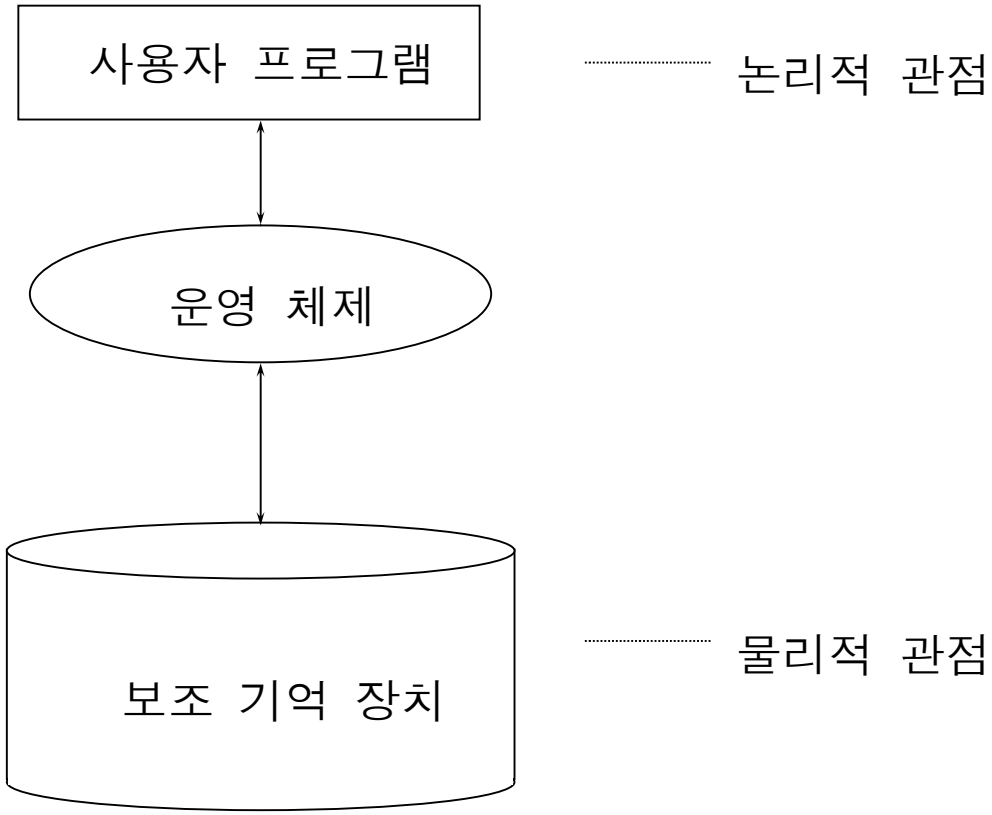
3. 화일 입출력 제어

❖ 입출력 제어 환경

◆ 운영 체제 (operating system)

– 다수 user를 위해 컴퓨터의 자원을 관리하는 S/W



▶ 운영 체제의 기능

- main memory manager
- process manager
- scheduler

- file management
  - 화일 조직 기법 제공
  - 사용자의 I/O 명령문 (예, READ/WRITE)
    → 지정된 저급 I/O 명령어(예, GET/PUT)로 변환
- device management
  - 물리적 기억장치에 대한 접근 제공

입출력 수퍼바이저 (I/O supervisor)
입출력 제어 시스템 (I/O control system)

- 입출력 제어 환경을 제공
- 사용자와 보조기억장치간의 **I/O** 제어 **:** 인터페이스
- 입출력 투명성 **(transparency) :** 논리적 관점을 물리적 관점으로 사상

▶ 입출력 제어 시스템의 기능

1) 화일 디렉토리(화일식별, 위치 정보)를 유지

2) 주기억장치와 보조기억장치 사이의 데이타 이동 통로(pathway)를 확립

3) CPU와 보조기억장치 사이의 통신 조정 기능
   - i) CPU와 보조기억장치 사이의 속도 차
   - ii) 송/수신자 사이의 데이타 전송 제어

4) 입/출력으로 사용될 화일 준비

5) 입/출력 완료후의 화일 관리

❖ 화일 디렉토리와 제어 정보

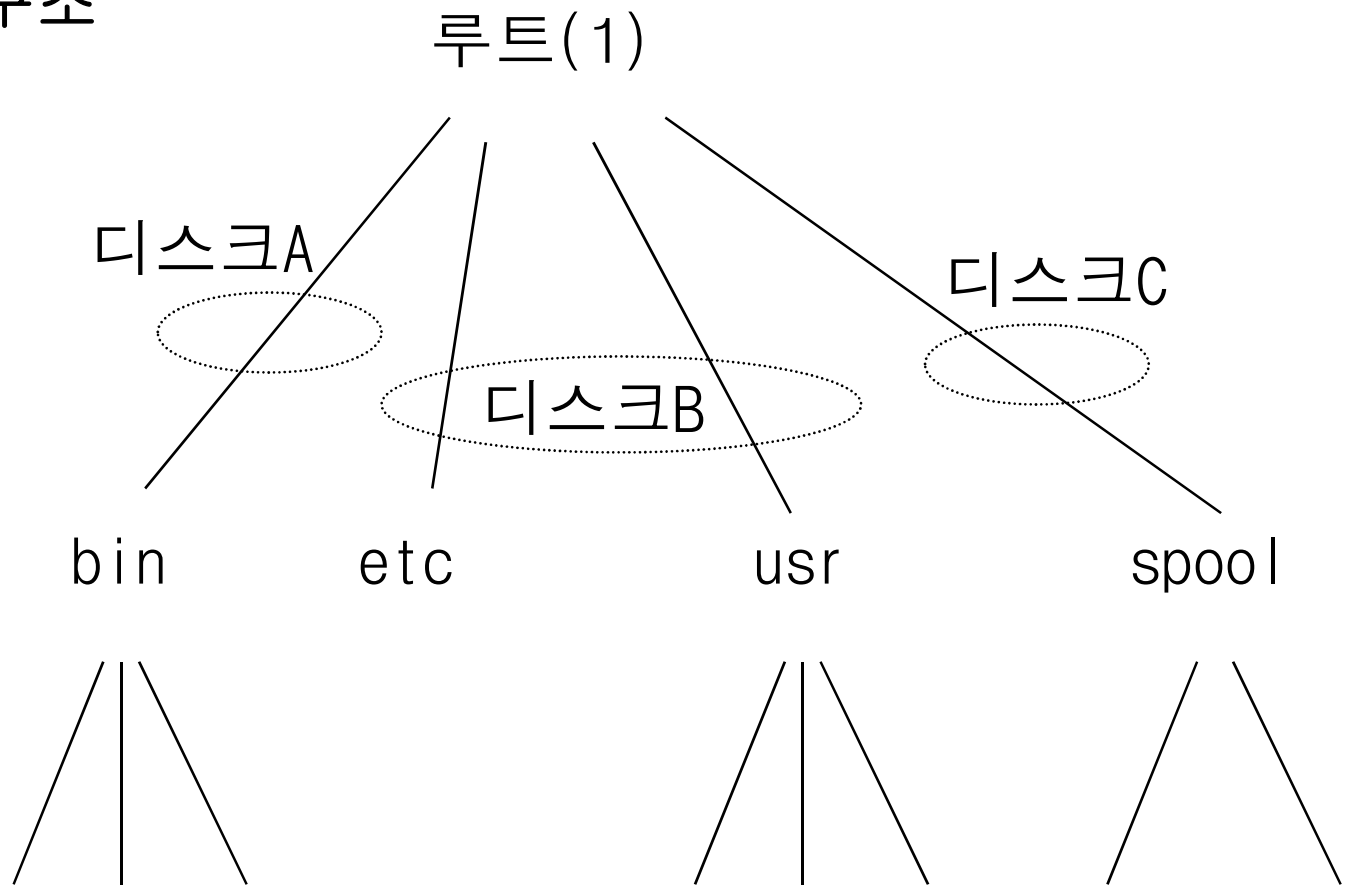
- **화일 관리 시스템 :**
  - 각 디스크 팩마다 : 장치 디렉토리나 VTOC가 존재
  - 장치 디렉토리 : 장치내의 모든 화일 정보
    - 화일의 이름, 위치, 크기, 형태
    - 저장 장치의 종류

- **화일 디렉토리의 구조(directory structure)**
  - 심볼 테이블 유지 : 화일에 이름 부여, 이름으로 탐색
  - 사용자 : 논리적 디렉토리와 화일 구조에만 신경
  - 시스템 : 화일의 공간할당문제를 담당

- **서브디렉토리 (subdirectory)**
  - 각기 다른 장치(디스크, 테이프)에 저장된 화일
  - 사용자 : 전체의 논리적 디렉토리 구조만으로 화일을 운영
  - 예: UNIX

▶ 디렉토리 구조의 예

- 계층구조



▶ 디렉토리 구조를 이용한 기본 연산

1) 탐색**(search) :** 특정화일을 찾기 위해 디렉토리 탐색

2) 화일 생성**(create file) :** 디렉토리에 첨가

3) 화일 삭제**(delete file) :** 디렉토리에서 삭제

4) 리스트 디렉토리**(list directory) :** 디렉토리 내용과
각 화일에 대한 디렉토리 엔트리의 값 표시

5) 백업**(backup) :** 신뢰도를 위해서,
테이프에 예비복사**(backup copy)** 유지

❖ 입출력 장치 제어

- 데이타를 판독/기록하기 위한 작업

  i) 요구된 화일의 위치 탐색 (디렉토리)

  ii) 주기억장치와 접근 장치 사이의 통로 설정

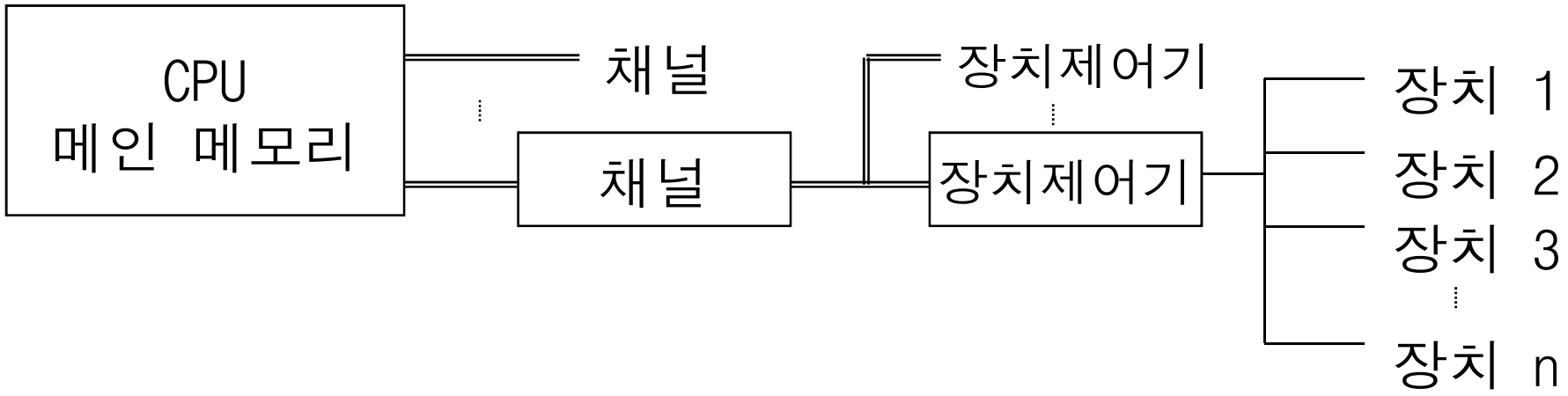
  iii) I/O 연산 신호를 보냄

★ 신호를 받은 장치
- 장치를 준비
- I/O 작업 도중의 오류에 대한 조치
- I/O 연산 수행후, I/O 요청 장치에 작업의 성공 여부 전달

▶ 입출력 채널(channel)

- 프로그램 가능 처리기 (programmable processor)
- 채널 프로그램(channel program)
  - 채널이 수행하는 프로그램
  - 장치 접근이나 데이타 경로 제어에 필요한 연산들을 지시함
  - OS에는 I/O채널을 가동시키는 표준 루틴이 포함되어 있음
- 채널의 종류
  - Selector Channel(고속I/O 장치), Multiplexor Channel(저속I/O 장치)

◆ 입출력 활동에 필요한 요소



- **입출력 채널 : CPU, 주기억장치 - 장치 제어 장치 및 각 보조 장치 사이의 중개자 역할**
  - 하나의 제어장치 : 동일한 장치들로 구성되어야 함
    - (예) 전부 디스크, 전부 라인프린터

- **제어 명령어 :** CPU → 채널

  i) 입출력 검사(TEST I/O) : 지정된 통로까지 사용 여부

  ii) 입출력 개시(START I/O)

  iii) 입출력 중지(HALT I/O)

- **인터럽트(interrupt) :** 채널 → CPU에게 작업 완료 통보
  - 인터럽트 발생 : 오류 검출시, I/O 작업 완료시
    - 인터럽트 발생 : OS는 인터럽트 처리 루틴으로 제어 전달, 인터럽트 발생 원인을 규명하고, 적절한 조치 후, 원래의 루틴으로 제어 반환함

❖ 화일의 입출력 (Write)

- 화일 기록 연산
  - 물리적 기록은 운영체제가 담당
  - 화일 관리자(file manager)
    - 화일에 관련된 작업과 입출력 장치를 취급하는 프로그램들
    - 화일 전송과 저장에 필요한 모든 프로시저
    - OS의 일부
  - I/O 버퍼
    - 디스크 내의 블록 판독을 위해 시스템 I/O 버퍼 확보
  - I/O 채널과 디스크 제어기
    - I/O 채널 : I/O 전문 처리 장치 (I/O를 위한 작은 CPU)
    - 채널 프로그램 : 버퍼 내의 데이타 존재 유무, 데이타 양, 저장 위치 지시 등을 위한 I/O 프로그램
    - 디스크 제어기 : 디스크의 실제 운영 장치

▶ 프로그램의 화일 Write 요청시의 작업

1) 프로그램의 **Write :** 운영 체제에 기록 연산 요청**(F<-r)**

2) 운영 체제 **:** 화일 관리자**(OS**의 일부**)**에게 지시

3) 화일 관리자 **:** 화일 **F**의 개방 여부**,** 접근 허용 여부**,** 물리적 화일 검사

4) 화일 관리자 **:** 블록의 물리적 위치 추적 **(**화일 할당 테이블 이용**)**

5) 화일 관리자 **:** 블록의 **I/O** 버퍼 내의 존재 유무 확인**,** 레코드 **r** 기록

6) 화일 관리자 **: I/O** 채널에 블록 위치와 디스크 기록 위치 지시

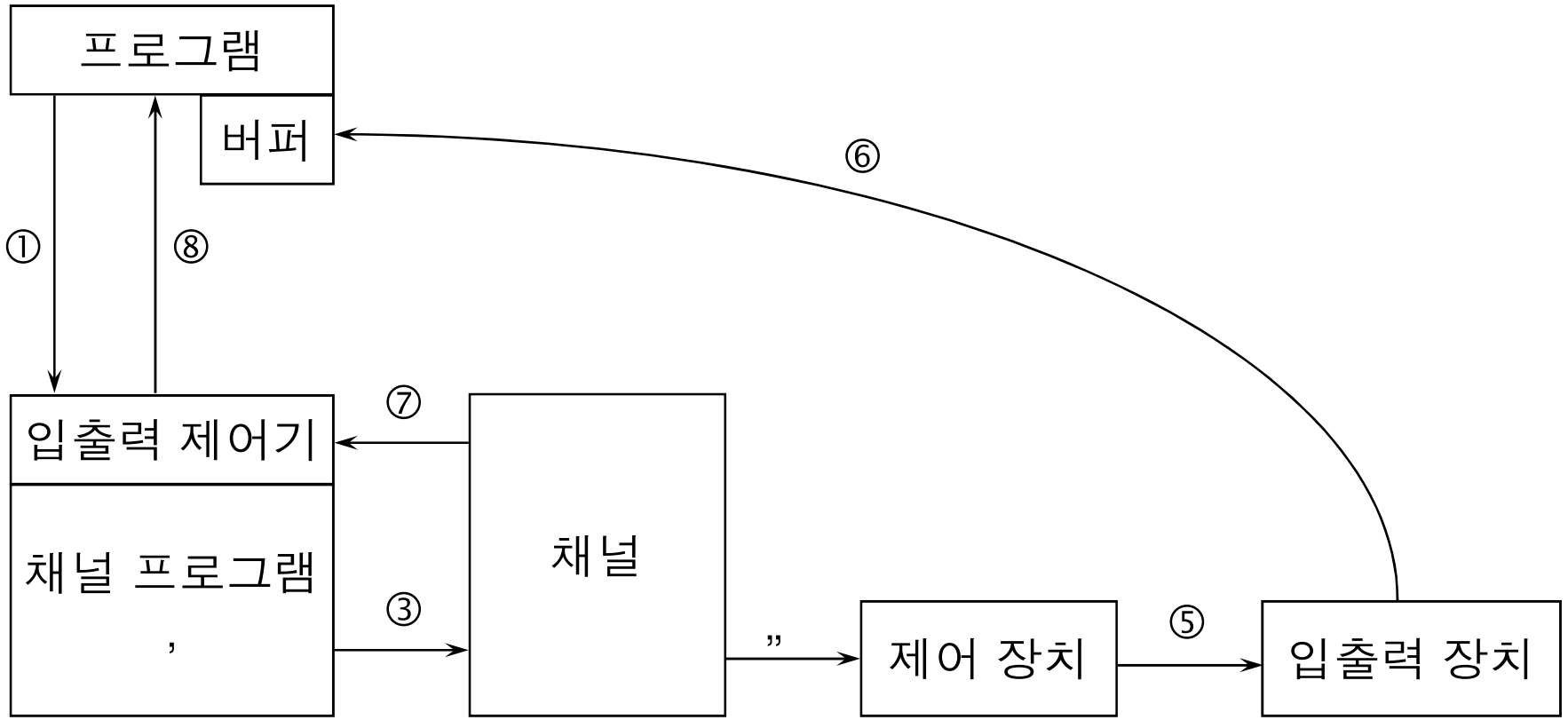
7) I/O 채널 **:** 디스크의 데이타 수신 가능 시간대와 데이타 형식 변환

8) I/O 채널 **:** 데이타를 디스크 제어기에 보냄

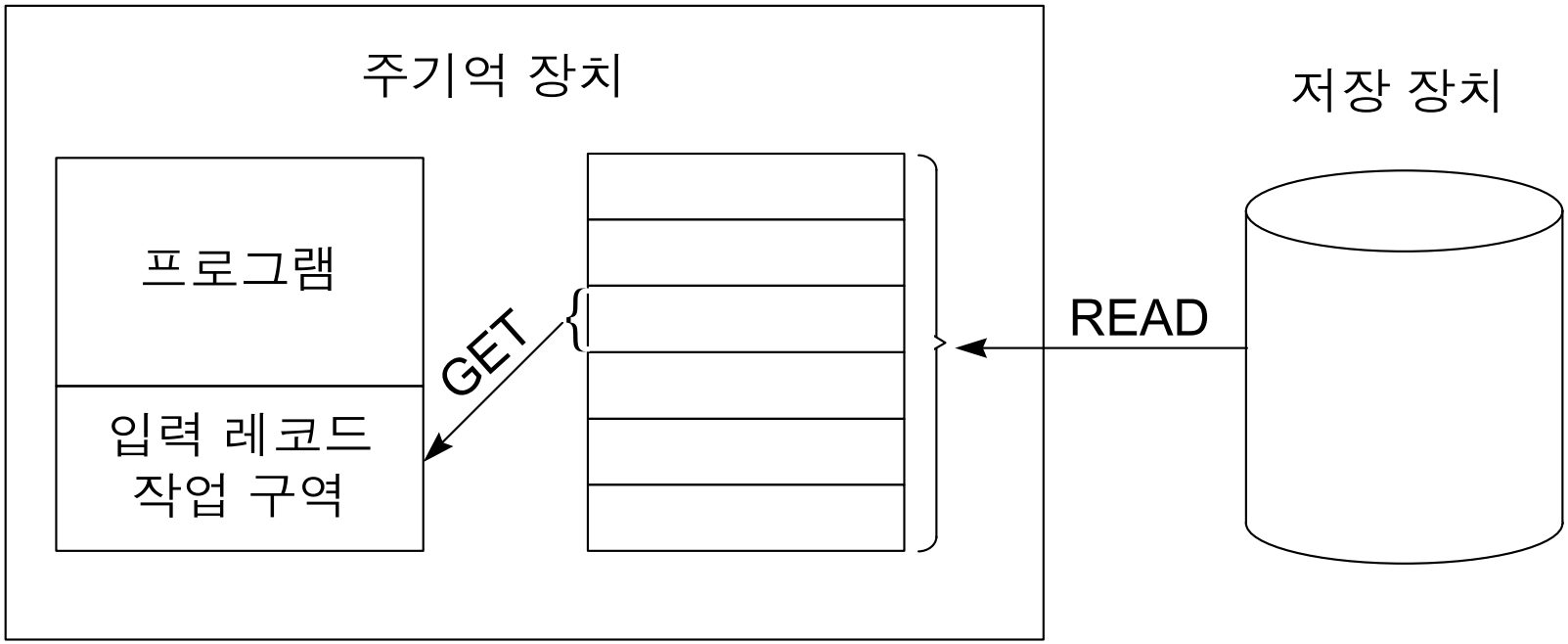
9) 디스크 제어기 **:** 헤드를 적절한 트랙으로 이동**,** 디스크에 저장

❖ 화일 입출력(Read)

◆ 프로그램의 화일 **READ** 요청시의 작업



▶ 입출력에서 버퍼의 활용



▶ 프로그램의 화일 READ 요청시의 작업

1) 프로그램의 **READ : I/O** 시스템에 인터럽트 발생

2) I/O 제어 시스템 **:** 주기억장치에 채널 프로그램 구성

3) 지정 채널은 이 채널 프로그램을 읽어서 실행

4) 지정된 제어 장치로 적절한 신호 전달

5) 신호 해석 **:** 요청 데이타를 판독할 장치의 연산을 제어

6) 데이타 **:** 경로따라 주기억장치의 버퍼 영역으로 이동

7) 채널 **:** 인터럽트를 걸어, 프로그램 수행을 재개하도록 **OS**에게 신호

8) 제어는 원래의 프로그램으로 반환

▶ 채널 명령어

- **I/O 채널**
  - 데이타의 교류 지시 (CPU의 I/O 명령 인계)
  - 비정상적인 상황에서는 인터럽트 발생

- **디스크에 대한 채널 명령어**
  - Search : 디스크 내의 적절한 정보 검색
  - Read : 레코드 판독, 메모리 내의 지정된 장소로 전송
  - Write : 메모리 내의 데이타를 디스크에 저장
  - Wait : 앞의 연산이 끝날 때까지 명령어의 실행 지연

- **출력 연산**

1) 데이타 전송 장치, I/O채널, 디스크 제어기 선정
2) CPU가 채널 프로그램 시작
3) I/O 채널 : 메모리 내의 데이타 요청
   디스크 제어기 : 데이타 전송 제어
4) 디스크 제어기 : 디스크 형식에 맞도록 데이타 코딩
5) 디스크 드라이브 : 데이타 기록

▶ 장치 제어기의 기능

- 채널 명령어의 실행 – **Search, read, Write** 등
- **I/O** 채널, 화일 관리자에게 상태 정보 제공
  - 장치 준비 여부, 데이타 전송 완료 등
- 호스트와 장치 사이의 데이타 형식 변환
  - 호스트 : 병렬 전송
  - I/O 장치 : 직렬 전송
- 데이타 전송 중 에러의 검사, 교정
  - 패리티 체크(parity check)
  - CC(cyclic check charaters), CRC, ECC 등

▶ 화일의 개방 (open)

– OPEN문(첫번째 READ 혹은 WRITE문과 연결 시행)

i) 오퍼레이터에게 테이프 릴이나 이동 가능한 디스크 등을 필요량만큼 준비 요구

ii) 필요한 채널 프로그램의 골격 구성

iii) 화일이 입력위해 개방되어 있는지 레이블 검사
- 출력을 위해 레이블을 다시 기록한다.

iv) 화일에 접근할 사용자의 권한 검사

v) 화일의 버퍼 구역을 구성하고 플래그에 적당한 초기값을 줌

vi) 입력 화일에 대해 예상 버퍼링을 하는 경우라면 첫번째 버퍼를 채움

vii) 시스템의 화일 디렉토리에 있는 화일 제어 블럭을 완성

▶ 화일의 폐쇄 (close)

– CLOSE문

– 묵시적(default)으로 프로그램이 끝났을 때 자동적으로 수행
– 나중에 다른 프로그램이 이 화일을 사용할 수 있게 준비

i) 출력 화일을 위한 버퍼 구역을 비움

ii) 버퍼 구역과 채널 프로그램이 차지한 구역을 반환

iii) 출력 화일에 화일 끝 표시(end-of-file mark)와 꼬리(tailer) 레이블을 기록

iv) 화일 기록 매체를 정리(rewind, dismount 등)

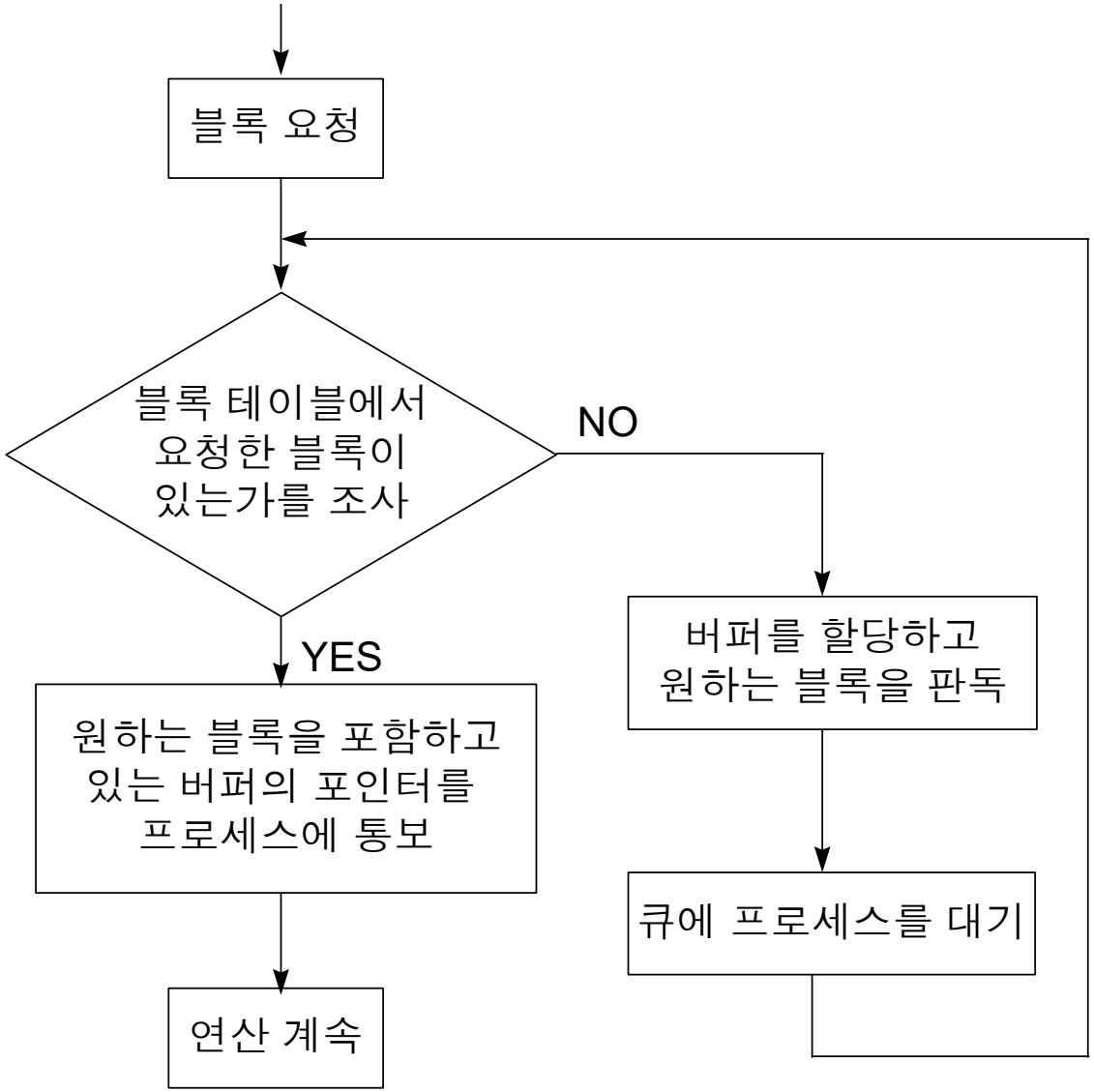
❖ 버퍼 관리

- **버퍼(buffer) :**
  - 화일에서 데이타를 읽어들이는 주기억장치 내의 일정 구역
  - 버퍼관리의 목적
    - CPU의 부담을 감소
    - 보조기억장치의 성능과 활용을 최대화

- **버퍼 관리자**
  - 제한된 주기억장치의 버퍼공간을 최적 분배
  - 사용자의 요구에 따라 버퍼 공간 할당
  - 사용하지 않는 주기억 공간을 관리
  - 버퍼 요구량이 할당 가능 공간을 초과시
    - 사용자 프로세스를 지연
    - 우선 순위가 낮은(또는 사용도가 낮은) 프로세스에 할당된 버퍼 공간을 회수

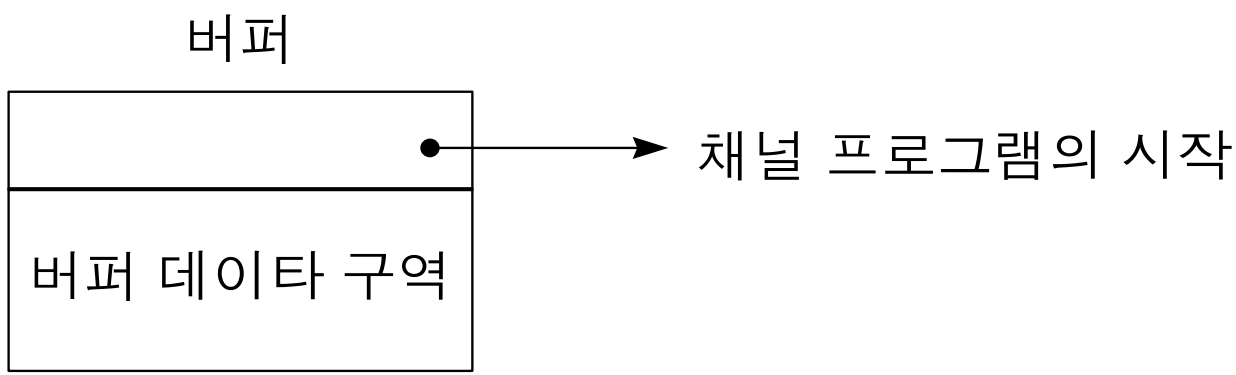
- **단편(fragmentation)에 의한 낭비 최소화하기 위해**
  - 버퍼의 크기와 한계를 OS의 페이지와 연관시키는 것이 좋다

▶ 블록 요청 처리 과정


블록 요청
블록 테이블에서
요청한 블록이
있는가를 조사
NO
YES
버퍼를 할당하고
원하는 블록을 판독
원하는 블록을 포함하고
있는 버퍼의 포인터를
프로세스에 통보
큐에 프로세스를 대기
연산 계속

[1] 단순 버퍼 시스템

- 버퍼의 데이타 구조
  - 가정
    - 1 record/block (Bf = 1)
    - 1 buffer/file
    - 프로그램의 요구에 따라 버퍼가 채워짐



<버퍼 구조>

▶ 버퍼를 채우는 채널 프로그램의 기본 구성

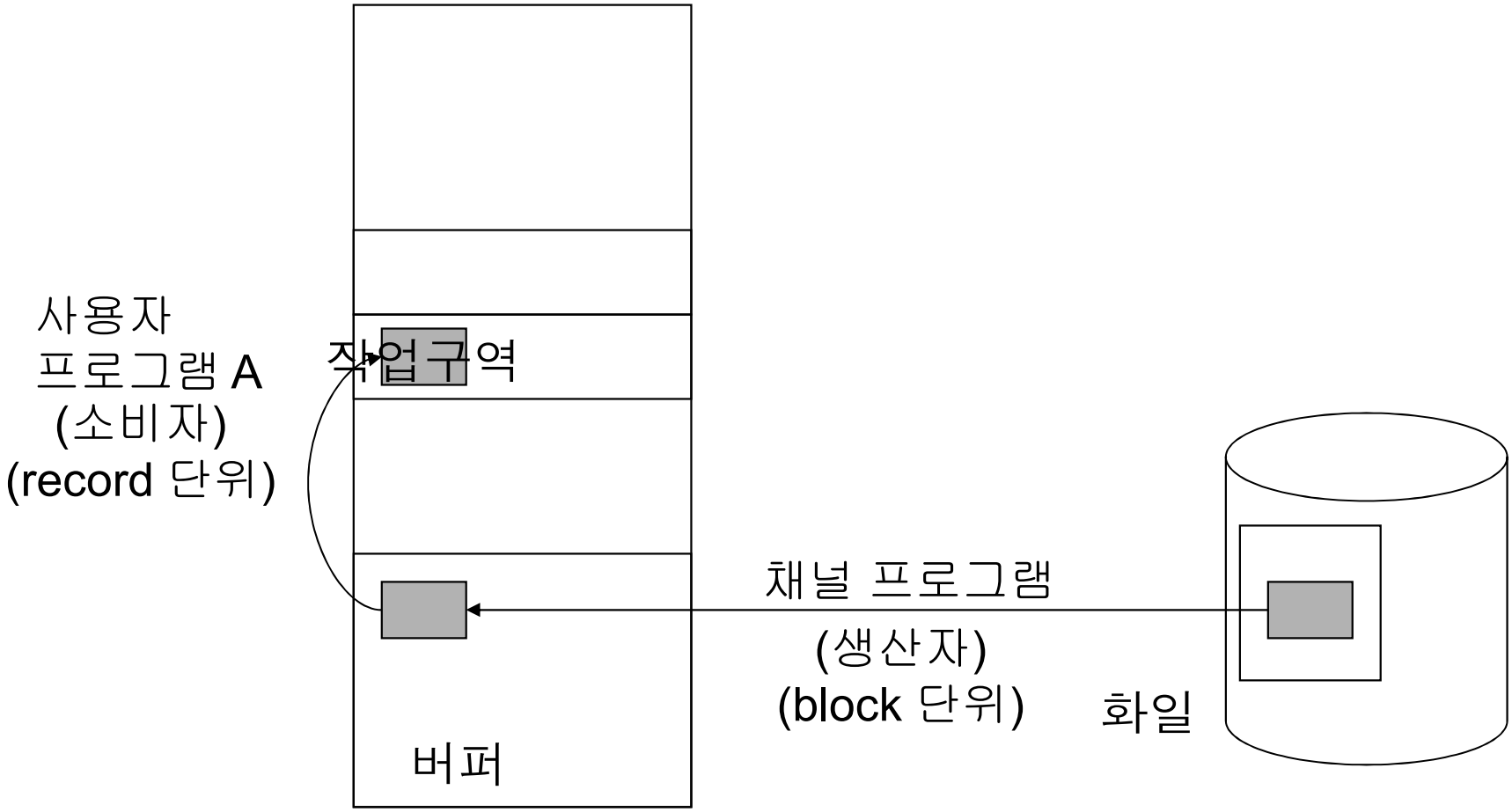
i) 프로그램의 **READ** 명령이 있을 때까지 기다림

ii) 제어 장치에 **I/O** 시작 명령을 내림

iii) 버퍼가 채워지기를 기다림

iv) 프로그램에 시켜서 버퍼로부터 데이타를 읽도록 인터럽트를 발생함

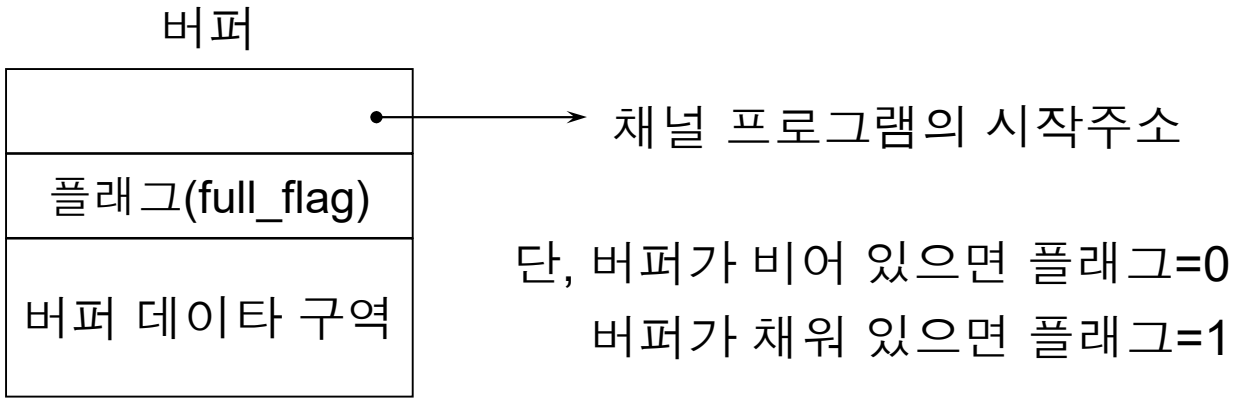
→ 사용자 프로그램은 버퍼가 찰 때까지 유휴 상태(idle)


사용자
프로그램 A
(소비자)
(record 단위)
작업구역
버퍼
채널 프로그램
(생산자)
(block 단위)
화일

▶ 예상 버퍼링 (anticipatory buffering)

- 프로그램이 대기 상태에 있을 가능성을 어느 정도 제거
- I/O 제어 시스템은 프로그램이 필요로 할 데이타를 미리 예측해서 항상 버퍼를 가득 채워 놓음
  - 프로그램은 버퍼가 채워질 때까지 기다릴 필요 없음

◆ 버퍼 구조



<예상 버퍼링을 위한 버퍼 구조>

▶ 예상 버퍼링의 채널 프로그램

- 생산자(**Producer**) 루틴

```
loop : if (full_flag == 1) goto loop;
                    //버퍼가 공백이 될 때까지 대기
       issue start-I/O command to control unit;
                    //디스크 제어기에 I/O시작 명령을 내린다.
       wait while buffer is being filled;
                    //버퍼가 채워지는 동안 대기
       full_flag = 1;
       goto loop;
```

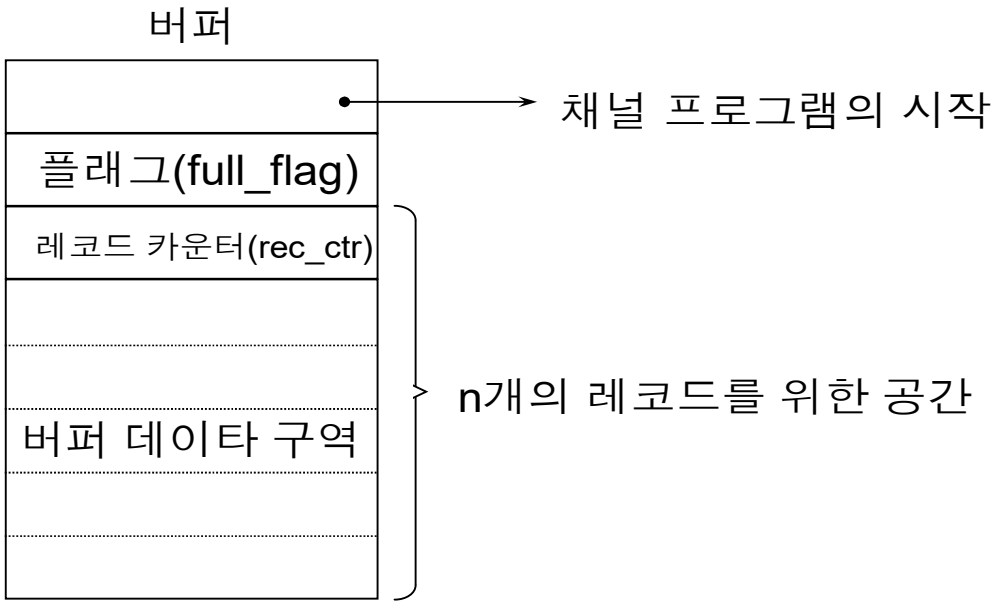
  - 입력 파일일 경우 : 생산자-채널 pgm, 소비자-사용자 pgm
  - 출력 파일일 경우 : 생산자-사용자 pgm, 소비자-채널 pgm

- 소비자(**Consumer**) 루틴

```
wait : if (full_flag == 0) goto wait;
                    //버퍼가 공백이면 대기
       read the buffer contents into the record work area;
                    //버퍼에 있는 레코드를 작업 구역으로 이동
       full_flag = 0;
       goto wait;
```

▶ Bf = n인 경우

- 생산자/소비자 루틴
  - 매 n+1번째 READ에만 장치 접근
- 버퍼 구조



단, 버퍼가 비어 있으면 플래그 = 0
그렇지 않으면 플래그 = 1
레코드 카운터 = 1, … , n

◆ 생산자 루틴

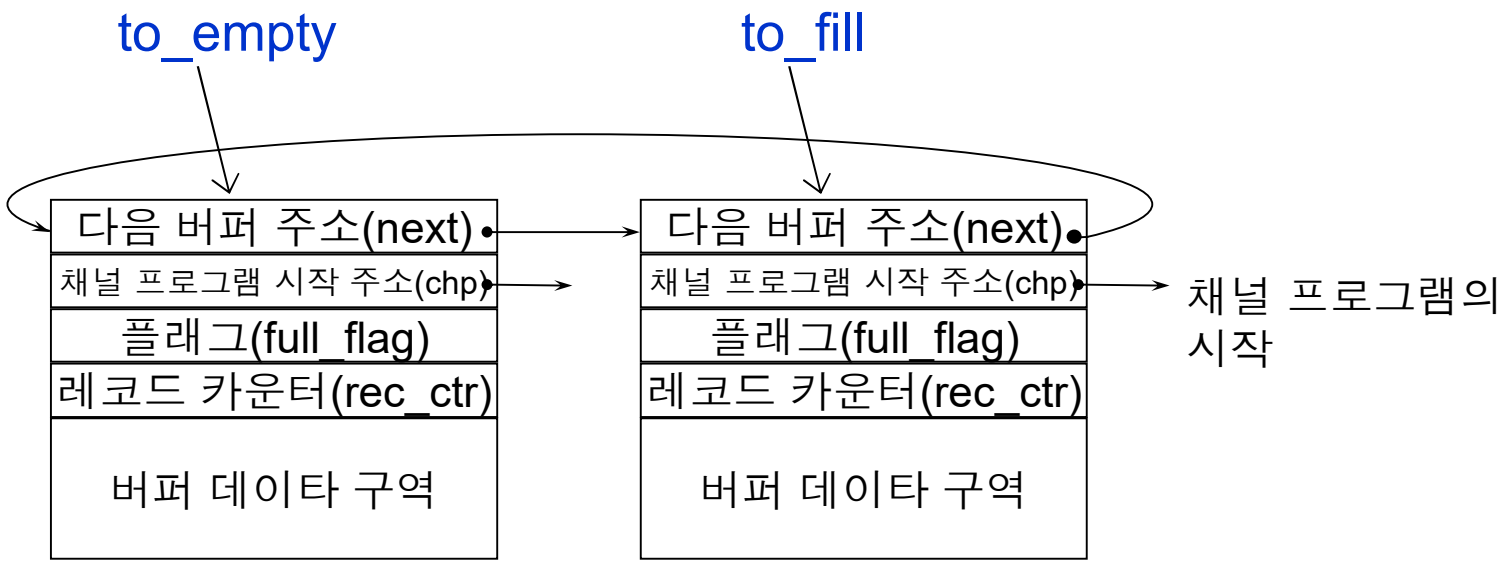
```
loop :  if (full_flag == 1) goto loop;
                //버퍼가 공백이 될 때까지 대기
          issue start-I/O command to control unit;
                //디스크 제어기에 I/O 시작 명령을 내린다.
          wait while buffer is being filled;
                //버퍼가 채워질 때까지 대기
          rec_ctr = 1;
          full_flag = 1;
          goto loop;
```

◆ 소비자 루틴

```
wait :  if (full_flag == 0) goto wait;
          read record(rec_counter) into the record work area;
                // recored_counter가 지시하는 레코드를 작업 구역으로 이동
          rec_counter = rec_counter + 1;
        if (rec_counter > n) full_flag = 0;
                //n개의 레코드를 모두 처리해서 버퍼가 공백이 된 경우
          goto wait;
```

[2] 이중 버퍼 시스템 (double buffer system)

- 화일당 두 개의 버퍼 구역을 할당
  - 하나를 비우는 동안 나머지 버퍼를 채움



단, 버퍼가 비워 있거나 채워지고 있는 중이면 full_flag=0

버퍼가 채워졌거나 비워지고 있는 중이면 full_flag=1

rec_ctr=1, ⋯ , n

- to_fill : 현재 채워지고 있거나 다음에 채워야 할 버퍼의 포인터
- to_empty : 현재 비워지고 있거나 다음에 비워져야 할 버퍼에 대한 포인터
- 생산자는 to_fill이 가리키는 버퍼를 채움
  - (초기→ 버퍼 1)
- 초기 상태 : 두 버퍼가 모두 공백
  플래그는 모두 0

◆ 생산자

```
loop :    if (to_fill.full_flag == 1) goto loop;
                  //to_fill.buffer가 공백이 될 때까지 대기
          issue start-I/O command to control unit;
                  //디스크 제어기에 I/O 시작 명령을 내린다.
          wait while to_fill.buffer is being filed;
                  // to_fill.buffer가 채워질 때까지 대기
          to_fill.rec_ctr = 1;
          to_fill.full_flag = 1;
          to_fill = to_fill.next_buffer;
                  // to _fill은 다음에 채워져야 할 버퍼를 지시
          goto loop;
```

◆ 소비자

```
wait :  if (to_empty.full_flag == 0) goto wait;
                //to_empty.buffer가 채워질 때 까지 대기
        read record[to_empty.rec_counter] into the record work area;
                //to_empty.record_counter가 지시하는 레코드를 작업 구역으로
    이동
        to_empty.rec_counter = to_empty.rec_counter + 1;
        if (to_empty.rec_counter > n)
                //n개의 레코드를 모두 처리해서 공백이 된 경우
         {
                to_empty.full_flag = 0;
                to_empty = to_empty.next;
                        //to_empty는 다음에 비워야 할 버퍼를 지시
         }
        goto wait.
```

❖ Unix에서의 입출력

- **UNIX** 에서는 화일을 단순히 바이트의 시퀀스로 가정
  - 디스크 화일, 키보드, 콘솔 장치도 화일로 취급

- 화일 기술자 **(file descriptor)**
  - 정수로 표현(0, 1, 2 등)
  - 화일 세부 정보를 저장한 배열의 인덱스 역할
  - 키보드(표준 입력 화일, STDIN) : 0
  - 출력화면(표준 출력 화일, STDOUT) : 1
  - 표준 에러 화일(STDERR) : 2
  - 사용자가 개방한 화일들 : 3 부터 부여

- 프로세스와 커널
  - 프로세스 : 동시에 실행 가능한 프로그램 (최상위 I/O)
  - 커널(kernel) : 프로세스 이하의 모든 계층 통합
    - I/O를 일련의 바이트 위에서의 연산으로 본다

▶ 커널의 I/O 시스템이 관리하는 테이블

- **화일 기술자 테이블(file descriptor table)**
  - 각 프로세스가 사용하는 화일 기록 테이블
- **개방 화일 테이블 (open file table)**
  - 현재 시스템이 사용중인 개방된 화일에 대한 엔트리 저장
  - 각 엔트리 : 판독/기록 허용 여부, 사용 프로세스 수, 다음 연산을 위한 화일 오프셋, 일반 함수들의 포인터 저장
- **인덱스 노드 (index node), 화일 할당 테이블(file allocation table)**
  - 화일의 저장 위치, 크기, 소유자 기록
- **인덱스 노드 테이블(index node table)**

▶ 화일 기술자 테이블과 개방 화일 테이블

화일
기술자 테이블
(프로그램당 하나)

화일 기술자	개방화일 테이블 엔트리
0(키보드) 1(화면) 2(에러) 3(일반화일) 4(일반화일) …	

개방 화일 테이블
(UNIX 전체에 하나)

R/W 모드	화일사용 프로세스수	다음접근 오프셋	Write 루틴포인터	…	inode 테이블 엔트리
… write …	… 1 …	… 100 …	… … …	… … …	… …

Inode 구조

inode 테이블의
한 엔트리

소유자 ID
장치
그룹 이름
화일 유형
접근 권한
화일 접근 시간
화일 수정 시간
화일 크기 (블록수)
블록 카운트
화일 할당 테이블

화일 할당
테이블

데이타 블록번호 0
데이타 블록번호 1
…
데이타 블록번호 9

▶Unix에서의 화일 입출력

– 예) 화일 기술자 값이 3인 화일의 레코드 판독 명령

1) 프로그램의 화일 기술자 테이블에서 개방 화일 테이블을 이용, 개방 화일 테이블의 해당 엔트리 검색

2) 개방 화일 테이블에서 inode 테이블 포인터를 이용, inode 테이블의 해당 엔트리 검색

3) Inode 테이블에서 해당 화일의 데이타가 저장된 디스크 블록의 주소를 얻어 디스크에서 데이타 블록 판독

▶화일 이름과 디스크 화일의 연결

- 디렉토리 구조와 화일의 개방
  - 화일을 식별하는 inode 번호와 그에 해당하는 화일 이름들을 데이타로 저장한 화일
  - Inode에 대한 포인터는 화일에 대한 모든 정보 참조

- 화일 이름과 **inode**
  - 여러 화일 이름이 같은 inode를 포인터로 가리킬 수 있다
  - 한 화일 이름이 삭제되더라도 포인터 수만 감소

화일 이름	inode 번호
…	…

Unix 디렉토리 화일