) // hit 한 경우
            {
                pagefault = 0;
                frameNo = j;
                scarr[j] = 1;
                break;
            }
        }

        if(pagefault) // page fault 난 경우
        {
            while(scarr[frameTableP] == 1) // scarr 값이 1 인 경우
            {
                scarr[frameTableP] = 0; // scarr 값을 0 으로 바꾸기
                frameTableP = (frameTableP + 1) % frameEntryNum;
            }

            frameTable[frameTableP] = pageNo; // scarr 값이 0 인 값이 나온 경우
frameTable 값 바꾸기
            frameNo = frameTableP;
            frameTableP = (frameTableP + 1) % frameEntryNum;
            pagefaultCnt++;

        }

        physicalAddress = cacluatePhysicalAddress(virtualAddress[i], pageSize + 1,
virtualAddressLenght - pageSize, frameNo);

        fprintf(file, "%-10d  %-10d  %-10d  %-10d  %-10d  ", i + 1,
virtualAddress[i], pageNo, frameNo, physicalAddress);
        if(pagefault)
            fprintf(file, "%-10s\n", "F");
        else
            fprintf(file, "%-10s\n", "H");
    }

    printNumber(pagefaultCnt, number);

    fprintf(file, "Total Number of Page Faults: %s\n", number);
```

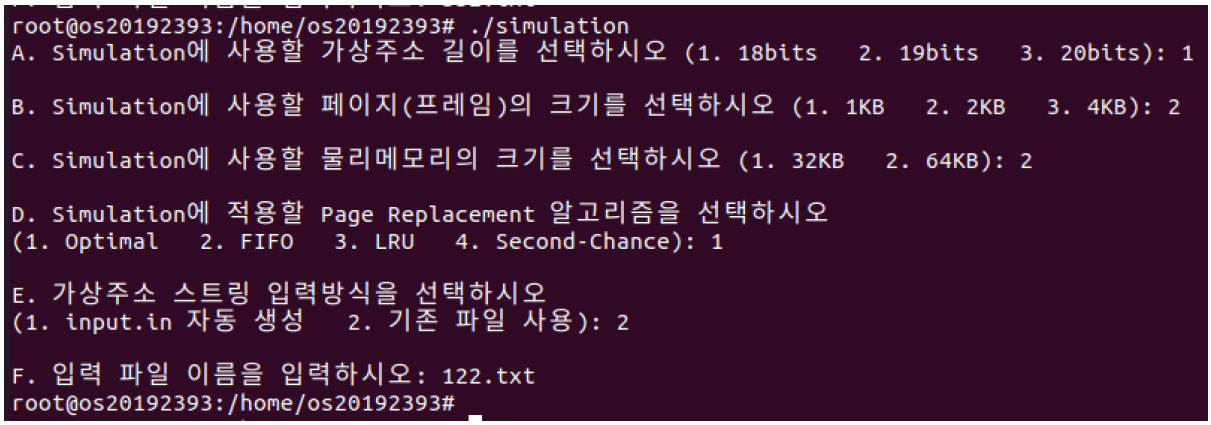
```
    // 파일 닫기
    fclose(file);
}
```

3. 실행 화면 캡쳐

실행 화면 (자동 생성 사용한 경우)



실행 화면 (기존 파일 사용한 경우)



결과 화면

가상 주소 길이 : 1, 페이지 크기 : 2, 물리 메모리 크기 : 2, 알고리즘 : Optimal

```
4977        131270      64          3           6342        H
4978        1458        0           26          54706       F
4979        171161      83          19          40089       H
4980        115047      56          0           359         F
4981        234397      114         8           17309       F
4982        128549      62          2           5669        F
4983        4210        2           14          28786       F
4984        1321        0           26          54569       H
4985        80310       39          1           2486        H
4986        42087       20          1           3175        F
4987        150129      73          25          51825       H
4988        200817      98          3           6257        F
4989        212959      103         22          47071       H
4990        240900      117         12          25860       H
4991        124413      60          28          58877       H
4992        53101       25          27          57197       H
4993        116572      56          0           1884        H
4994        147129      71          7           16057       H
4995        208903      102         12          24583       F
4996        246992      120         24          50384       H
4997        128903      62          2           6023        H
4998        85822       41          23          48958       H
4999        217218      106         24          49282       F
5000        59886       29          16          33262       H
Total Number of Page Faults: 2,047
"output.opt" 5003L, 355107C
```

가상 주소 길이 : 1, 페이지 크기 : 2, 물리 메모리 크기 : 2, 알고리즘 : FIFO

```
4977        131270      64          2           4294        F
4978        1458        0           3           7602        F
4979        171161      83          11          23705       H
4980        115047      56          4           8551        F
4981        234397      114         5           11165       F
4982        128549      62          6           13861       F
4983        4210        2           7           14450       F
4984        1321        0           3           7465        H
4985        80310       39          9           18870       H
4986        42087       20          8           17511       F
4987        150129      73          20          41585       H
4988        200817      98          9           18545       F
4989        212959      103         10          22495       F
4990        240900      117         11          23812       F
4991        124413      60          16          34301       H
4992        53101       25          12          26477       F
4993        116572      56          4           10076       H
4994        147129      71          13          28345       H
4995        208903      102         13          26631       F
4996        246992      120         17          36048       H
4997        128903      62          6           14215       H
4998        85822       41          14          30526       F
4999        217218      106         15          30850       F
5000        59886       29          16          33262       F
Total Number of Page Faults: 3,697
"output.fifo" 5003L, 355107C
```

가상 주소 길이 : 1, 페이지 크기 : 2, 물리 메모리 크기 : 2, 알고리즘 : LRU

```
4977        131270      64          1           2246        F
4978        1458        0           24          50610       F
4979        171161      83          12          25753       H
4980        115047      56          19          39271       F
4981        234397      114         25          52125       F
4982        128549      62          31          65061       F
4983        4210        2           14          28786       F
4984        1321        0           24          50473       H
4985        80310       39          4           8630        F
4986        42087       20          10          21607       F
4987        150129      73          21          43633       H
4988        200817      98          3           6257        F
4989        212959      103         18          38879       F
4990        240900      117         2           5380        F
4991        124413      60          26          54781       H
4992        53101       25          27          57197       F
4993        116572      56          19          40796       H
4994        147129      71          6           14009       F
4995        208903      102         5           10247       F
4996        246992      120         11          23760       F
4997        128903      62          31          65415       H
4998        85822       41          22          46910       F
4999        217218      106         13          26754       F
5000        59886       29          29          59886       F
Total Number of Page Faults: 3,686
"output.lru" 5003L, 355107C
```

가상 주소 길이 : 1, 페이지 크기 : 2, 물리 메모리 크기 : 2, 알고리즘 : Second_chance

```
4977        131270      64          0           198         F
4978        1458        0           3           7602        F
4979        171161      83          1           3225        H
4980        115047      56          4           8551        F
4981        234397      114         5           11165       F
4982        128549      62          6           13861       F
4983        4210        2           7           14450       F
4984        1321        0           3           7465        H
4985        80310       39          9           18870       F
4986        42087       20          11          23655       F
4987        150129      73          14          29297       H
4988        200817      98          15          30833       F
4989        212959      103         16          34783       F
4990        240900      117         17          36100       F
4991        124413      60          10          22013       H
4992        53101       25          18          38765       F
4993        116572      56          4           10076       H
4994        147129      71          19          40633       F
4995        208903      102         20          40967       F
4996        246992      120         22          46288       F
4997        128903      62          6           14215       H
4998        85822       41          23          48958       F
4999        217218      106         25          51330       F
5000        59886       29          26          53742       F
Total Number of Page Faults: 3,667
"output.sc" 5003L, 355107C
```

가상 주소 길이 : 1, 페이지 크기 : 3, 물리 메모리 크기 : 2, 알고리즘 : Optimal

```
4980        221913      54          10          41689       H
4981        213654      52          10          41622       F
4982        245435      59          6           28347       H
4983        128591      31          1           5711        F
4984        7733        1           8           36405       F
4985        217544      53          5           20936       H
4986        182441      44          5           22697       F
4987        103691      25          2           9483        F
4988        192282      46          0           3866        F
4989        234400      57          15          62368       H
4990        137846      33          7           31350       H
4991        242992      59          6           25904       H
4992        122999      30          6           24695       F
4993        24060       5           15          65020       F
4994        70697       17          14          58409       F
4995        104707      25          2           10499       H
4996        49905       12          3           13041       H
4997        256051      62          13          55347       H
4998        81904       19          13          57328       F
4999        61433       14          3           16377       F
5000        259437      63          11          46445       F
Total Number of Page Faults: 2,208
"output.opt" 5003L, 355107C
```

4. 실행 결과 분석 내용

가상 주소 길이, 페이지 크기, 물리 페이지 크기를 동일하게 하고 동일한 txt 파일에 대해 4 개의 알고리즘을 실행한 결과 fault 의 개수가
Optimal : 2047
FIFO: 3697
LRU : 3686
Second_chance : 3667

FIFO < LRU < Optimal 순서대로 이론에서 말한 값과 같게 optimal 이 fault 개수가 가장 작게 나오는 것을 확인할 수 있다.
하지만 이론과 다르게 LRU < FIFO 가 되는 경우가 발생하는데 해당 경우 input.in 을 만들어서 사용할 경우 지역성으로 인해 위와 같은 경우가 발생할 수 있다.

Optimal 은 현 시점에서 가장 오랫동안 사용되지 않을 것 같은 페이지를 새로운 페이지와 교체한다. 가장 이상적인 방법이지만, reference string 의 순서를 아는 것은 불가능하므로 완전한 구현은 불가능하다.

FIFO 은 프레임에 들어온 페이지 순서대로 새로운 페이지와 교체된다. FIFO queue 를 이용하여 간단하게 구현할 수 있으나 성능이 reference string 순서에 영향을 받고, 프레임 개수가 늘어나도 page fault 횟수가 늘어나는 경우가 존재한다. (Belady's Anomaly)

LRU 은 가장 많이 사용되는 방법으로 현 시점에서 가장 오랫동안 사용되지 않은 페이지를 새로운 페이지와 교체한다. 프레임마다 counter 변수를 두거나 stack 으로 구현할 수 있지만 전자의 경우 page fault 시마다 카운터 값이 가장 큰 프레임 탐색에 대한 오버헤드가, 후자의 경우 업데이트 비용에 대한 오버헤드 가 발생한다.

Second Chance 은 FIFO 방식에 reference bit 를 두어 page fault 시 현재 교체 대상 페이지가 최근에 참조되었으면 한 번의 기회를 더 주는 방식이다. 현재 next victim 이 가리키고 있는 페이지의 reference bit 의 값이 1 이면 0 으로 변경 후 다음 페이지를 확인하며, 0 일 경우 새로운 페이지와 교체한다.