운영체제 과제#4

20212908 이진

과제 채점 시, 컴파일 옵션에 '-lm'을 추가해서 컴파일해주시기 바랍니다!
math.h 헤더파일을 사용하고 있어서 해당 옵션이 필요합니다.
감사합니다:)

[구현과정 및 소스코드 설명]

0. 기본 입출력 구현

```
int main() {
	userInput();	// 사용자 입력 처리
	setOpFileName();	// 출력 파일 이름 설정
	memset(pageList, -1, sizeof(pageList)); //페이지 리스트 초기화
	readData();
	writeData();	// 알고리즘 수행
}
```

userInput() 함수에서 가상 주소 길이, 페이지 크기, 물리 메모리 크기, 적용할 알고리즘, 가상주소 스트링 입력 방식을 입력 받는다. 그 후, 적용할 알고리즘에 따라 출력 파일명을 다르게 설정해주는 setOpFileName() 함수를 호출해주었다. 그 후, pageList 를 -1 로 초기화해주었다. pageList[i] = j 는 i 번째 프레임에 j 번 페이지가 저장되어있다는 의미이다. 그 후 readData() 함수에서 inputFile 의 값을 차례로 읽으면서 inputList 배열에 값을 저장해준다. 그 후 writeData()에서 inputList 배열을 활용하여 적용할 각각의 알고리즘을 수행한다. 해당 함수가 끝나면 프로그램이 종료된다.

```
	ipFileName = malloc(256 * sizeof(char));
	if (ip == 1) {	//input.in 자동 생성
		strcpy(ipFileName, "input.in");
		generateRandomInput(vaLength);
	} else if (ip == 2) {	//기존 파일 사용
		printf("\nF. 입력 파일 이름을 입력하시오: ");
		scanf("%s", ipFileName);
	}
```

userInput() 함수의 일부분이다. 가상주소 스트링 입력 방식이 1 번이면 generateRandomInput() 함수를 호출하여 'input.in' 파일을 자동으로 생성해준다. 2 번이면 입력 파일 이름을 사용자에게 입력받아 저장한다.

```
void generateRandomInput(int vaLength) {
	FILE *inputFile = fopen(ipFileName, "w");
```

```
    if (inputFile == NULL) {
        printf("input.in 파일을 열 수 없습니다\n");
        return;
    }
    unsigned long maxValue = pow(2, vaLength) - 1;    // 최대값
    srand(time(NULL));
    for (int i = 0; i < maxInputLength; i++) {
        unsigned long va = rand() % (maxValue + 1);        // 0~최대값 사이 랜덤 숫자 생성
        fprintf(inputFile, "%lu\n", va);
    }
    fclose(inputFile);
}
```

vaLength 는 사용자에게 입력받은 가상주소 길이이다. 만약 vaLength 가 18bits 라면 가상주소 10 진수가 가질 수 있는 최댓값은 2^18-1 이다. 해당 값을 maxValue 에 저장해주었다. 그 후, srand(time(NULL)), rand()을 사용하여 랜덤숫자를 추출하였다. 0 ~ maxValue 사이 값으로 만들기 위해 rand() % (maxValue + 1) 연산을 하였다.

```
void setOpFileName() {
    opFileName = malloc(256 * sizeof(char));
    switch(prAlgorithm) {
        case 1:
            strcpy(opFileName, "output.opt");
            break;
        case 2:
            strcpy(opFileName, "output.fifo");
            break;
        case 3:
            strcpy(opFileName, "output.lru");
            break;
        case 4:
            strcpy(opFileName, "output.sc");
            break;
    }
}
```

setOpFileName()은 출력 파일명을 명시해주는 함수이다. 사용자가 설정한 page replacement 알고리즘마다 출력 파일명이 달라야하기 때문이다. opFileName 전역변수를 동적 할당해준 후 prAlgorithm 값마다 저장되는 값이 다르게 구현하였다.

```
void readData() {
	FILE *inputFile = fopen(ipFileName, "r");
	if (inputFile == NULL) {
		printf("파일을 열 수 없습니다\n");
		return;
	}
	// 파일에서 숫자 읽어서 inputList 에 저장하기
	inputList = (int *)malloc(maxInputLength * sizeof(int));
	for (int i = 0; i < maxInputLength; i++) {
		fscanf(inputFile, "%d", &inputList[i]);
	}
	fclose(inputFile);
	free(ipFileName);
}
```

readData() 함수에서는 ipFileName 에 해당하는 파일을 읽기모드로 열어준 다음 inputList 에 값을 저장해준다. 그러기 위해 inputList 를 동적할당해주었고, 값을 모두 저장한 이후에는 파일 포인터를 해제해주고, 입력 파일명을 동적으로 저장해두었던 ipFileName 변수도 해제해준다.

```
void writeData() {
	FILE *outputFile = fopen(opFileName, "w");
	if (outputFile == NULL) {
		printf("파일을 열 수 없습니다\n");
		return;
	}
	fprintf(outputFile, "%-10s%-14s%-14s%-14s%-14s%-10s\n","No.", "V.A.", "Page No.", "Frame No.", "P.A.",
"Page Fault");
	int VA = 0, PageNo = 0, FrameNo = 0, PA = 0;
	char Fault = 'F';
	for (int i = 0; i < maxInputLength; i++) {
		VA = inputList[i];
		PageNo = VA / (1024 * pageSize);
		FrameNo = simulate(PageNo, i);
		int offset = VA % (1024 * pageSize);
		PA = (FrameNo * (1024 * pageSize)) + offset;
		if (faultFlag == true) Fault = 'F';
		else Fault = 'H';
		fprintf(outputFile, "%-10d%-14d%-14d%-14d%-14d%-10c\n", i + 1, VA, PageNo, FrameNo, PA, Fault);
	}
```

```
    outputTotalFaults();
    fprintf(outputFile, "Total Number of Page Faults: %s", totalFaults);

    // 동적 메모리 및 파일 포인터 해제
    fclose(outputFile);
    free(inputList);
    free(opFileName);
    if (prAlgorithm == 3) free(lruHead);
    if (prAlgorithm == 4) free(scHead);
}
```

outputFile 에 출력해야하므로, 쓰기모드로 파일을 열고, 기본 틀을 파일에 써준다. 그 후, maxInputLength 만큼 반복문을 돌면서 inputList 에 저장해두었던 가상주소들을 몇 번째 프레임에 할당해줄 것인지 명시해준다. 반복문이 끝나면 outputTotalFaults() 함수를 통해 page faults 개수 세어놨던 것을 천 단위로 쉼표가 찍히도록 포맷팅 해준다. 포맷팅해준 문자열도 출력해준 후, 각종 동적 메모리와 파일포인터를 해제하면 함수가 끝난다.

VA 변수는 가상주소로 readData()에서 받아와 저장한 inputList 의 값을 의미한다. PageNo 변수는 페이지 번호로, 가상주소/페이지크기로 구할 수 있다. 페이지크기를 의미하는 pageSize 의 단위가 KB 이므로 1024 를 곱해서 B 로 단위를 맞춰준다. FrameNo 변수는 프레임 번호로, 적용할 페이지 교체 알고리즘을 simulate 한 후의 반환값으로 얻을 수 있다. Simulate 함수는 페이지번호와, 현재 위치를 인자로 보내면 해당 페이지 번호가 할당되는 프레임 번호를 리턴해준다. PA 변수는 물리주소로 (프레임번호 * 페이지크기) + 오프셋 연산을 통해 값을 얻을 수 있다. 오프셋 = 가상주소 % 페이지크기로 얻을 수 있다. faultFlag 에 따라 Fault 변수를 'H'혹은 'F'로 변경해준다. 이 모두를 적절한 포맷으로 파일에 입력해준다. 위 과정을 maxInputLength 만큼 반복해주는 것이다.

```
int simulate(int pageNo, int curPos) {
    // 각 page replacement 알고리즘에 맞는 pageNo 에 대한 frameNo 반환
    switch(prAlgorithm) {
        case 1:
            return simulateOptimal(pageNo, curPos);
            break;
        case 2:
            return simulateFIFO(pageNo);
            break;
        case 3:
            return simulateLRU(pageNo);
```

```
                break;
            case 4:
                return simulateSecondChance(pageNo);
                break;
        }
        return -1;
}
```

simulate() 함수는 페이지 번호와 현재 위치를 인자로 받는다. 이 때, 현재 위치는 Optimal page replacement algorithm 에서만 필요한 것이다. 사용자에게 입력받은 적용할 알고리즘에 따라 Optimal, FIFO, LRU, SecondChance 알고리즘을 각각 다르게 호출해준다. 모두 페이지 번호를 받으면 알맞은 프레임 번호를 리턴해주는 함수들이다.

1. Optimal 구현

Optimal Page Replacement Algorithm 은 앞으로 가장 오랫동안 사용되지 않은 page 를 쫓아내는 것이다. 그렇기에 현재 위치를 의미하는 curPos 변수가 필요하다. 만약 프레임 내의 모든 페이지 번호가 재사용된다면 가장 늦게 재사용되는 page 를 쫓아낸다. 하지만, 모든 페이지 번호가 재사용되지 않을 수 있다. (재사용되지 않는 페이지들이 존재할 수 있다.) 재사용되지 않는 페이지가 딱 1 개 존재한다면 해당 페이지가 가장 오랫동안 사용되지 않는 페이지가 되는 것이기 때문에 해당 페이지를 쫓아낸다. 만약, 재사용되지 않는 페이지가 여러 개 존재한다면, 누가 가장 오랫동안 사용되지 않는 페이지인지 특정할 수 없기 때문에 그 경우에는 FIFO 를 적용하여 가장 먼저 들어와있었던 페이지를 쫓아낸다.

누가 가장 오랫동안 사용되지 않았는지를 판단하기 위해 optDist 배열을 활용하였다. 이는 현재 inputList 의 인덱스로부터 재사용되는 인덱스까지 얼마나 떨어져있는지에 대한 정보가 저장되어있다. 또한, 특정할 수 없을 때 FIFO 를 적용하기 위해 optFifo 배열을 활용하였다. optFifo[i] = j 는 프레임 번호 i 가 삽입된 순서 j 를 의미한다. 누가 가장 먼저 사용됐는지만 확인하면 되기 때문에 Fifo 를 완벽하게 구현하지 않고 삽입될 때마다 optFifoCnt 를 누적해서 저장해주는 방식을 사용하였다. optFifo 배열 내에 최솟값인 인덱스를 x 라고 하면 프레임 x 번이 현재 프레임들 중에 가장 먼저 삽입된 프레임을 의미한다.

```
int simulateOptimal(int pageNo, int curPos) {
        faultFlag = true;
        cntFaults++;
        for (int i = 0; i < frameNumber; i++) {
            if (pageList[i] == -1) {        // 빈 프레임이 있을 경우 저장
                pageList[i] = pageNo;
```

```
                optFifo[i] = optFifoCnt++;
                return i;
            } else if (pageList[i] == pageNo) {        // 이미 프레임에 존재할 경우
                faultFlag = false;
                cntFaults--;
                optFifo[i] = optFifoCnt++;
                return i;
            }
        }
        int victim = optUpdate(curPos);
        pageList[victim] = pageNo;
        optFifo[victim] = optFifoCnt++;
        return victim;
}
```

simulateOptimal() 함수이다.pageList 를 순회하면서 -1 값이 저장된 인덱스가 있는지 확인한다. 이는 해당 프레임 번호에 해당하는 페이지번호가 할당되지 않았음을 의미한다. 빈 공간이기 때문에 해당 프레임에 페이지번호를 저장해준다. 삽입에 해당하므로 optFifo 를 업데이트해준다. 또한, 만약 pageList 를 순회하면서 해당 페이지 번호가 이미 저장되어있음이 확인된다면, faultFlag 를 false 로 바꿔준 후, optFifo 만 업데이트해준다.

만약 빈 프레임도 없고, 프레임에 이미 저장되어있지도 않다면 optUpdate 를 통해 victim 프레임을 찾아야한다. Victim 프레임을 찾았으면 해당 프레임에 페이지 번호를 업데이트 해주고 optFifo 도 업데이트 해주고 프레임 번호를 리턴해준다.

```
int optUpdate(int curPos) {
        for (int i = 0; i < frameNumber; i++) {    // 재사용 거리 초기화
            optDist[i] = -1;
        }
        int PA = 0;
        for (int i = 0; i < frameNumber; i++) {
            for (int j = curPos + 1; j < maxInputLength; j++) {
                PA = inputList[j] / (1024 * pageSize);    // 가상 주소 -> 페이지 번호 변환
                if (pageList[i] == PA) {
                    optDist[i] = j;        // 얼마나 멀리 떨어져있는지 거리 저장
                    break;
                }
            }
        }
        int tmp = optCount();      // 미래에 재사용되는게 몇 개인지
        if (tmp != -1) return tmp;    // 모두 재사용됐거나 하나만 재사용되지 않았을 경우 리턴
```

```
    // 재사용되지 않은 페이지가 여러 개일 경우 (FIFO 적용)
    int minCnt = 1e9;
    int ret = 0;
    for (int i = 0; i < frameNumber; i++) {
        if (optDist[i] == -1 && minCnt > optFifo[i]) {
            minCnt = optFifo[i];
            ret = i;
        }
    }
    return ret;
}
```

optUpdate() 함수이다. optDist 를 모두 -1 로 초기화해준 후 현재 위치인 curPos 이후부터 inputList 를 순회하면서 재사용됐는지를 확인해준다. 만약 재사용됐다면 optDist 를 업데이트해준 후 다음 프레임에 해당하는 페이지 번호에 대해 또 순회해준다. 모두 순회하고 나면 optDist 에는 재사용됐다면 해당 거리가, 재사용되지 않았다면 -1 이 저장되어있다.

optCount() 함수를 통해 프레임 번호를 리턴해준다. 만약, 재사용되지 않은 페이지가 여러 개일 경우 optCount()함수는 -1 을 리턴해주는데, 이 때는 FIFO 를 적용하여 victim 을 선정해야한다. minCnt, ret 은 optFifo 배열에서 가장 작은 값과 인덱스를 저장하기 위한 변수이다. optFifo 에서 최솟값이 프레임번호들 중 가장 먼저 업데이트됐다는 의미이므로 ret 을 리턴해준다.

```
int optCount() {
    int cnt = 0, maxDist = 0, ret = 0, noRet = 0;
    // 재사용된 페이지 개수, 재사용 거리 최댓값, 최댓값일 때의 페이지 번호, 재사용되지 않은 페이지 번호
    for (int i = 0; i < frameNumber; i++) {
        if (optDist[i] == -1) {
            noRet = i;
            continue;
        }
        cnt++;
        if (maxDist < optDist[i]) {
            maxDist = optDist[i];
            ret = i;
        }
    }
    if (cnt == frameNumber) return ret;          // 모두 재사용됐을 경우
```

```
    if (cnt == frameNumber - 1) return noRet;       // 하나만 재사용되지 않았을 경우
    return -1; // 재사용되지 않은 페이지가 여러 개인 경우
}
```

optCount()는 optUpdate()에서 업데이트된 optDist 배열을 활용한다. optDist 배열을 순회하면서 -1(재사용되지 않은 프레임)이 아닌 프레임이 몇 개인지를 의미하는 cnt 변수, optDist 의 최댓값을 의미하는 maxDist 변수, 최댓값일 때의 프레임 번호를 의미하는 ret 변수, 재사용되지 않은 페이지 번호를 의미하는 noRet 변수를 선언해주었다. 만약 재사용되지 않은 페이지 번호가 1 개라면 noRet 이 victim 이 된다. 모두 사용됐을 경우에는 ret 이 victim 이 되고, 재사용되지 않은 페이지가 여러 개라면 -1 을 리턴하여 optUpdate()에서 다른 처리를 할 수 있게 해준다.

2. FIFO 구현

First In First Out(FIFO) Page Replacement Algorithm 은 가장 간단한 알고리즘으로, 먼저 들어온 페이지를 먼저 쫓아내는 방식이다. Beladys' Anomaly 가 발생할 수 있으며 구현이 용이한 대신 성능이 최악이라는 특징이 있다.

```
typedef struct {
    int items[5003];
    int front;
    int rear;
} Queue;
Queue q = {.front = 0, .rear = 0};
```

FIFO 방식이므로 자료구조로는 queue 가 가장 적합하다. 그렇기에 구조체 Queue 를 선언해주었고, 구조체 변수 q 를 선언하여 front, rear 을 0 으로 초기화해주었다. front 는 queue 의 앞 쪽(먼저 삽입된 쪽) 인덱스를 저장하고, rear 에는 뒷 쪽(나중에 삽입된 쪽) 인덱스를 저장한다.

```
int simulateFIFO(int pageNo) {
    faultFlag = true;
    cntFaults++;
    for (int i = 0; i < frameNumber; i++) {
        if (pageList[i] == -1) {        // 빈 프레임 존재
            pageList[i] = pageNo;
            fifoInsert(i);
            return i;
        }
        else if (pageList[i] == pageNo) {     // 이미 프레임에 존재
            faultFlag = false;
```

```
                cntFaults--;
                return i;
            }
        }
        int front = fifoDelete();
        pageList[front] = pageNo;
        fifoInsert(front);
        return front;
}
```

simulateFIFO() 함수 또한 simulateOpt() 와 마찬가지로 페이지 번호를 입력받은 후 프레임 번호를 리턴해준다. pageList 배열을 순회하면서 -1(빈 프레임)이면 입력해준 후 fifoInsert() 함수를 통해 q 에 삽입해준다. 페이지번호가 이미 pageList 배열에 존재한다면 faultFlag 를 false 로 바꿔주고 바로 리턴해준다. 만약, 빈프레임도 아니고 프레임에 이미 존재하지도 않다면 fifoDelete() 함수를 통해 front 에 해당하는 값을 반환하고 front 를 하나 키워서 delete 한 이후의 배열 인덱스를 의미하도록 수정해준다. 해당 front 에 페이지 번호를 저장하고 나면 이 또한 새로운 삽입에 해당하므로 fifoInsert()를 수행해준다.

```
void fifoInsert(int item) {
        if ((q.rear + 1) % 5003 == q.front) {
            return;
        }
        q.rear = (q.rear + 1) % 5003;
        q.items[q.rear] = item;
}
int fifoDelete() {
        if (q.front == q.rear) return -1;
        q.front = (q.front + 1) % 5003;
        return q.items[q.front];
}
```

각각 fifoInsert()와 fifoDelete() 함수이다. 입력 배열의 최댓값이 5000 이므로 Queue 구조체 items 배열도 여유롭게 5003 개의 배열로 잡아주었다. 그러면 큐에서 delete(pop) 연산할 때 뒷 쪽 데이터들을 앞으로 옮겨주는 작업으르 하지 않아도 돼서 시간복잡도를 절약할 수 있다.

3. LRU 구현

Least Recently Used(LRU) Page Replacement Algorithm 은 가장 오랫동안 사용되지 않은 page 와 새로 로딩될 page 를 교체한다. FIFO 와 달리 Belady's Anomaly 가 발생하지

않으며 Optimal 에 더 가깝다는 장점이 있다. 해당 알고리즘은 Counter 를 이용하는 방법과 Stack 을 이용하는 방법이 있다. 이 중, Stack 을 이용하는 방법은 새로 들어올 page 가 기존 stack 에 있다면 bottom 으로 이동시키고, 없다면 bottom 에 추가하는 방식이다. top 에는 update 가 가장 오래된 page 가 존재한다. 그렇기에 기존 stack 에 없고, 더 이상 추가할 공간이 없다면 top 을 pop 시킨 후, 새로운 page 를 bottom 에 추가한다.

```
typedef struct node {
	int data;
	struct node *next;
} Stack;
Stack *lruHead = NULL;
```

해당 구현 방식대로 원래는 stack 으로 구현하려고 했으나, 추가하려는 page 가 stack 에 있는지 여부를 순회해서 탐색해야한다는 점, stack 에 있다면 top 에 있지 않더라도 제거해야 한다는 점, pop 은 top 에서 하지만 추가는 bottom 에서 한다는 점을 고려해봤을 때 stack 보다는 linked-list 가 더 적합한 자료구조라고 판단해 linked-list 로 구현하였다.

```
int simulateLRU(int pageNo) {
	faultFlag = true;
	cntFaults++;
	for (int i = 0; i < frameNumber; i++) {
		if (pageList[i] == -1) {
			lruInsert(i);
			pageList[i] = pageNo;
			return i;
		} else if (pageList[i] == pageNo) {
			faultFlag = false;
			cntFaults--;
			lruUpdate(i);
			return i;
		}
	}
	int front = lruHead -> data;
	pageList[front] = pageNo;
	lruInsert(front);
	lruHead = lruHead -> next;
	return front;
```

```
}
```

simulateLRU 도 마찬가지로 해당 프레임이 비어있는지, 해당 프레임에 이미 페이지 번호가 존재한지 여부를 확인한 후, 둘 다 아니라면 lruHead 가 가리키고 있는 프레임번호를 victim 으로 설정해준다. Victim 으로 설정해준 후, lruHead 는 그 다음 데이터를 가리키게 함으로써 delete 효과를 준다.

```
void lruUpdate(int data) {
	Stack *prev = NULL, *cur = lruHead;
	while(cur != NULL) {	//Stack 에 data 존재하는지 탐색
		if (cur->data == data) {	//존재한다면
			faultFlag = false;
			if (cur -> next == NULL) return;	//이미 마지막 노드
			break;
		}
		prev = cur;
		cur = cur -> next;
	}

	if (cur != NULL) {	//Stack 에 데이터 존재
		if (prev != NULL) prev -> next = cur -> next;	//Stack 에서 제거
		else lruHead = cur -> next;	//data 가 첫 번째 노드인 경우
	}
	lruInsert(data);	//Stack 마지막에 삽입
}
```

페이지 번호가 이미 존재하는 경우, lruUpdate()를 호출한다. 이는 Stack 에서 data 를 탐색하고, 존재한다면 이를 Stack 에서 제거하고 lruInsert()를 호출해준다.

```
void lruInsert(int data) {
	Stack *newNode = (Stack*)malloc(sizeof(Stack));
	newNode -> data = data;
	newNode -> next = NULL;

	if (lruHead == NULL) {
		lruHead = newNode;
	} else {
		Stack *last = lruHead;
		while (last -> next != NULL) last = last -> next;
		last -> next = newNode;
	}
```

```
}
```

lruInsert 는 lruHead 마지막에 프레임번호에 해당하는 데이터를 새로운 노드로 생성하여 삽입해주는 것이다.

4. Second-Chance(One-handed Clock)

Second-Chance(One-handed Clock) Page Replacement Algorithm 은 최근에 참조가 많이 됐던 page 에게 한 번 더 기회를 주는 알고리즘이다. 이를 위해 reference bit 를 활용하여 1 이면 0 으로 바꾼 후 다음 포인터로 이동하고, 0 이면 해당 프레임을 victim 으로 선정하여 쫓아낸 후 해당 위치에 새로운 페이지를 삽입시킨다. 이런 알고리즘을 구현하기 위해, 포인터가 계속 다음으로 이동한다는 점에 초점을 맞춰 circular queue 이 적합하다고 판단하였다.

```
int simulateSecondChance(int pageNo) {
        faultFlag = true;
        cntFaults++;
        for (int i = 0; i < frameNumber; i++) {
                if (pageList[i] == -1) {
                        scInsert(i);
                        pageList[i] = pageNo;
                        return i;
                } else if (pageList[i] == pageNo) {
                        faultFlag = false;
                        cntFaults--;
                        scRefOn(i);          // 참조했으니 reference bit = 1 로 업데이트
                        return i;
                }
        }
        int victim = scUpdate();
        if (victim != -1) pageList[victim] = pageNo;        //교체할 페이지 찾음
        else scInsert(pageNo);    // scHead == NULL 일 경우 그냥 삽입
        return victim;
}
```

simulateSecondChance()는 위의 simulate 함수들과 마찬가지로 페이지 번호를 인자로 받아 프레임 번호를 반환해준다. pageList 를 순회하면서 빈 프레임이면 scInsert() 함수를 통해 Circular queue 에 삽입해준 후 해당 프레임 번호를 반환한다. 이미 프레임에 존재하는 페이지 번호면 faultFlag 를 false 로 바꿔준 후 참조했다는 의미로 scRefOn() 함수를 통해 reference bit 을 1 로 바꿔주는 작업 후에 프레임 번호를 반환한다. 만약, 빈

프레임도 이미 존재하는 페이지 번호도 아니라면, scUpdate 를 통해 victim 을 선정한 후 반환해준다.

```
int scUpdate() {
	if (scHead == NULL) return -1;
	Circular *start = vtm;
	while(1) {
		if (vtm == NULL) vtm = scHead;
		if (!(vtm -> refBit)) {
			int victimData = vtm -> data;
			vtm = vtm -> next;
			return victimData;
		}
		vtm -> refBit = false;
		vtm = vtm -> next;
	}
}
```

scUpdate() 함수는 vtm 을 이동시키면서 refBit 가 false 인 프레임번호를 찾아 리턴한다. 만약 순회하면서 refBit 가 true 이면 false 로 변경한 후 다음으로 이동한다.

```
void scRefOn(int data) {
	Circular *cur = scHead;
	while (cur != NULL) {
		if (cur -> data == data) {
			cur -> refBit = true;
			return;
		}
		cur = cur -> next;
	}
}
```

scRefOn() 함수는 scHead 부터 순회하면서 해당 data 에 해당하는 node 를 찾아 refBit 를 true 로 변경한다.

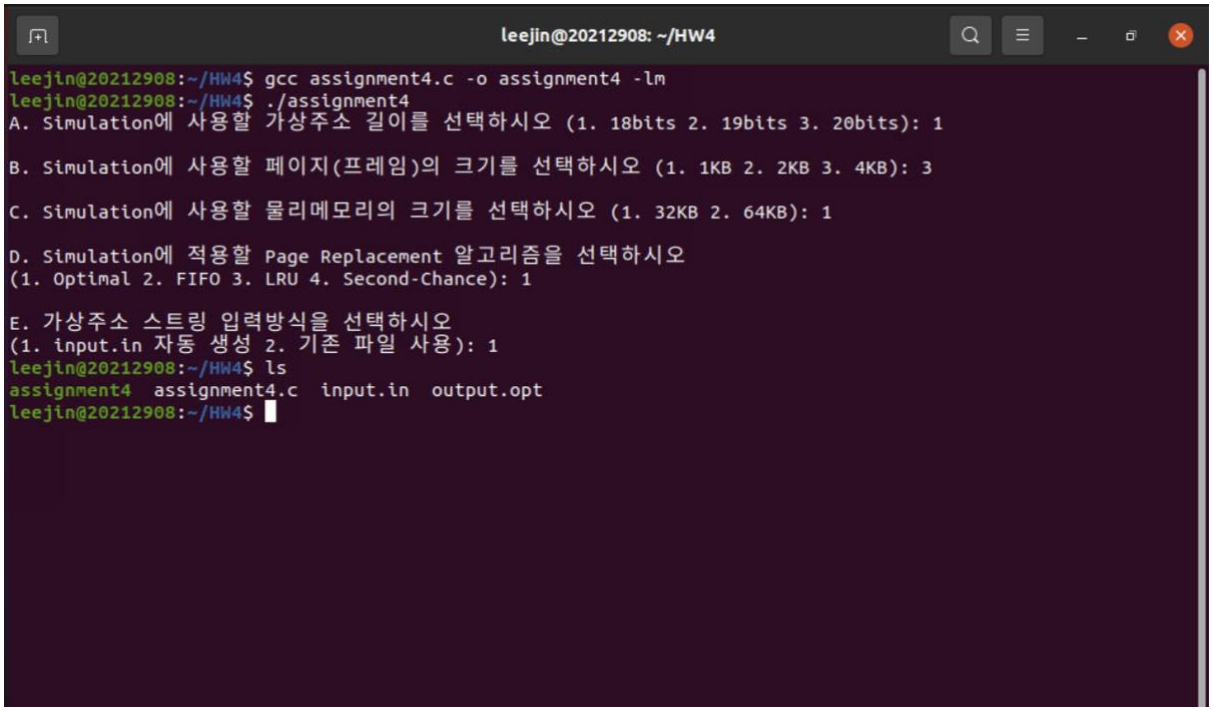
```
void scInsert(int data) {
	Circular *newNode = (Circular*)malloc(sizeof(Circular));
	newNode -> data = data;
	newNode -> refBit = false;

	if (scHead == NULL) {
```

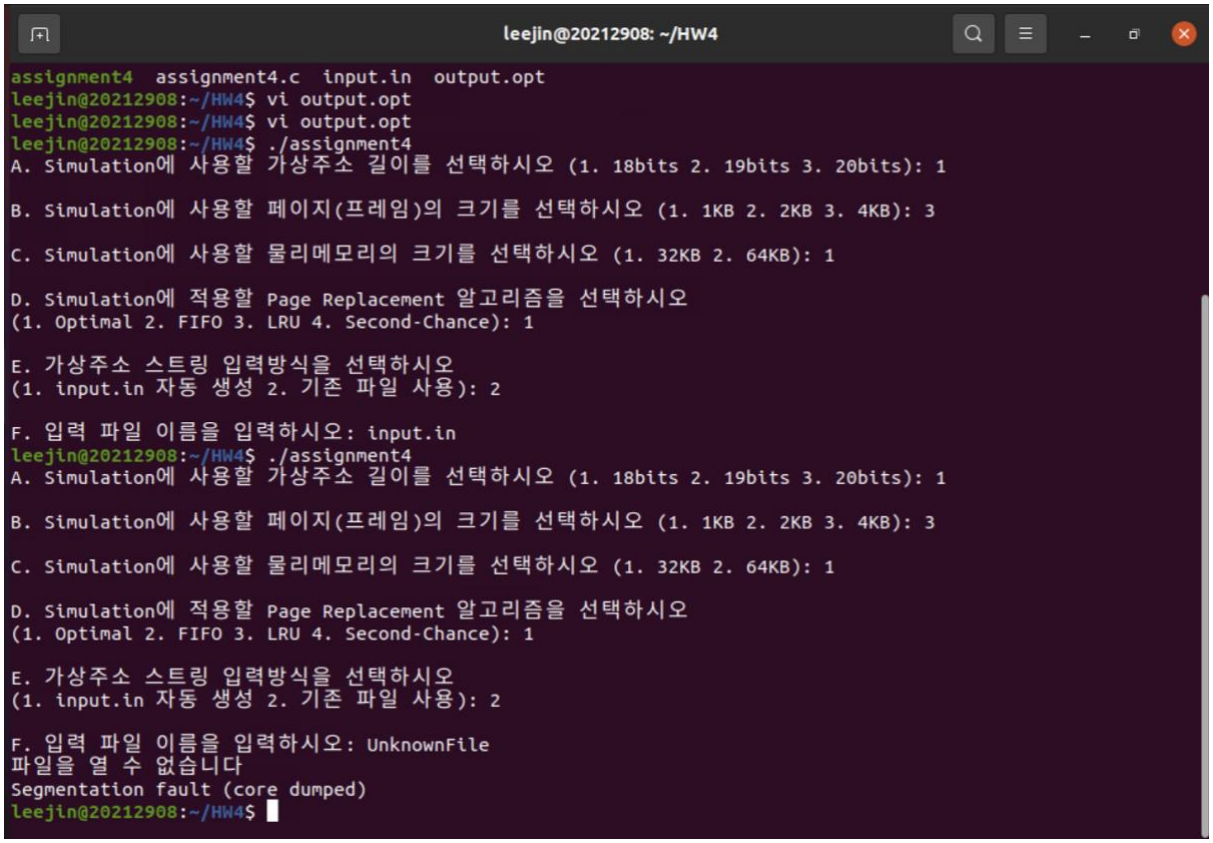
```
            scHead = newNode;
            newNode -> next = scHead;
        } else {
            Circular *last = scHead;
            while (last -> next != scHead) last = last -> next;
            last -> next = newNode;
            newNode -> next = scHead;
        }
}
```

scInsert() 함수는 해당 데이터를 가지면서 refBit 가 false 인 새로운 노드를 생성 후 scHead 뒤에 삽입한다.

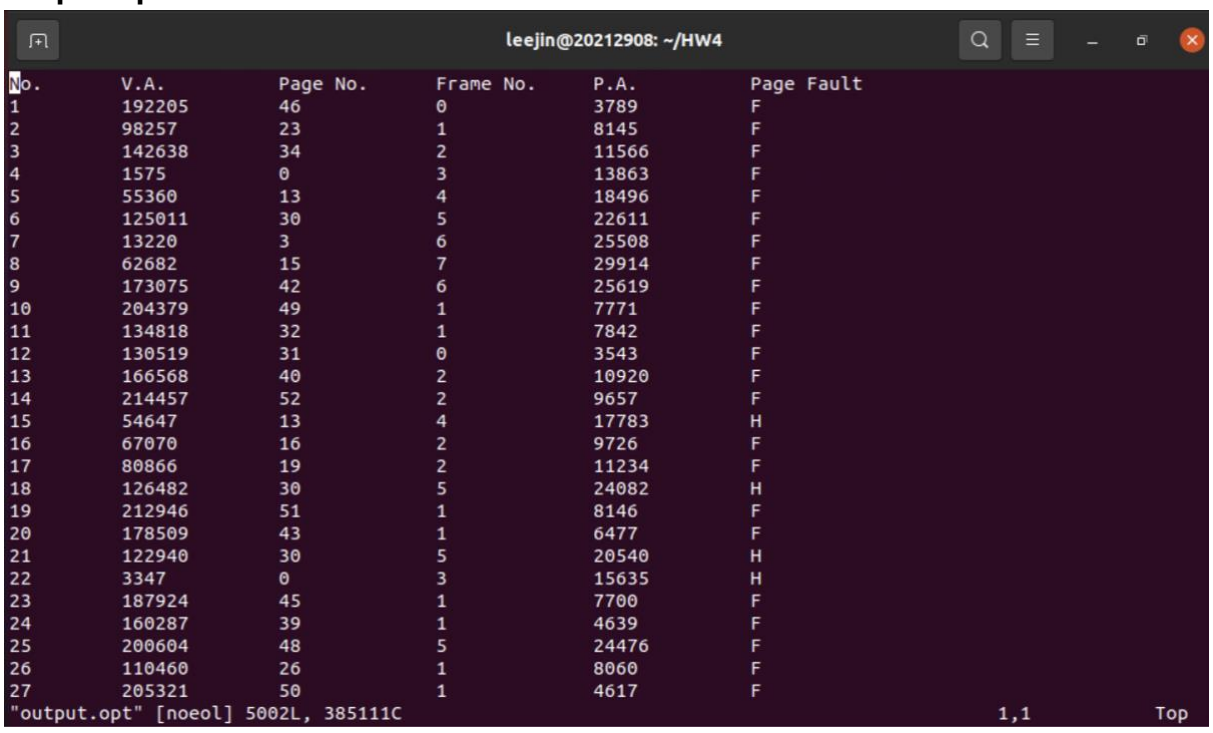
[실행 화면 스냅샷]

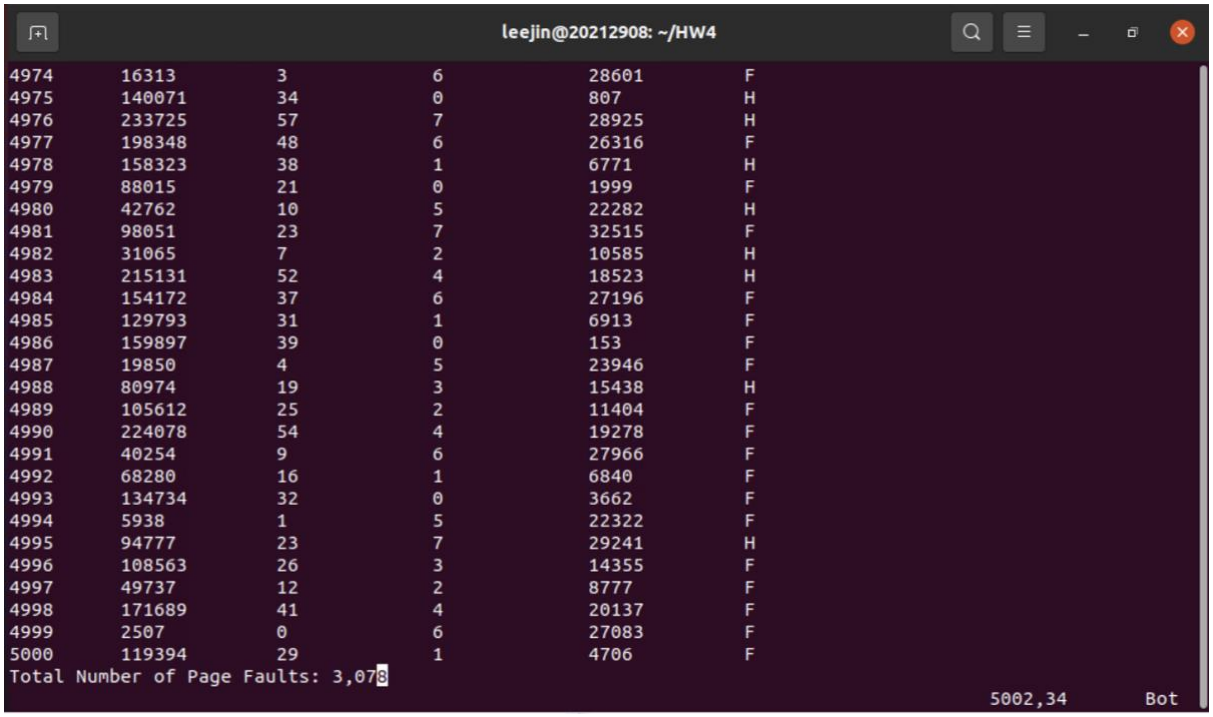
초기 실행 화면



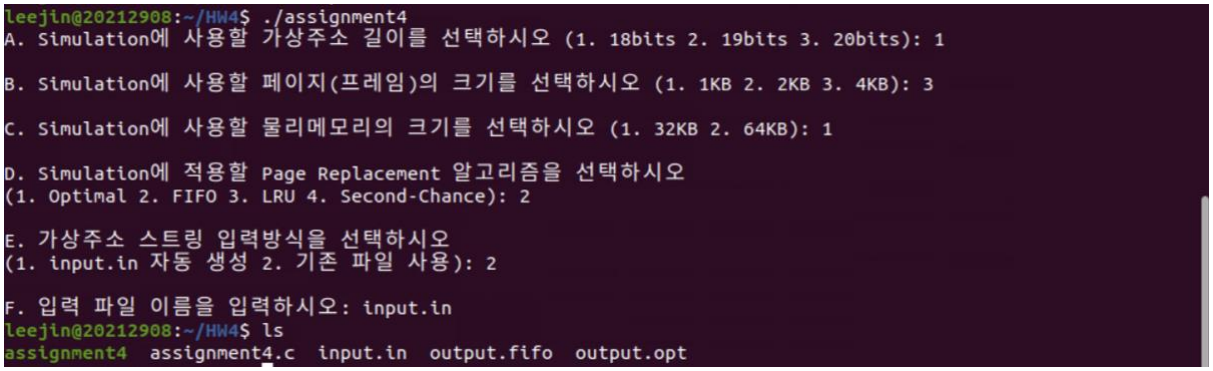

leejin@20212908: ~/HW4
assignment4 assignment4.c input.in output.opt
leejin@20212908:~/HW4$ vi output.opt
leejin@20212908:~/HW4$ vi output.opt
leejin@20212908:~/HW4$ ./assignment4
A. Simulation에 사용할 가상주소 길이를 선택하시오 (1. 18bits 2. 19bits 3. 20bits): 1
B. Simulation에 사용할 페이지(프레임)의 크기를 선택하시오 (1. 1KB 2. 2KB 3. 4KB): 3
C. Simulation에 사용할 물리메모리의 크기를 선택하시오 (1. 32KB 2. 64KB): 1
D. Simulation에 적용할 Page Replacement 알고리즘을 선택하시오
(1. Optimal 2. FIFO 3. LRU 4. Second-Chance): 1
E. 가상주소 스트링 입력방식을 선택하시오
(1. input.in 자동 생성 2. 기존 파일 사용): 2
F. 입력 파일 이름을 입력하시오: input.in
leejin@20212908:~/HW4$ ./assignment4
A. Simulation에 사용할 가상주소 길이를 선택하시오 (1. 18bits 2. 19bits 3. 20bits): 1
B. Simulation에 사용할 페이지(프레임)의 크기를 선택하시오 (1. 1KB 2. 2KB 3. 4KB): 3
C. Simulation에 사용할 물리메모리의 크기를 선택하시오 (1. 32KB 2. 64KB): 1
D. Simulation에 적용할 Page Replacement 알고리즘을 선택하시오
(1. Optimal 2. FIFO 3. LRU 4. Second-Chance): 1
E. 가상주소 스트링 입력방식을 선택하시오
(1. input.in 자동 생성 2. 기존 파일 사용): 2
F. 입력 파일 이름을 입력하시오: UnknownFile
파일을 열 수 없습니다
Segmentation fault (core dumped)
leejin@20212908:~/HW4$

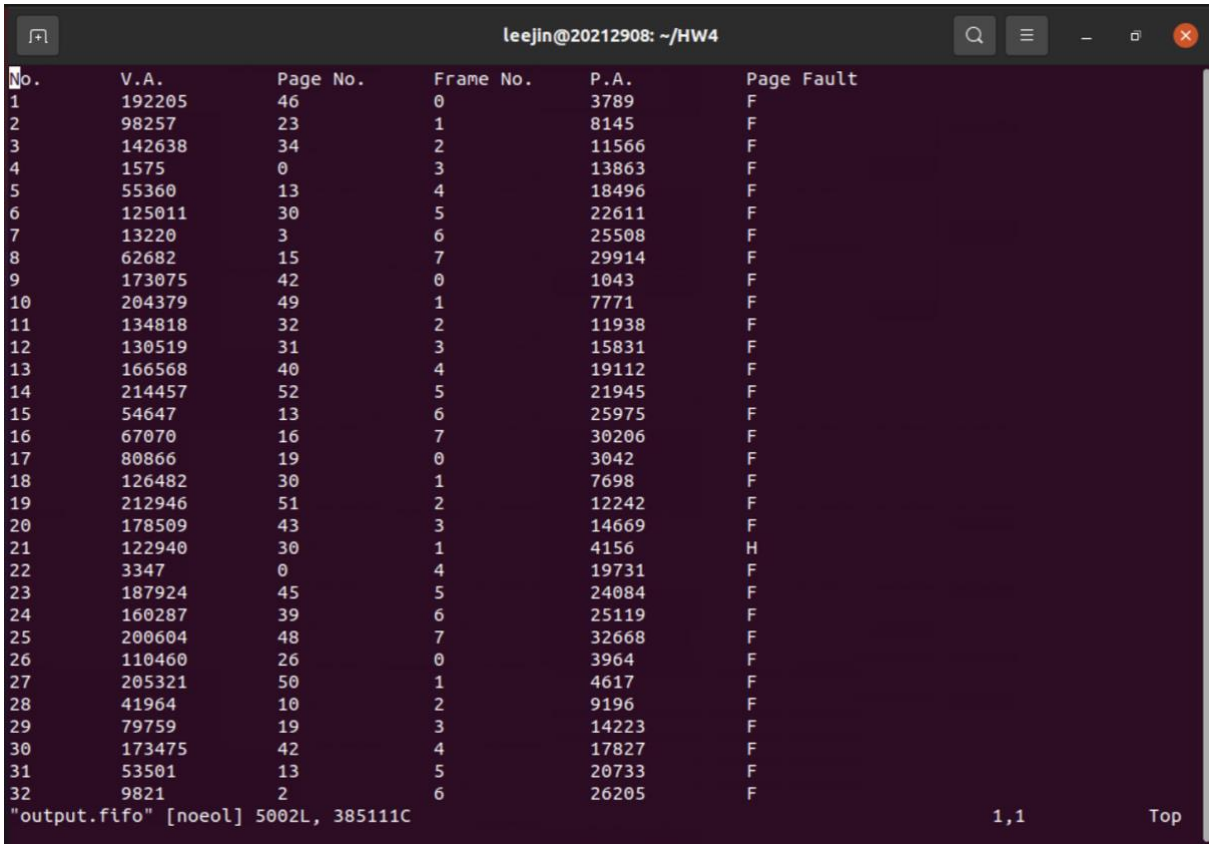
output.opt

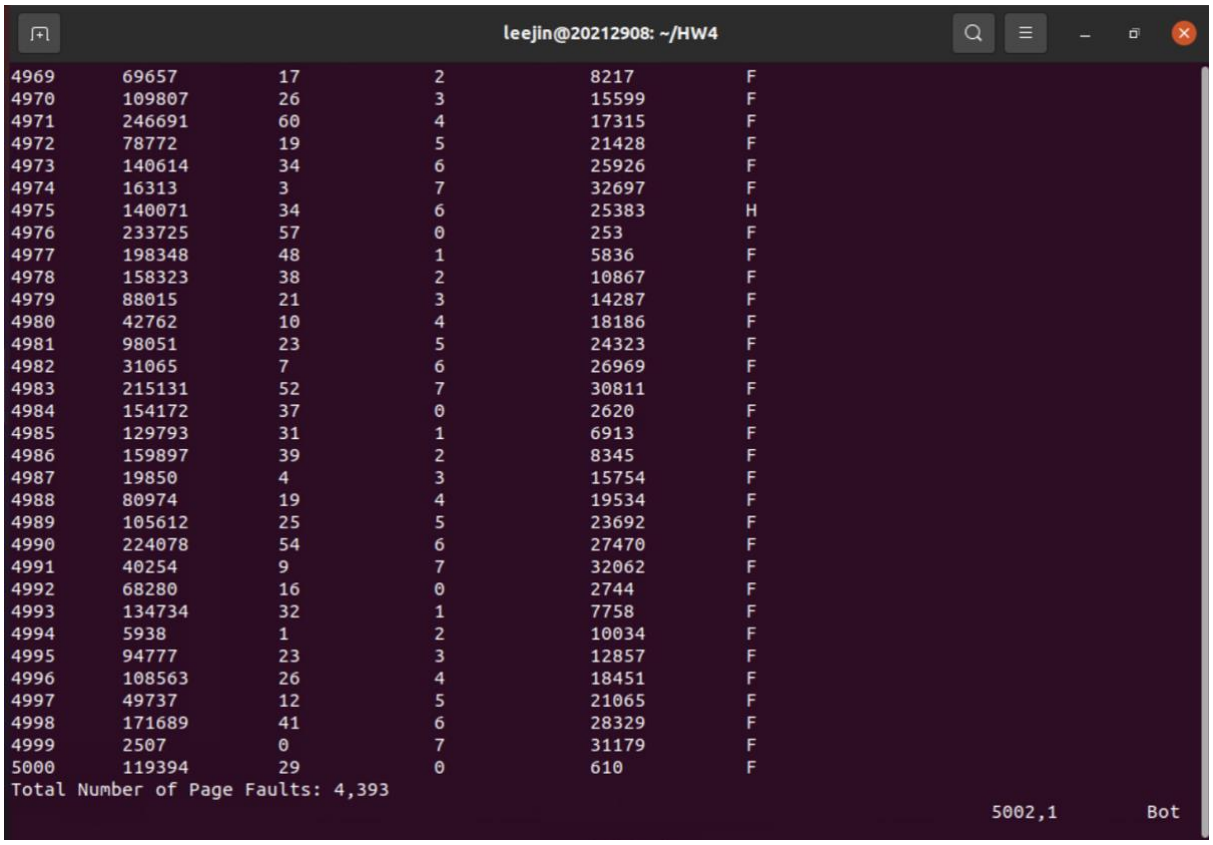

leejin@20212908: ~/HW4
No. V.A. Page No. Frame No. P.A. Page Fault
1 192205 46 0 3789 F
2 98257 23 1 8145 F
3 142638 34 2 11566 F
4 1575 0 3 13863 F
5 55360 13 4 18496 F
6 125011 30 5 22611 F
7 13220 3 6 25508 F
8 62682 15 7 29914 F
9 173075 42 6 25619 F
10 204379 49 1 7771 F
11 134818 32 1 7842 F
12 130519 31 0 3543 F
13 166568 40 2 10920 F
14 214457 52 2 9657 F
15 54647 13 4 17783 H
16 67070 16 2 9726 F
17 80866 19 2 11234 F
18 126482 30 5 24082 H
19 212946 51 1 8146 F
20 178509 43 1 6477 F
21 122940 30 5 20540 H
22 3347 0 3 15635 H
23 187924 45 1 7700 F
24 160287 39 1 4639 F
25 200604 48 5 24476 F
26 110460 26 1 8060 F
27 205321 50 1 4617 F
"output.opt" [noeol] 5002L, 385111C 1,1 Top


leejin@20212908: ~/HW4
4974 16313 3 6 28601 F
4975 140071 34 0 807 H
4976 233725 57 7 28925 H
4977 198348 48 6 26316 F
4978 158323 38 1 6771 H
4979 88015 21 0 1999 F
4980 42762 10 5 22282 H
4981 98051 23 7 32515 F
4982 31065 7 2 10585 H
4983 215131 52 4 18523 H
4984 154172 37 6 27196 F
4985 129793 31 1 6913 F
4986 159897 39 0 153 F
4987 19850 4 5 23946 F
4988 80974 19 3 15438 H
4989 105612 25 2 11404 F
4990 224078 54 4 19278 F
4991 40254 9 6 27966 F
4992 68280 16 1 6840 F
4993 134734 32 0 3662 F
4994 5938 1 5 22322 F
4995 94777 23 7 29241 H
4996 108563 26 3 14355 F
4997 49737 12 2 8777 F
4998 171689 41 4 20137 F
4999 2507 0 6 27083 F
5000 119394 29 1 4706 F
Total Number of Page Faults: 3,078
5002,34 Bot

output.fifo


leejin@20212908:~/HW4$ ./assignment4
A. Simulation에 사용할 가상주소 길이를 선택하시오 (1. 18bits 2. 19bits 3. 20bits): 1
B. Simulation에 사용할 페이지(프레임)의 크기를 선택하시오 (1. 1KB 2. 2KB 3. 4KB): 3
C. Simulation에 사용할 물리메모리의 크기를 선택하시오 (1. 32KB 2. 64KB): 1
D. Simulation에 적용할 Page Replacement 알고리즘을 선택하시오
(1. Optimal 2. FIFO 3. LRU 4. Second-Chance): 2
E. 가상주소 스트링 입력방식을 선택하시오
(1. input.in 자동 생성 2. 기존 파일 사용): 2
F. 입력 파일 이름을 입력하시오: input.in
leejin@20212908:~/HW4$ ls
assignment4 assignment4.c input.in output.fifo output.opt


leejin@20212908: ~/HW4
No. V.A. Page No. Frame No. P.A. Page Fault
1 192205 46 0 3789 F
2 98257 23 1 8145 F
3 142638 34 2 11566 F
4 1575 0 3 13863 F
5 55360 13 4 18496 F
6 125011 30 5 22611 F
7 13220 3 6 25508 F
8 62682 15 7 29914 F
9 173075 42 0 1043 F
10 204379 49 1 7771 F
11 134818 32 2 11938 F
12 130519 31 3 15831 F
13 166568 40 4 19112 F
14 214457 52 5 21945 F
15 54647 13 6 25975 F
16 67070 16 7 30206 F
17 80866 19 0 3042 F
18 126482 30 1 7698 F
19 212946 51 2 12242 F
20 178509 43 3 14669 F
21 122940 30 1 4156 H
22 3347 0 4 19731 F
23 187924 45 5 24084 F
24 160287 39 6 25119 F
25 200604 48 7 32668 F
26 110460 26 0 3964 F
27 205321 50 1 4617 F
28 41964 10 2 9196 F
29 79759 19 3 14223 F
30 173475 42 4 17827 F
31 53501 13 5 20733 F
32 9821 2 6 26205 F
"output.fifo" [noeol] 5002L, 385111C
1,1 Top


leejin@20212908: ~/HW4
4969 69657 17 2 8217 F
4970 109807 26 3 15599 F
4971 246691 60 4 17315 F
4972 78772 19 5 21428 F
4973 140614 34 6 25926 F
4974 16313 3 7 32697 F
4975 140071 34 6 25383 H
4976 233725 57 0 253 F
4977 198348 48 1 5836 F
4978 158323 38 2 10867 F
4979 88015 21 3 14287 F
4980 42762 10 4 18186 F
4981 98051 23 5 24323 F
4982 31065 7 6 26969 F
4983 215131 52 7 30811 F
4984 154172 37 0 2620 F
4985 129793 31 1 6913 F
4986 159897 39 2 8345 F
4987 19850 4 3 15754 F
4988 80974 19 4 19534 F
4989 105612 25 5 23692 F
4990 224078 54 6 27470 F
4991 40254 9 7 32062 F
4992 68280 16 0 2744 F
4993 134734 32 1 7758 F
4994 5938 1 2 10034 F
4995 94777 23 3 12857 F
4996 108563 26 4 18451 F
4997 49737 12 5 21065 F
4998 171689 41 6 28329 F
4999 2507 0 7 31179 F
5000 119394 29 0 610 F
Total Number of Page Faults: 4,393
5002,1 Bot

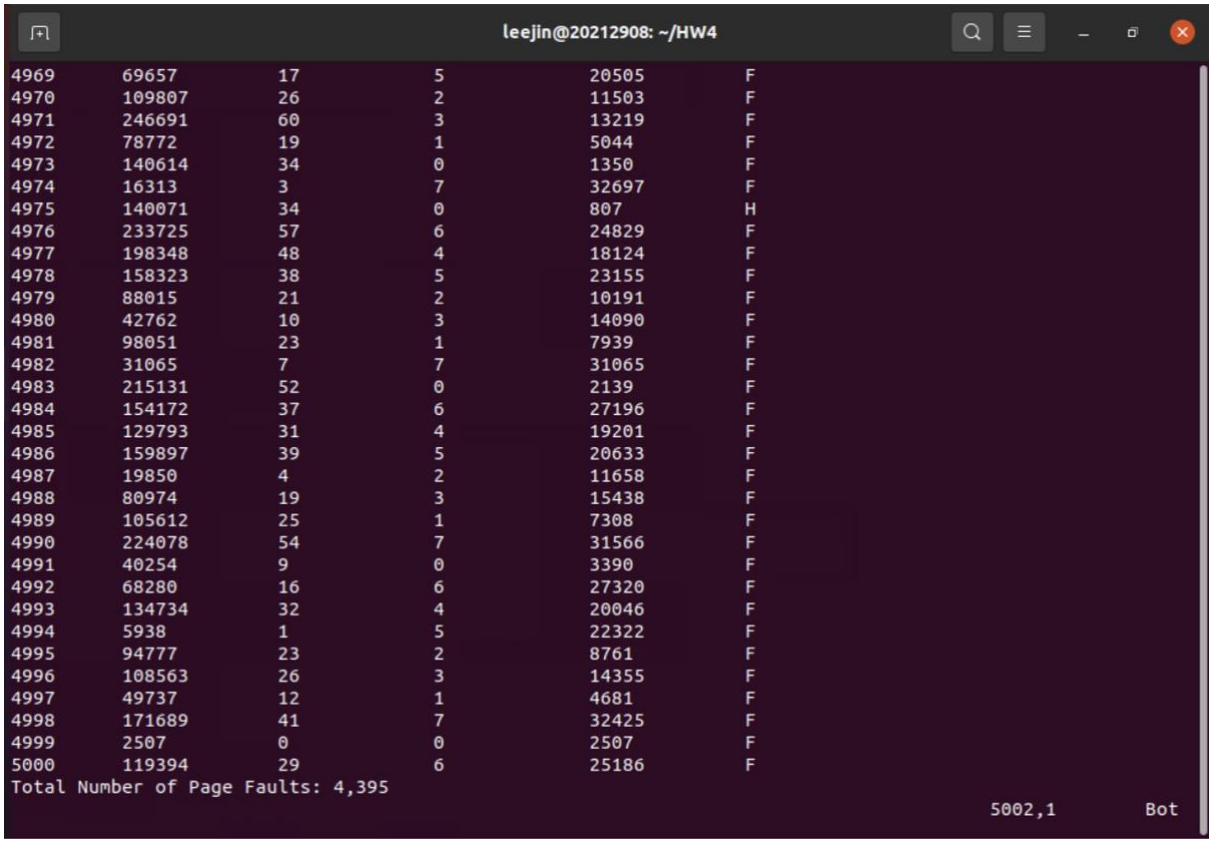
output.lru

```
leejin@20212908:~/HW4$ ./assignment4
A. Simulation에 사용할 가상주소 길이를 선택하시오 (1. 18bits 2. 19bits 3. 20bits): 1

B. Simulation에 사용할 페이지(프레임)의 크기를 선택하시오 (1. 1KB 2. 2KB 3. 4KB): 3

C. Simulation에 사용할 물리메모리의 크기를 선택하시오 (1. 32KB 2. 64KB): 1

D. Simulation에 적용할 Page Replacement 알고리즘을 선택하시오
(1. Optimal 2. FIFO 3. LRU 4. Second-Chance): 3

E. 가상주소 스트링 입력방식을 선택하시오
(1. input.in 자동 생성 2. 기존 파일 사용): 2

F. 입력 파일 이름을 입력하시오: input.in
leejin@20212908:~/HW4$ ls
assignment4  assignment4.c  input.in  output.fifo  output.lru  output.opt
leejin@20212908:~/HW4$ vi output.lru
```

leejin@20212908: ~/HW4

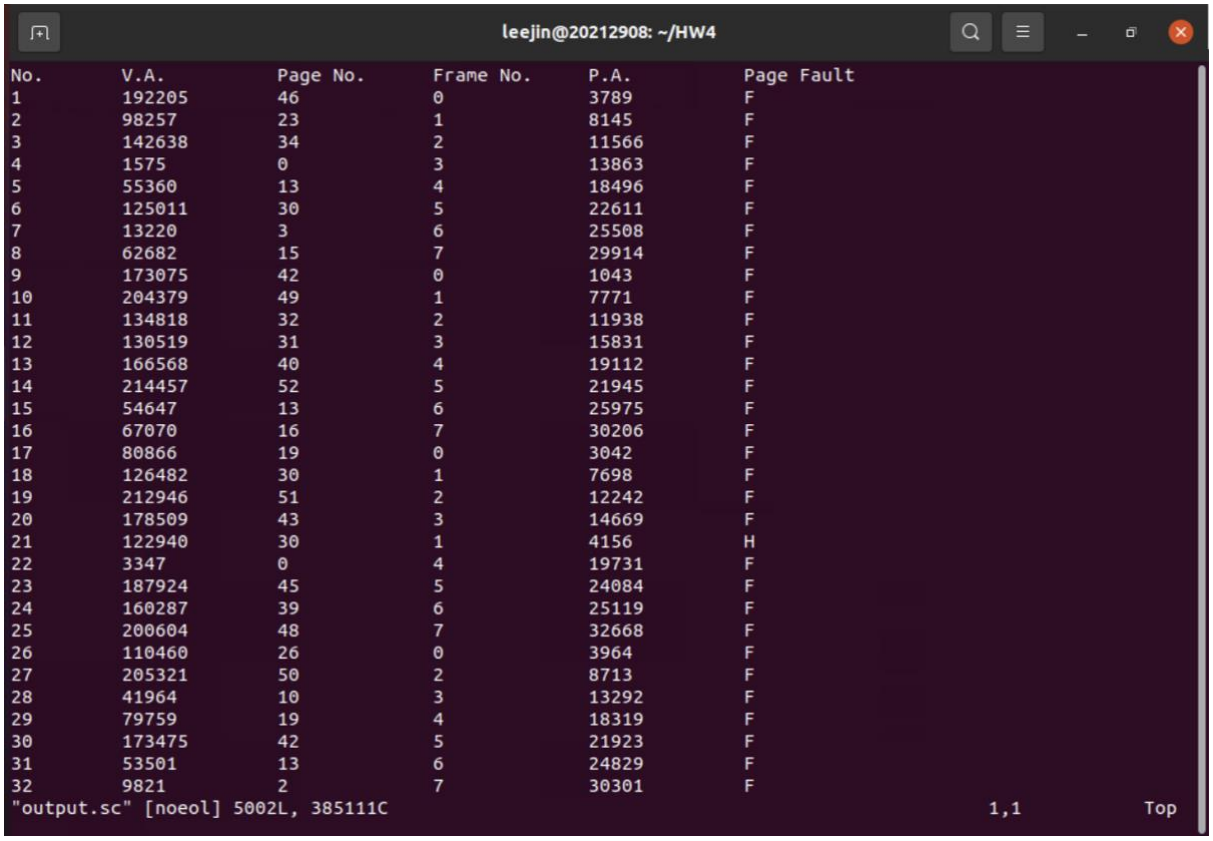
No.	V.A.	Page No.	Frame No.	P.A.	Page Fault
1	192205	46	0	3789	F
2	98257	23	1	8145	F
3	142638	34	2	11566	F
4	1575	0	3	13863	F
5	55360	13	4	18496	F
6	125011	30	5	22611	F
7	13220	3	6	25508	F
8	62682	15	7	29914	F
9	173075	42	0	1043	F
10	204379	49	1	7771	F
11	134818	32	2	11938	F
12	130519	31	3	15831	F
13	166568	40	4	19112	F
14	214457	52	5	21945	F
15	54647	13	6	25975	F
16	67070	16	7	30206	F
17	80866	19	0	3042	F
18	126482	30	1	7698	F
19	212946	51	2	12242	F
20	178509	43	3	14669	F
21	122940	30	1	4156	H
22	3347	0	4	19731	F
23	187924	45	5	24084	F
24	160287	39	6	25119	F
25	200604	48	7	32668	F
26	110460	26	0	3964	F
27	205321	50	2	8713	F
28	41964	10	3	13292	F
29	79759	19	1	6031	F
30	173475	42	4	17827	F
31	53501	13	5	20733	F
32	9821	2	6	26205	F

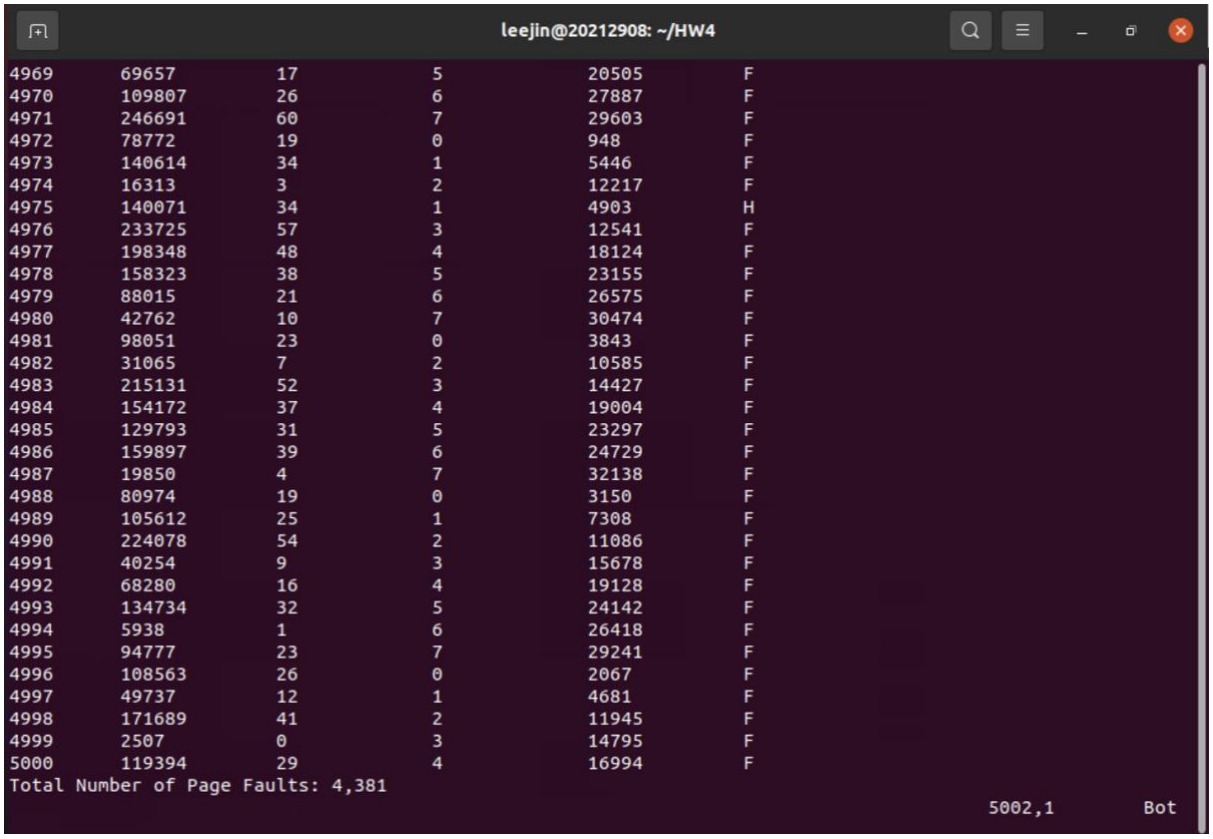
"output.lru" [noeol] 5002L, 385111C 1,1 Top



output.sc

```
leejin@20212908:~/HW4$ ./assignment4
A. Simulation에 사용할 가상주소 길이를 선택하시오 (1. 18bits 2. 19bits 3. 20bits): 1

B. Simulation에 사용할 페이지(프레임)의 크기를 선택하시오 (1. 1KB 2. 2KB 3. 4KB): 3

C. Simulation에 사용할 물리메모리의 크기를 선택하시오 (1. 32KB 2. 64KB): 1

D. Simulation에 적용할 Page Replacement 알고리즘을 선택하시오
(1. Optimal 2. FIFO 3. LRU 4. Second-Chance): 4

E. 가상주소 스트링 입력방식을 선택하시오
(1. input.in 자동 생성 2. 기존 파일 사용): 2

F. 입력 파일 이름을 입력하시오: input.in
leejin@20212908:~/HW4$ ls
assignment4  assignment4.c  input.in  output.fifo  output.lru  output.opt  output.sc
leejin@20212908:~/HW4$ vi output.sc
```


leejin@20212908: ~/HW4
No. V.A. Page No. Frame No. P.A. Page Fault
1 192205 46 0 3789 F
2 98257 23 1 8145 F
3 142638 34 2 11566 F
4 1575 0 3 13863 F
5 55360 13 4 18496 F
6 125011 30 5 22611 F
7 13220 3 6 25508 F
8 62682 15 7 29914 F
9 173075 42 0 1043 F
10 204379 49 1 7771 F
11 134818 32 2 11938 F
12 130519 31 3 15831 F
13 166568 40 4 19112 F
14 214457 52 5 21945 F
15 54647 13 6 25975 F
16 67070 16 7 30206 F
17 80866 19 0 3042 F
18 126482 30 1 7698 F
19 212946 51 2 12242 F
20 178509 43 3 14669 F
21 122940 30 1 4156 H
22 3347 0 4 19731 F
23 187924 45 5 24084 F
24 160287 39 6 25119 F
25 200604 48 7 32668 F
26 110460 26 0 3964 F
27 205321 50 2 8713 F
28 41964 10 3 13292 F
29 79759 19 4 18319 F
30 173475 42 5 21923 F
31 53501 13 6 24829 F
32 9821 2 7 30301 F
"output.sc" [noeol] 5002L, 385111C
1,1 Top


leejin@20212908: ~/HW4
4969 69657 17 5 20505 F
4970 109807 26 6 27887 F
4971 246691 60 7 29603 F
4972 78772 19 0 948 F
4973 140614 34 1 5446 F
4974 16313 3 2 12217 F
4975 140071 34 1 4903 H
4976 233725 57 3 12541 F
4977 198348 48 4 18124 F
4978 158323 38 5 23155 F
4979 88015 21 6 26575 F
4980 42762 10 7 30474 F
4981 98051 23 0 3843 F
4982 31065 7 2 10585 F
4983 215131 52 3 14427 F
4984 154172 37 4 19004 F
4985 129793 31 5 23297 F
4986 159897 39 6 24729 F
4987 19850 4 7 32138 F
4988 80974 19 0 3150 F
4989 105612 25 1 7308 F
4990 224078 54 2 11086 F
4991 40254 9 3 15678 F
4992 68280 16 4 19128 F
4993 134734 32 5 24142 F
4994 5938 1 6 26418 F
4995 94777 23 7 29241 F
4996 108563 26 0 2067 F
4997 49737 12 1 4681 F
4998 171689 41 2 11945 F
4999 2507 0 3 14795 F
5000 119394 29 4 16994 F
Total Number of Page Faults: 4,381
5002,1 Bot

[실행 결과 분석 내용]

각각의 알고리즘에 대한 총 Page Faults 의 개수는 다음과 같다.

적용된 알고리즘	Total number of Page Faults
Optimal	3,078
Fifo	4,393
LRU	4,395
Second-chance	4,301

Optimal Page Replacement Algorithm 은 모든 알고리즘 중 가장 적은 page fault 를 발생시킨다. 이론적으로 가장 효율적이다. 하지만 실제 시스템에서는 미래의 메모리 접근을 예측할 수 없기 때문에 실질적으로 구현할 수 없는 이상적인 모델이다.

Fifo Page Replacement Algorithm 은 가장 간단한 알고리즘 중 하나로, 메모리에 올라온 지 가장 오래된 페이지를 교체한다. 해당 방법은 구현은 쉽지만, 자주 사용되는 페이지를 교체할 위험이 있어 비효율적이다. 이를 Belady's Anomaly 라고 하며, 프레임 수를 늘려도 page fault 가 증가하는 현상을 일으킬 수 있다.

LRU Page Replacement Algorithm 은 가장 오랫동안 사용되지 않은 페이지를 교체한다. 그럼에도 Fifo 와 거의 같은 수준의 page fault 를 보이며, 이는 입력 파일에 생성된 주어진 가상주소들이 LRU 가 가정하는 패턴과 다를 수 있음을 시사한다.

Second-chance Page Relacement Algorithm 은 Fifo 와 LRU 보다 적은 page fault 를 발생시킨다. 참조 비트가 설정되어있는 페이지는 기회를 한 번 더 받아 메모리에 남게 되기 때문에 Fifo 보다 페이지의 사용빈도를 어느정도 고려한다.

종합적으로, Optimal 알고리즘이 가장 효율적이지만 실제 시스템에서는 구현할 수 없으며, 실제 시스템에서는 LRU 나 Second-chance 와 같은 알고리즘이 더 적합하다. 생성된 입력파일에서는 Second-chance 가 LRU 와 Fifo 보다 약간 더 나은 성능을 보이는 것으로 나타난다.