❓ Disk Manager 2주차 분석 QnA

1. `storage/disk_manager.c` 에서 Heap File 관련된 코드를 보았는데, 이 코드들이 어떤 맥락에서 작성되었고, 역할이 무엇인지 궁금합니다.

```
630
631     /* Initialize the system heap file for booting purposes. This field is reseted after the heap file is created by the
632      * boot manager */
633
634     vhdr->boot_hfid.vfid.volid = NULL_VOLID;
635     vhdr->boot_hfid.vfid.fileid = NULL_PAGEID;
636     vhdr->boot_hfid.hpgid = NULL_PAGEID;
637
```

> **Mentor**
> 보통 모듈의 진행방식은 다음과 같습니다.
>
> heap file 생성(record) → file manager를 통한 file 할당(큐브리드만의 논리구조입니다. Not OS file) → disk_manager를 통한 섹터 예약
>
> 이렇게 진행됩니다. 따라서 볼륨을 새롭게 초기화하는 맥락에 있어서 page 및 file(not OS file) 식별을 하기 위한 변수들입니다.

2. `log_append_*` 라는 prefix로 적힌 코드들의 역할이 궁금합니다. (`log_append_redo_data`, `log_append_undo`, `log_append_dboutside_redo` 함수들이 영구 볼륨 타입일 때 호출이 되던데 왜 호출을 하게 되는 것인지 감이 오지 않습니다.)

> **Mentor**
> 영구 볼륨 타입일 때 호출 되는 이유는 한마디로 log recovery를 위해서 log가 저장되는 패턴으로 이 해하시면서 코드 분석을 진행하시면 좋습니다.
>
> **Team Leader**
> log_append_redo_data(thread_p, RVDK_FORMAT, ...)를 이용되는 구문이 offset = -1과 offset = 0 두 군데서 나타나는데 차이가 있냐는 추가 질문의 답변은 아래와 같습니다.
> - 같은 형식의 로그 레코드를 남기는 것이 맞습니다.
>
> addr.offset = -1 과 addr.offset = 0의 차이
> - 데이터베이스 복구 과정에서 같은 로그를 보게 되면 같은 동작을 수행하게 됩니다. 위에서 똑같은 로그를 남겼기 때문에 addr.offset == -1인 경우에는 복구 동작을 생략하기 위한 용도로 사용됩니다.
> - 혹시나 이런 코드가 왜 남겨져 있는지 (혹은 남아 있는지)가 궁금할 수 있기 때문에 이에 대해서도 답변 드리겠습니다. 해당 코드를 제거하기 위해서는 많은 검증이 요구되기 때문에 제거하지 못하고 남겨둔 코드로 추측됩니다. 검증이 완료된다면 제거 대상이 될 수 있습니다.

3. 각 함수들에서 사용되는 `THREAD_ENTRY *thread_p`이 존재 이유와 어떤 역할을 하는 것인지 궁금합니다.

> **Mentor**
> OS 관점에서 이해하면 좋습니다.
>
> - 간단하게 설명하면, 같은 코드를 여러 스레드가 병렬로 실행할 수 있다는 점이 가장 큰 장점입니다.
> - 여러 스레드는 여러 유저가 될 수도 있고,여러 실행 루트가 될 수도 있습니다.
> - 자세한 내용은 멀티스레딩을 참조하길 바랍니다.

4. `storage/file_io.c` 의 `fileio_format` 함수 내에서 볼륨 헤더로 사용할 공간을 `malloc_io_page_p`라는 이름으로 하나의 페이지 크기만큼 할당하는 것을 볼 수 있었습니다. 추후에 볼륨 헤더의 정보들을 `addr.pgptr`로 접근해서 기록하는 것으로 볼 수 있었는데, `malloc_io_page_p`를 `addr.pgptr`로 접근 할 수 있는 이유에 대해서 명확하게 파악이 되지 않습니다. 찾아보니 `fileio_write_or_add_to_dwb`라는 함수에서 `malloc_io_page_p`를 이용하던데, 혹시 이 함수와 관련이 있는 것인지 궁금합니다. 또한 `addr.pgptr`이 참조하는 공간이 정확히 어느 위치의 어떤 것인지 궁금합니다.

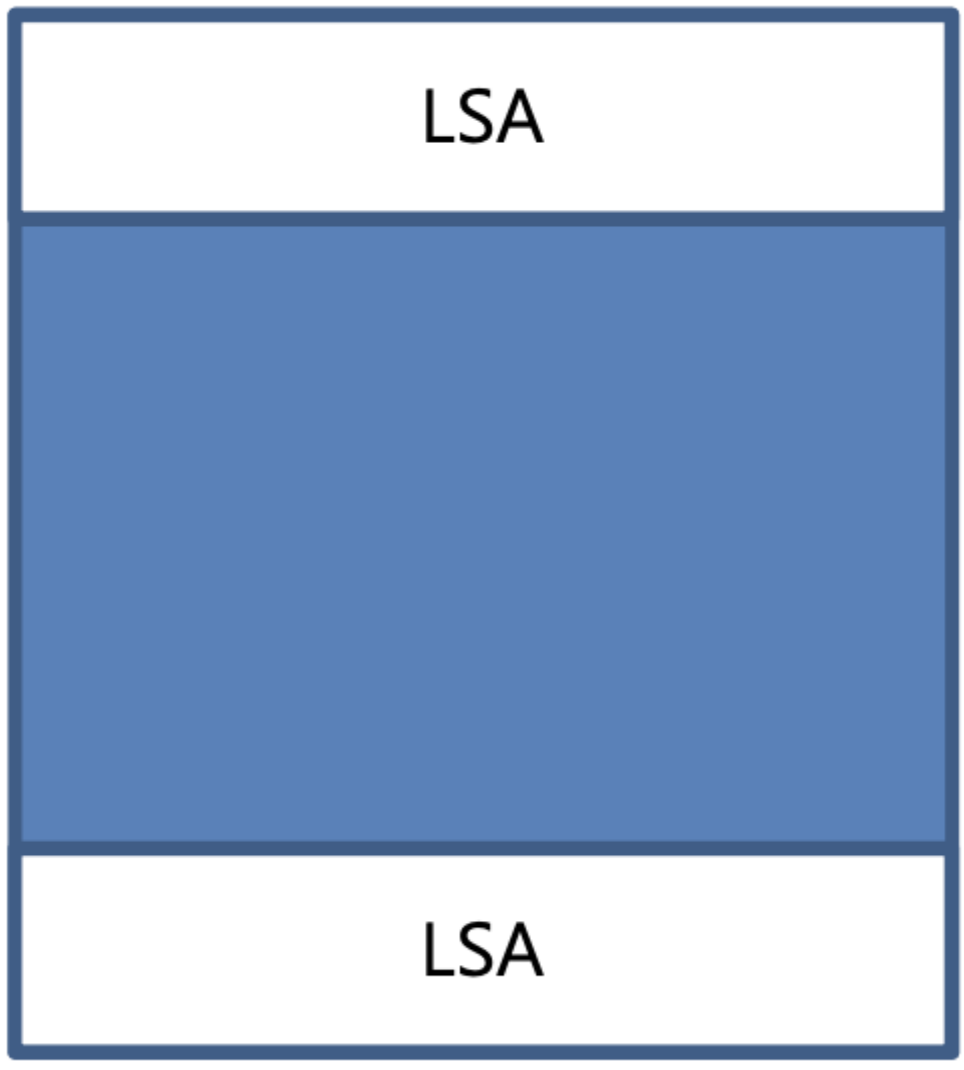
```
2400    malloc_io_page_p = (FILEIO_PAGE *) malloc (page_size);
2401    if (malloc_io_page_p == NULL)
2402      {
2403        er_set (ER_ERROR_SEVERITY, ARG_FILE_LINE, ER_OUT_OF_VIRTUAL_MEMORY,
2404        return NULL_VOLDES;
2405      }
```

> **Team Leader**
> fileio_ 부분에서 dummy page를 디스크에 쓰는 이유는 크게 2가지와 같습니다.
>
> 1. 페이지 포맷팅한 내용을 기록하기 위해서
> 2. 실제 디스크 파일에 공간을 확보하기 위해서
>
> 입니다. 특히 2번이 보장되지 않으면, 디스크 공간이 충분하지 않을 때 문제가 될 수 있습니다.
>
> **Mentor**
> “malloc_io_page_p 를 addr.pgptr 로 접근 할 수 있는” 부분이 어디인지 정확히 알려주면 답변에 도움이 더 잘될거라 생각합니다.
>
> 참조하는 공간이 정확히 어디인지 모르겠습니다. (물리적인 저장장치, module, 등등) DB를 생성하면 만들어지는 로그 파일에 기록되는 것으로 알고 있습니다.
>
> **Mentee**
> storage/disk_manager.c의 597라인에서 (disk_format 함수 내) addr.pgptr이 참조하는 주소를 vhdr로 할당하는 것을 볼 수 잇었습니다. 그리고 그 이후의 /* initialize the header */ 부분에서 vhdr의 필드들을 설정하는 것을 볼 수 있었습니다. (vhdr이 DISK_VOLUME_HEADER * 인 것을 미루어 보아, 볼륨 내 첫 섹터의 볼륨 헤더를 말하는 것으로 이해했습니다.)
>
> 볼륨 헤더는 1개의 페이지라고 이해하고 있었고 별도의 addr.pgptr에게 1개의 페이지만큼 할당하는 구문을 찾지 못했던 것을 생각했을 때, 유일하게 storage/file_io.c의 fileio_format 함수에서 malloc_io_page_p가 1개의 페이지 크기 만큼 할당 받는 것을 볼 수 있었습니다.
>
> 그래서 addr.pgptr이 malloc_io_page_p와 혹시 관련이 있는가 싶었고, fileio_write_or_add_to_dwb라는 함수가 혹시 malloc_io_page_p 가 참조하는 공간을 addr.pgptr과 연결되는 부분이 있는지 궁금했습니다.
>
> 만일 malloc_io_page_p와 addr.pgptr이 서로 관계 없다면, 각각이 어떤 역할을 하는 값인지 궁금합니다.
>
> **Mentor**
> 588 라인에서 addr.pgptr에 페이지 할당을 진행합니다.
>
> 위에서 질문한 fileio_write_or_add_to_dwb는 크게 관련이 없지만, disk_manager 볼륨과는 관련이 있습니다.

> LOG_DATA_ADDR addr는 file_io가 일어나고 vol header page가 생성된 사실을 log에 기록합니다.
> FILEIO_PAGE * malloc_io_page_p는 실제 페이지를 저장하는 메모리 공간입니다. 다른 점을 찾자면, addr.pgptr에 저장하는 건 페이지의 주소값입니다.

5. `storage/page_buffer.c` 의 `pgbuf_set_lsa_as_temporary` 함수 (`disk_format` 함수에서도 이용되는) lsa가 Log Sequence Address라는 것을 볼 수 있었습니다. lsa가 무엇인지 궁금합니다. 또한 lsa가 어느 위치에 저장되어 관리되는지 궁금합니다.

> **Mentor**
> 여기서의 lsa는 Page LSA입니다.
>
> - 페이지가 저장된 순서 및 페이지의 validation을 체크합니다.
> - 아래와 같은 구조로 구성되어있습니다. 페이지 구조의 시작과 끝 부분에 있습니다.



6. (5번과 관련지을 수 있을 것 같은데...) `storage/disk_manager.c`의 라인 620번 근처에서 `log_get_db_start_parameter` 함수의 인자로 데이터베이스 생성시간과, LSA 체크 포인터를 받아오는 것을 볼 수 있었습니다. 어떤 맥락에서 두 인자가 사용되는 것인지 궁금합니다.

> **Mentor**
> **질**문에 대해서 다시 찾아보고 답변하겠습니다.

7. 볼륨 헤더를 작성하고, STAB을 기록한 후에는 이를 디스크에 반영을 해주는 과정이 있어야 한다고 생각을 했습니다. 이와 같은 과정을 `pgbuf_flush_all`, `dwb_flush_force` 등으로 수행해주는 것 같은데 두 함수가 어떤 작업을 하는 것인지 잘 이해가 되지 않습니다. 또한 마지막에는 `fileio_synchronize`까지 호출하면서 동기화를 해준다는 것까지 확인을 했는데, 이것이 무엇과 무엇의 동기화인지도 궁금합니다.

> **Mentor**
> 실제로 memory에서 disk로 flush 할때는 Write & Synchronize라는 두 과정을 거치게 됩니다.
>
> Write and synchronize : buffer에 write한 다음, 그 buffer를 disk에 물리적으로 저장하는 게 synchronize 입니다.(os 별 함수 이름이 조금씩 상이합니다.)

> pgbuf_flush_all , dwb_flush_force의 질문은 해당 함수의 호출 위치를 정확하게 써줘야 정확한 답변이 가능합니다. 왜냐하면 write 시점에 따라 이전에 있는 data를 flush하고 새로운 데이터를 받을지 혹은 새로운 파일을 flush 할지 다르기 때문입니다.
>
> dwb_flush 함수는 자세히 알고 싶으면 제가 공유한 브리핑을 참고하시면 되겠습니다.(force라는 단어가 들어갔으니 강제로 flush하는 것입니다. 이전에 있는 data를 flush하고 새롭게 쓰는 것으로 판단 할 수 있습니다.) pgbuf_flush_all는 DWB를 사용하지 않는다면 진행하는 route입니다.
>
> 간단히 설명하면 두 함수 다 disk에 flush 하기 위해서(write+sync) 호출합니다.
>
> fileio_synchronize는 flush를 통해 메모리와 실제적으로 동기화 해준다는 의미입니다.