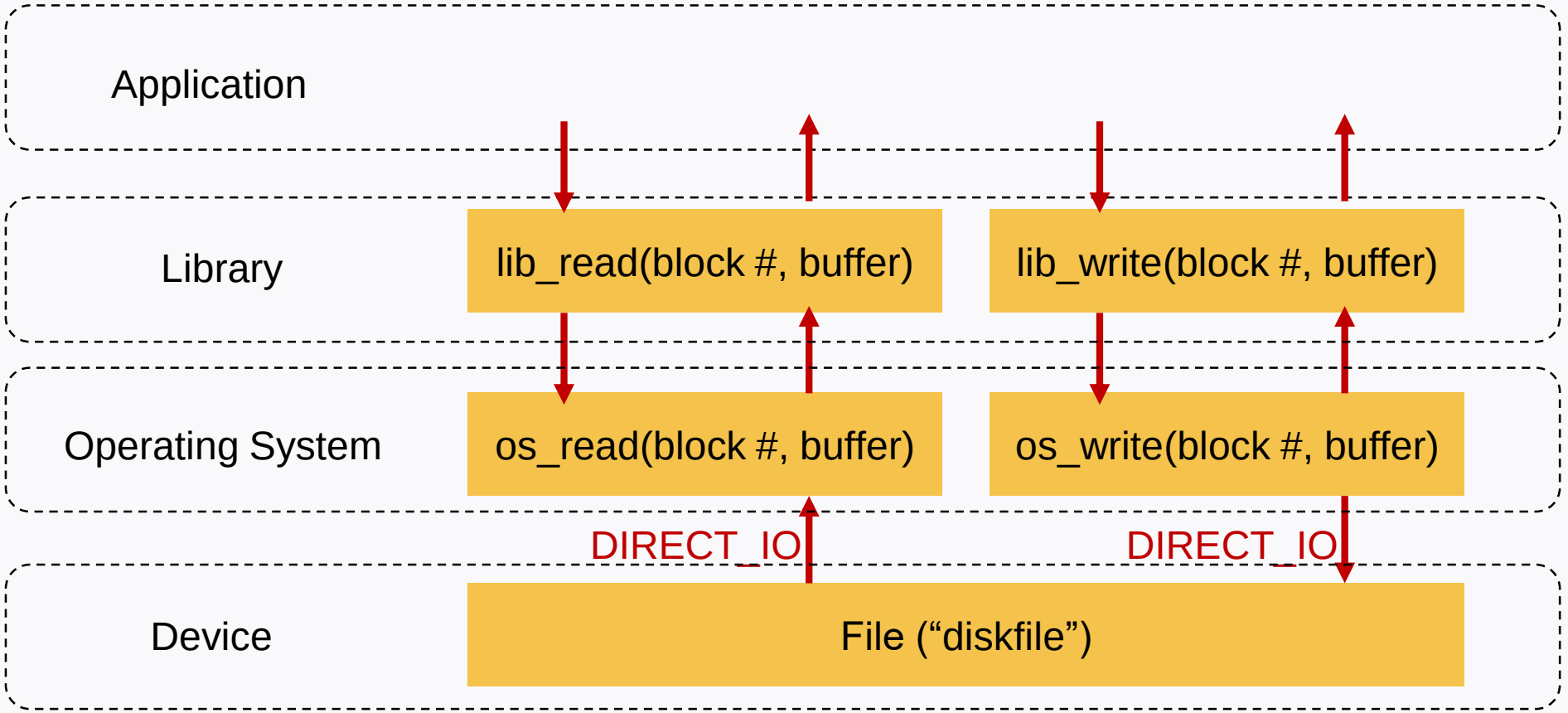
임베디드 운영체제 실습 과제
Buffer Cache 구현

Buffer Cache 구조

- 아래의 구조를 갖는 Buffer Cache Simulator를 개발한다.



Buffer Cache 구현 요구사항 (1)

- Application ~ OS 계층은 실제로는 하나의 응용으로 구현한다.
- Device는 diskfile 이름의 파일을 생성하여 사용한다.
  - Linux의 dd command를 사용하여 적당한 크기의 파일을 생성할 것
- main() 함수에서 lib_read() 및 lib_write() 함수를 호출할 것
- Library & OS의 read/write 함수는 블록 번호와 Application Buffer의 주소를 인자로 받음
  - 블록은 4KB로 하며, disk는 4KB의 연속된 저장공간으로 한다.

Buffer Cache 구현 요구사항 (2)

- 다음의 세 가지 기능을 구현한다. 각 기능이 잘 동작함을 보여야 하며 방법은 고민해 볼 것

1) Buffered Read
   - OS 레이어(os_read, os_write 함수)에 버퍼 기능을 구현한다.
   - 버퍼는 메모리를 사용하며, 총 N개를 관리할 수 있도록 한다.
   - 버퍼 각각은 하나의 블록 내용을 저장(4KB)할 수 있으며, 데이터 저장 공간과 메타데이터 (블록 번호, *빠른 탐색을 위한 데이터*) 저장 공간으로 구성된다.
   - *빠른 탐색을 위한 데이터 구조 (e.g., hash)*는 자유롭게 정한다.
   - 읽기 요청 블록이 Buffer에 없다면 Disk에서 Buffer로 읽은 후, Buffer의 내용을 상위 레이어에 전달한다. 요청한 블록이 Buffer에 있다면 파일에서 읽지 않고 Buffer의 내용을 전달한다.

Buffer Cache 구현 요구사항 (3)

2) Delayed Write

- 응용이 Write한 내용을 Buffer에 기록하고 리턴한다.
- 모든 Buffer가 사용 중인 경우, *Victim을 선정*하여 Buffer에서 방출시킨 후, 빈 Buffer를 만들어 기록한다.
- Buffer에 Write된 내용이 Disk에 쓰여지지 않았다면 Dirty 상태로 표시한다.
- Dirty 상태의 Buffer가 Victim으로 선정된 경우, Disk에 내용을 쓴 후, 회수할 수 있다.
- Dirty Buffer를 Disk에 Write하는 기능을 Thread (flush thread)를 이용하여 해볼 것
  - Buffer가 flush thread에 의해 Disk에 Write 중인 경우, 응용이 overwrite하지 못하도록 동기화 (locking) 메커니즘을 사용할 것

Buffer Cache 구현 요구사항 (4)

3) Replacement Policy

- Buffer 부족 시, Victim을 선정하는 알고리즘에는 여러 가지가 있다.
- FIFO, LRU, LFU 등 널리 알려진 알고리즘을 3개 이상 구현한다.
- Normal distribution 또는 Zipfian Distribution을 따르는 Block Access Sequence를 생성하고, 구현한 Replacement Policy에서 Hit가 얼마나 발생하는 지 테스트해 본다. 여기서 Block Access Sequence는 Block Read만 포함하도록 한다.