

# 2025 운영체제PBL 기말과제 보고서

직관적인 페이지 교체 알고리즘 이해를 위한 실습 프로그램

발표자 지승민

1조:지승민, 옥귀동, 이한얼, 박준하

CONTENTS

# 소개 목차

---

01

제작 배경

02

목표

03

기능설명 및 시연

04

코드설명

05

역할 구분

---

# 제작 배경

---

## 제작 배경

- 운영체제(OS) 과목은 직접 실습할 수 있는 환경이 부족해 이론 중심으로 진행되어, 페이지 교체 알고리즘 같은 개념을 직관적으로 이해하기 어려움.
- 단순한 이론 검증을 넘어, 실제 난수를 기반으로 히트율을 분석해보고자 직접 워크로드를 생성함.

## 제작 목표

- 접근 패턴별 부재율 차이를 시각적으로 확인할 수 있는 시뮬레이터를 개발하여 개념을 직관적으로 이해하기 위해 돕고, 추후 학생들도 직접 실험을 통해 운영체제 개념을 체감하고 학습 효과를 높이고자 함.
-

# 프로그램 환경 구축

## 실습하기 쉬운 C언어 환경

- 학생들이 난수 생성 등 반복 연산이 많은 시뮬레이션을 빠르고 원활하게 실행할 수 있도록, 즉각적이고 친숙한 C언어 기반 실습 환경을 구축
- 효율적인 메모리 관리와 저수준 시스템 호출을 지원하는 C언어 환경을 통해, 실습 과정의 정확성과 실행 성능을 극대화

## 사용자 인터페이스를 통한 알고리즘 선택

- 사용자가 프로그램 실행 시 간단한 숫자 입력으로 원하는 페이지 교체 알고리즘(LRU, FIFO, CLOCK, MRU)을 즉시 선택할 수 있어 학습과 실습 효율성 증진
- 선택한 알고리즘에 따라 1-50개의 버퍼와, 그에 따른 히트율이 txt 에 출력되어 사용자가 다양한 알고리즘의 성능 차이를 빠르게 비교 가능

# 기능설명


## 난수 생성

- 다양한 접근 패턴(순차적, 랜덤, 가우시안 분포, 80:20)을 기반으로 페이지 참조 시퀀스를 자동 생성.
- 생성된 시퀀스는 텍스트 파일(.txt)로 저장되어 교체 알고리즘의 입력으로 사용
- 숫자 100,000개의 난수

## 활용목적

- 다양한 메모리 접근 시나리오를 구성하여, 교체 알고리즘의 성능 차이를 체감할 수 있도록 지원

워크로드 파일 생성 완료: random.txt, 8020.txt, sequential.txt, gaussian.txt

os >  gaussian.txt

```
1 14
2 18
3 24
4 30
5 37
6 26
7 33
8 26
9 7
10 24
11 25
12 14
13 16
14 20
15 20
16 31
17 28
18 35
19 36
20 23
21 22
22 26
23 17
24 24
25 22
26 36
27 25
28 18
29 31
30 26
31 24
32 23
33 29
34 19
35 27
36 27
```

# 기능설명

## 페이지 교체 알고리즘

- LRU, FIFO, CLOCK, MRU 알고리즘을 구현하여 입력된 시퀀스에 대한 페이지 교체 시뮬레이션
- 각 알고리즘에 대해 버퍼 개수와 그에 따른 히트율을 출력
- 결과를 .txt파일로 저장해서 excel를 이용해 별도의 그래프로 가시화

## 활용 목적

- 알고리즘별 동작 원리와 성능 차이를 직접 실습하며 이해도 향상

```
root@DESKTOP-D00RFPU:~/OS_test/os# ./a.out
==== 페이지 접근 시퀀스 선택 ====
1. 지역성 없는 랜덤 워크로드
2. 80-20 워크로드
3. 순차적 워크로드
4. 가우시안 분포 워크로드
선택: 1

==== 알고리즘 선택 ====
1. LRU
2. FIFO
3. CLOCK
4. MRU
선택: 2
결과가 'FIFO_random_results.txt'에 저장되었습니다.
```

```
ostep-code-master > os_project > hit_rate_results > ⓧ CLOCK_8020_results.txt
1 # frameSize hitRate(%)
2 1 6.35
3 2 12.56
4 3 18.72
5 4 24.52
6 5 30.34
7 6 35.92
8 7 41.35
9 8 46.34
10 9 51.00
11 10 55.40
12 11 59.50
13 12 63.25
14 13 66.57
15 14 69.70
16 15 72.40
17 16 74.72
18 17 76.89
19 18 78.75
20 19 80.25
21 20 81.74
22 21 82.92
```

# 시연영상

---

# 코드 설명

```
#define PAGE_COUNT 50      // 작업 집합 페이지 수 (0~49)
#define SEQ_LENGTH 100000  // 한 워크로드당 참조 수열 길이
#define M_PI 3.14159265358979323846
```

```
void gen_random(const char* filename, unsigned int seed) {
    srand(seed);
    FILE* fp = fopen(filename, "w");
    if (!fp) {
        perror(filename);
        return;
    }
    for (int i = 0; i < SEQ_LENGTH; i++) {
        int p = rand() % PAGE_COUNT;
        fprintf(fp, "%d\n", p);
    }
    fclose(fp);
}
```

<gen\_random-지역성 없는 순수 랜덤 워크로드>

- int p= rand() %PAGE\_COUNT;  
0~49 사이의 숫자를 무작위 입력  
무작위로 페이지 참조 시퀀스 생성(100,000)



# 코드 설명

```
// 균등 난수 [0,1)
static double uniform_rand() {
    return rand() / (RAND_MAX + 1.0);
}
```

```
void gen_8020(const char* filename, unsigned int seed) {
    srand(seed);
    FILE* fp = fopen(filename, "w");
    if (!fp) {
        perror(filename);
        return;
    }
    int hot_n = PAGE_COUNT / 5;
    if (hot_n < 1) hot_n = 1;
    for (int i = 0; i < SEQ_LENGTH; i++) {
        double r = uniform_rand();
        int p;
        if (r < 0.8) {
            p = rand() % hot_n;
        } else {
            p = hot_n + (rand() % (PAGE_COUNT - hot_n));
        }
        fprintf(fp, "%d\n", p);
    }
    fclose(fp);
}
```

## <gen\_8020- 80-20 워크로드>

- 80-20 워크로드는 실제 컴퓨터의 메모리 접근 패턴을 단순화해 모방한 것
- 80% 확률로 0~9 중 하나를, 20% 확률로 10~49 중 하나를 선택해서 입력함

```
double r = uniform_rand(); // 0-1 미만의 난수 생성
int p;
if (r < 0.8) { //80%로 0-9 선택
    p = rand() % hot_n;
} else { // 20%로 10-49선택
    p = hot_n + (rand() % (PAGE_COUNT - hot_n));
}
```

# 코드 설명

```
// 3. 순차적 워크로드
void gen_sequential(const char* filename) {
    FILE* fp = fopen(filename, "w");
    if (!fp) {
        perror(filename);
        return;
    }
    for (int i = 0; i < SEQ_LENGTH; i++) {
        int p = i % PAGE_COUNT;
        fprintf(fp, "%d\n", p);
    }
    fclose(fp);
}
```

## <gen\_sequential-순차적워크로드>

- for (int i = 0; i < SEQ\_LENGTH; i++) { //100.000번  
int p = i % PAGE\_COUNT; // 0~49 사이 숫자를 순서대로 반복  
fprintf(fp, "%d\n", p); // 해당 숫자를 파일에 기록  
}

# 코드 설명

```
// 4. 가우시안 분포 워크로드
void gen_gaussian(const char* filename, unsigned int seed) {
    srand(seed);
    FILE* fp = fopen(filename, "w");
    if (!fp) {
        perror(filename);
        return;
    }
    double mu = (PAGE_COUNT - 1) / 2.0;
    double sigma = PAGE_COUNT / 6.0;
    for (int i = 0; i < SEQ_LENGTH; i++) {
        // Box-Muller 변환
        double u1 = uniform_rand();
        double u2 = uniform_rand();
        double z0 = sqrt(-2.0 * log(u1)) * cos(2.0 * M_PI * u2);
        int p = (int)round(mu + sigma * z0);
        if (p < 0) p = 0;
        if (p >= PAGE_COUNT) p = PAGE_COUNT - 1;
        fprintf(fp, "%d\n", p);
    }
    fclose(fp);
}
```

<gen\_gaussian-가우시안 분포 워크로드>

- 평균 24.5, 표준편차 약 8.3인 가우시안 분포를 이용해 0~49 사이 숫자 생성.
  - 중앙(20~30)에 접근이 많고, 양쪽 끝은 적게 접근
  - 실제 프로그램의 메모리 접근 패턴(특정 영역 집중 접근)을 흉내
- 
- `double z0 = sqrt(-2.0 * log(u1)) * cos(2.0 * M_PI * u2);` // 평균 0, 표준편차 1 정규분포를 통해 생성

# 코드 설명

---

```
int main() {  
    // 파일 단위 생성  
    gen_random("random.txt", 2025);  
    gen_8020("8020.txt", 2025);  
    gen_sequential("sequential.txt");  
    gen_gaussian("gaussian.txt", 42);  
    printf("워크로드 파일 생성 완료: random.txt, 8020.txt, sequential.txt, gaussian.txt\n");  
    return 0;  
}
```

<main()>

- 난수 생성 시드(seed) : 프로그램 여러번 실행해도 같은 난수 결과가 나오도록 고정하는 숫자 (ex 2025, 42)
- 다양한 난수를 원하면 숫자를 수정

# 코드 설명

```
for (int i = 0; i < frameSize; i++) {  
    if (frames[i].page == page) {  
        frames[i].lastUsed = time++;  
        h++;  
        found = 1;  
        break;  
    }  
}
```

```
if (!found) {  
    int lruIdx = 0;  
    for (int i = 1; i < frameSize; i++) {  
        if (frames[i].lastUsed < frames[lruIdx].lastUsed) {  
            lruIdx = i;  
        }  
    }  
    frames[lruIdx].page = page;  
    frames[lruIdx].lastUsed = time++;  
    f++;  
}  
fclose(fp);  
*hit = h;  
*fault = f;
```

<simulateLRU\_counts- 가장 오래 전에 사용된 페이지를 교체 >

- if (frames[i].page == page){~~  
히트 발생 → lastUsed 업데이트, hit 후 반복 종료

- if (!found)  
if (frames[i].lastUsed < frames[lruIdx].lastUsed)  
~~~~  
hit가 아니라면 lastused에서 가장 작은 값(오래된)을 교체  
넣은 페이지의 사용 시간을 지금 시간으로 업데이트  
부재 수 증가

외부에서도 쓸 수 있게 결과 반환

# 코드 설명

```
}  
while (fscanf(fp, "%d", &page) == 1) {  
    int found = 0;  
    for (int i = 0; i < frameSize; i++) {  
        if (frames[i].page == page) {  
            h++;  
            found = 1;  
            break;  
        }  
    }  
    if (!found) {  
        frames[next].page = page;  
        next = (next + 1) % frameSize;  
        f++;  
    }  
}
```

<simulateFIFO\_counts- 선입선출 >

- 파일에서 페이지 번호를 하나씩 읽음
- 히트 발생, 반복문 종료

next- 가장 먼저 들어온 숫자 가르킴

- frames[next] = page;  
next = (next + 1) % frameSize; (원형 큐)  
프레임안에 새로운 숫자를 덮고  
next를 그 다음 오래된 숫자를 가르킴



# 코드 설명

```
frames[i].refBit = 0;
}
while (fscanf(fp, "%d", &page) == 1) {
    int found = 0;
    for (int i = 0; i < frameSize; i++) {
        if (frames[i].page == page) {
            frames[i].refBit = 1;
            h++;
            found = 1;
            break;
        }
    }
```

```
if (!found) {
    while (frames[hand].refBit == 1) {
        frames[hand].refBit = 0;
        hand = (hand + 1) % frameSize;
    }
    frames[hand].page = page;
    frames[hand].refBit = 1;
    hand = (hand + 1) % frameSize;
    f++;
}
```

<simulateClock\_counts- 참조 비트가 0인 페이지를 교체 >

- 히트 발생 → refBit(참조비트) 1 업데이트 후 반복 종료

- while (frames[hand].refBit == 1) {~~~  
참조비트가 1이면 0으로 바꾸고 검사  
참조비트가 0이면 새로운 숫자를 덮어쓰고 참조비트 1로 설정

# 코드 설명

```
}
if (!found) {
    int mruIdx = 0;
    for (int i = 1; i < frameSize; i++) {
        if (frames[i].lastUsed > frames[mruIdx].lastUsed) {
            mruIdx = i;
        }
    }
    for (int i = 0; i < frameSize; i++) {
        if (frames[i].page == -1) {
            mruIdx = i;
            break;
        }
    }
    frames[mruIdx].page = page;
    frames[mruIdx].lastUsed = time++;
    f++;
}
```

<simulateMRU\_counts- 가장 최근에 사용된 페이지를 교체 >

- 히트 발생 → lastUsed 업데이트 후 반복 종료
- if (frames[i].lastUsed > frames[mruIdx].lastUsed) {~~  
LRU와 기호 반대
- if (frames[i].page == -1){ ~~  
LRU는 -1값을 빈자리로 인식해 자동으로 처리되지만  
MRU는 알고리즘이 부등호가 반대라서 처리X  
→ 별도로 빈자리(-1)를 찾는 코드를 적용



# 코드 설명

```
int main() {
    int algoChoice, fileChoice;

    const char* filenames[] = { "random.txt", "8020.txt", "sequential.txt", "gaussian.txt" };
    const char* algoNames[] = { "LRU", "FIFO", "CLOCK", "MRU" };
    const char* fileStems[] = { "random", "8020", "sequential", "gaussian" };

    printf("==== 페이지 접근 시퀀스 선택 ==== \n");
    printf("1. 지역성 없는 랜덤 워크로드 \n 2. 80-20 워크로드 \n 3. 순차적 워크로드 \n 4. 가우시안 분포 워크로드 \n 선택: ");
    scanf("%d", &fileChoice);
    if (fileChoice < 1 || fileChoice > 4) {
        printf("잘못된 시퀀스 선택. \n");
        return 1;
    }

    printf("\n==== 알고리즘 선택 ==== \n");
    printf("1. LRU \n 2. FIFO \n 3. CLOCK \n 4. MRU \n 선택: ");
    scanf("%d", &algoChoice);
    if (algoChoice < 1 || algoChoice > 4) {
        printf("잘못된 알고리즘 선택. \n");
        return 1;
    }
}
```

```
for (int frameSize = 1; frameSize <= PAGESET_SIZE; frameSize++) {
    int hit = 0, fault = 0;
    switch (algoChoice) {
        case 1:
            simulateLRU_counts(filenames[fileChoice-1], frameSize, &hit, &fault);
            break;
        case 2:
            simulateFIFO_counts(filenames[fileChoice-1], frameSize, &hit, &fault);
            break;
        case 3:
            simulateClock_counts(filenames[fileChoice-1], frameSize, &hit, &fault);
            break;
        case 4:
            simulateMRU_counts(filenames[fileChoice-1], frameSize, &hit, &fault);
            break;
    }
    double rate = (hit + fault) ? (double)hit / (hit + fault) * 100.0 : 0.0;
    fprintf(outFp, "%d %.2f \n", frameSize, rate);
}

fclose(outFp);
printf("결과가 '%s'에 저장되었습니다. \n", outfile);
return 0;
}
```

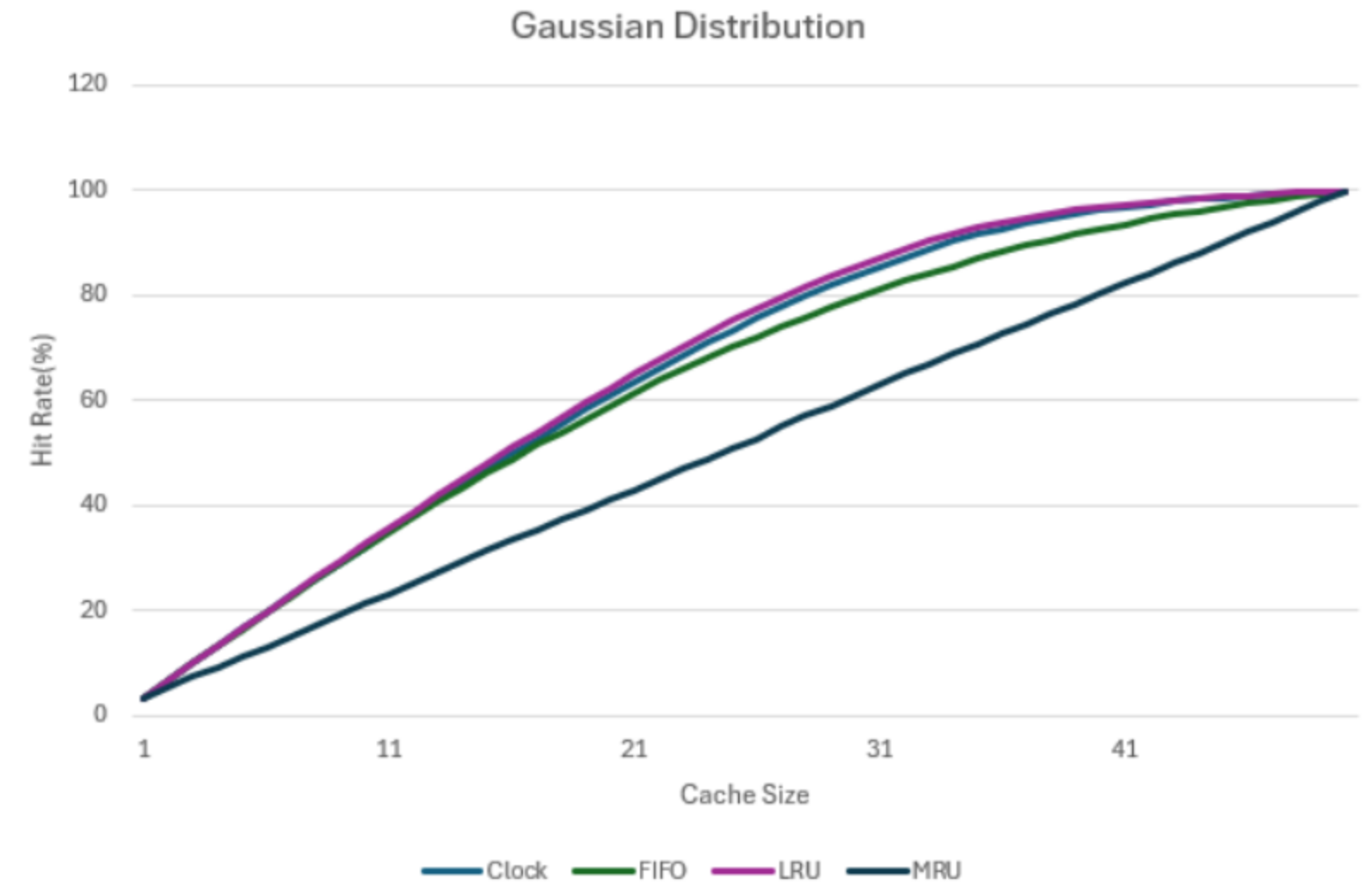
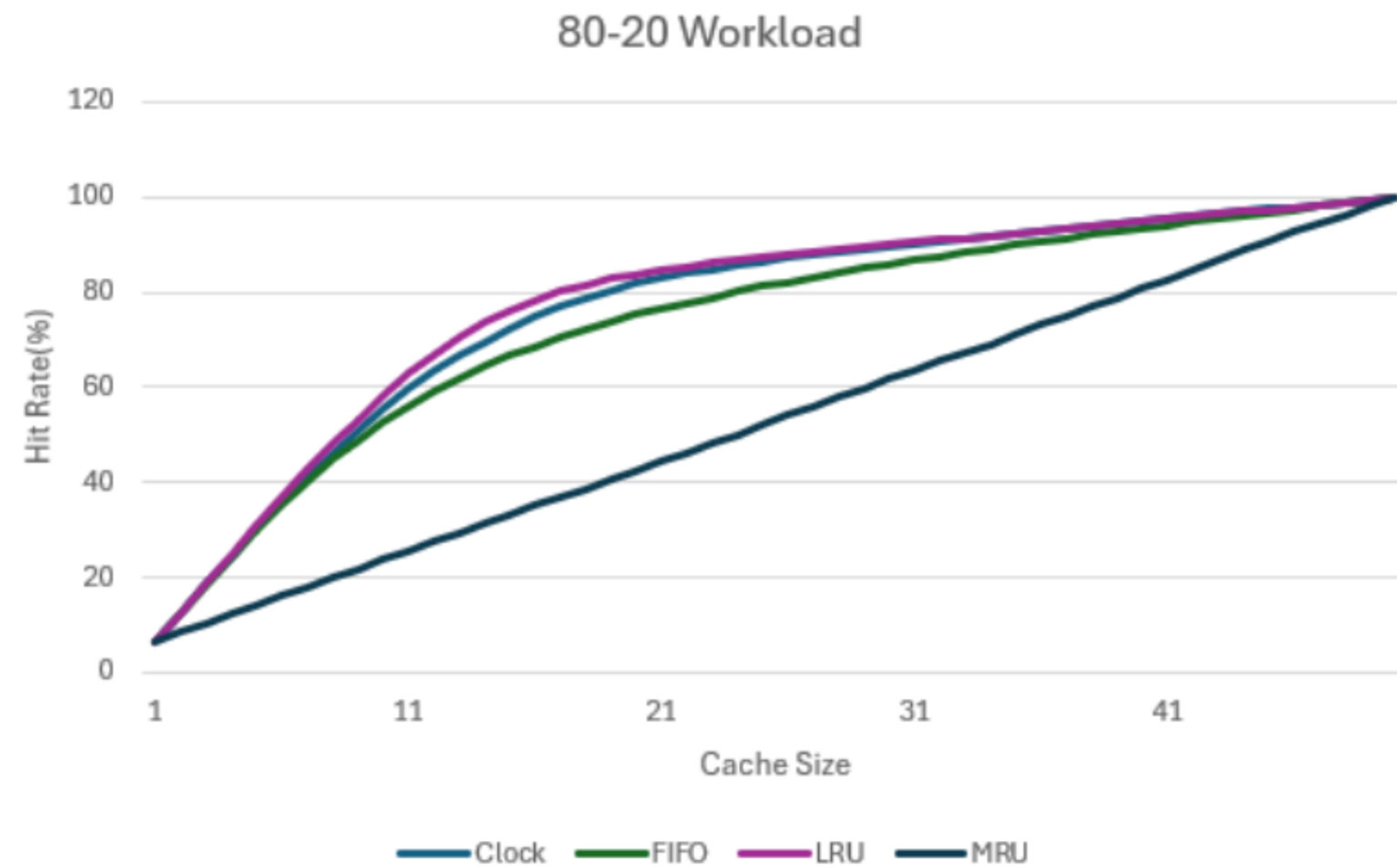
main()

- 난수 생성을 통해 만들어진 txt 파일을 사용자가 선택 할 수 있도록 적용
- 1~4 숫자를 넘어가면 "잘못된 시퀀스 선택" 출력

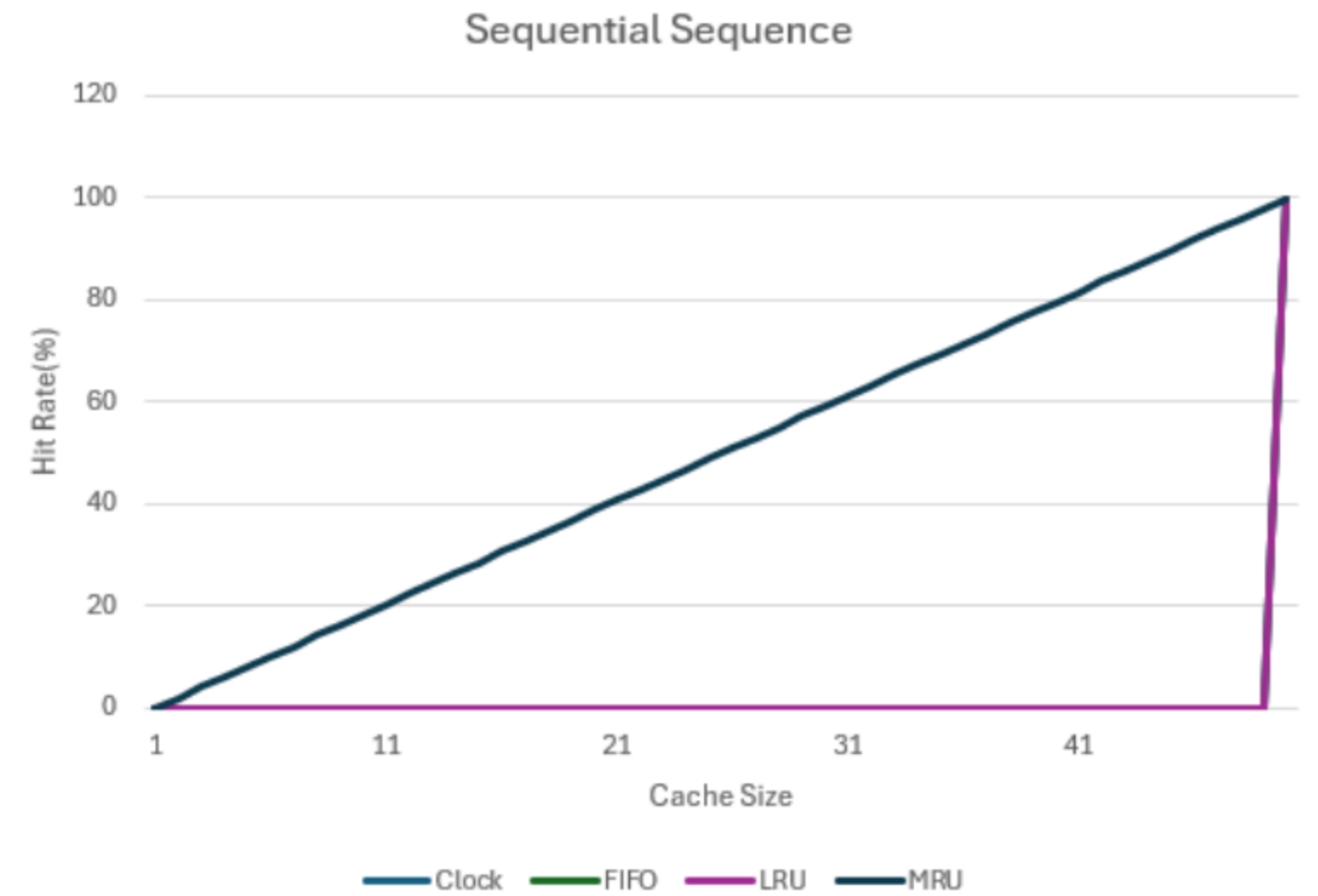
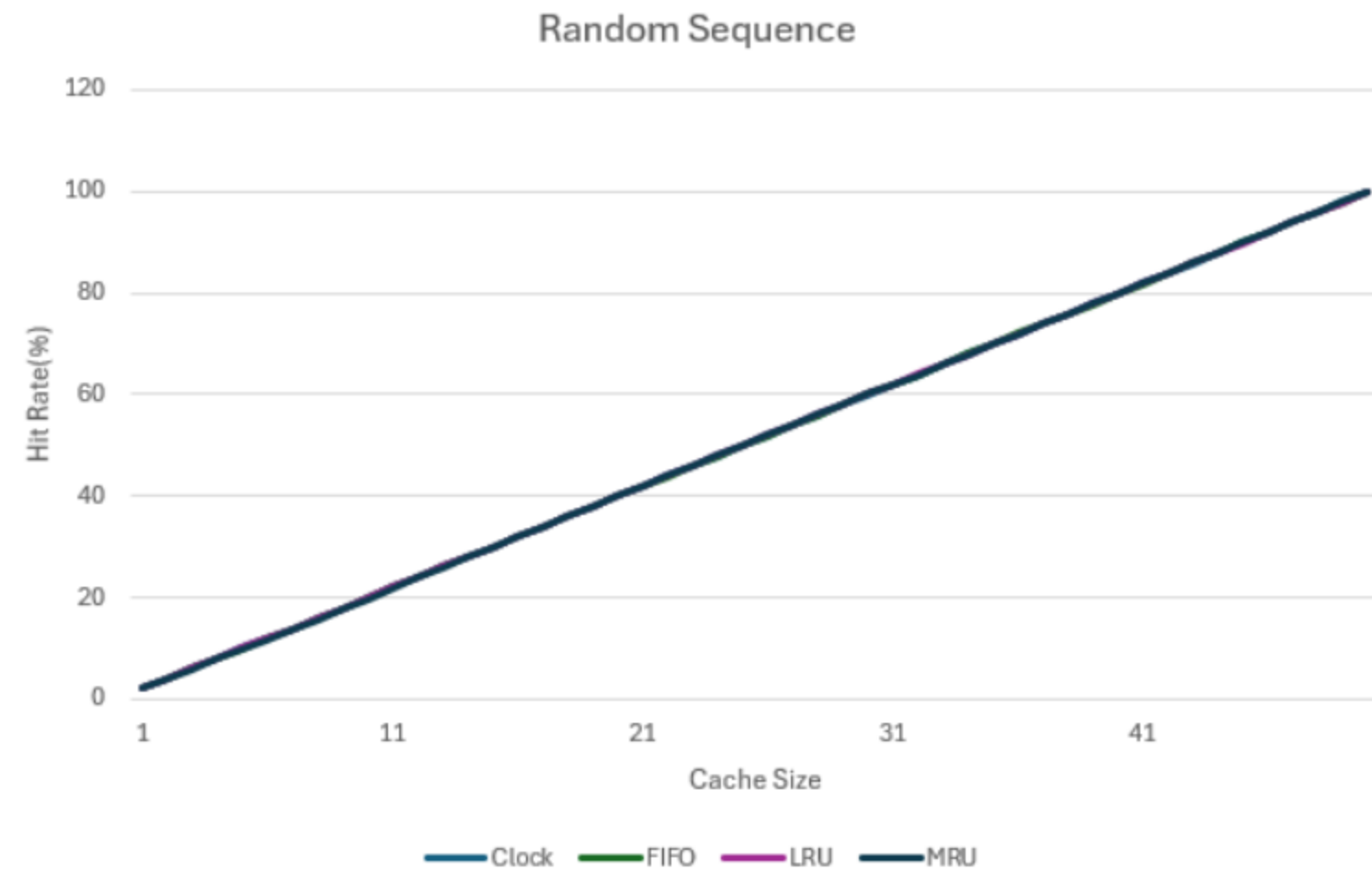
main()

- 페이지 교체 알고리즘( LRU, FIFO, Clock, MRU)를 선택하여 1-50개의 버퍼를 생성하고 그에 따른 히트율을 result.txt 저장

# 결과 그래프



# 결과 그래프



# 역할 분담



지승민

PPT 제작 및 발표  
LRU 작성

옥귀동

난수 생성코드 작성  
아이디어 기획

이한얼

FIFO작성  
그래프 작성  
코드 종합

박준하

MRU 작성  
CLOCK 작성

# 개선점



## POINT 01

### 웹 기반 시각화 기획

가독성 향상을 위해 다음 기회에 웹으로 작성하여 접근성 강화



## POINT 02

### 실행 과정 애니메이션 구현

단계별 애니메이션으로 동작 흐름도 가시화



## POINT 03

### 난수 입력 지원

정해진 텍스트 파일 이외에 사용자가 직접 입력하여 실습 편의성 향상



## POINT 04

### 자동으로 통계, 비교기능 추가

CSV 파일을 통해 자동으로 표를 만들 수 있게 개선

---

**THANK YOU**

