임베디드 시스템 실습 PA1

2016310936 우승민

PA1의 목표는 key-value ssd를 구현하는 것입니다. 우선 제가 사용하는 data management 방법을 간단하게 설명하겠습니다. metadata에서는 총 3가지 array를 통해 data의 정보를 확인합니다.

1. key\_pos array : array에 순서대로 사용되는 key값을 입력 받아 key의 existence를 판단합니다.

2. key array : key\_pos와 동일한 index를 사용함으로써 key와 block을 연결하는 역할을 합니다.

3. block array : block의 state를 unused/valid/invalid 값을 각각 0/1/2로 저장하여 block의 사용 유무를 판단합니다.

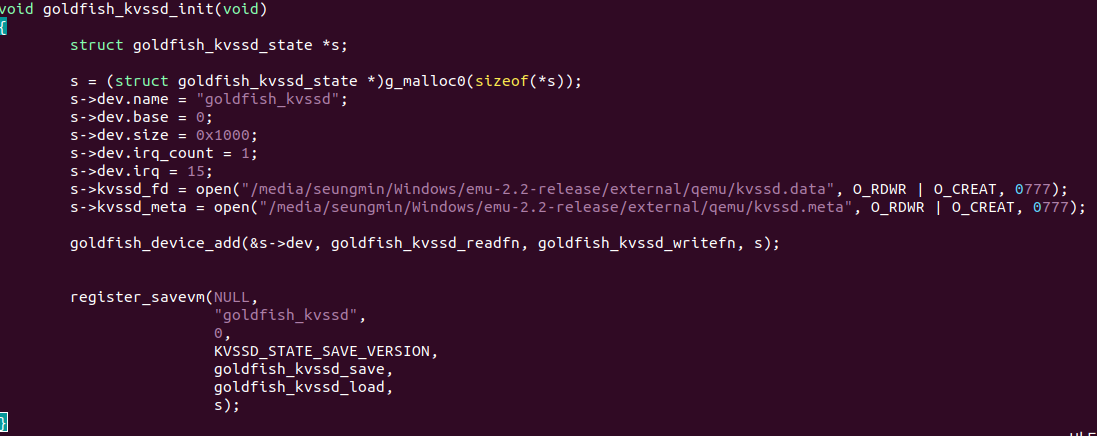
이 metadata를 바탕으로 QEMU에서 매 command 실행할 때마다 metadata의 정보를 읽고 key\_pos array를 통해 key의 existence를 판단한 후 key array를 통해 해당 key와 연결된 block의 위치를 찾고 kvssd.data에서 해당 위치의 data를 읽고 쓰는 역할을 수행합니다.

저는 key의 값을 입력 받는 과정은 key 16바이트를 모두 더하고 1을 추가로 더해주었습니다.

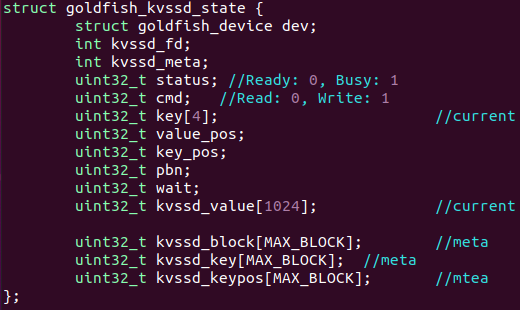
* key(real) = key[0]+key[1]+key[2]+key[3]+1

자세한 내용은 코드와 함께 설명하겠습니다.

kvssd의 init 함수입니다. fd에는 kvssd.data를 meta에너느 kvssd.meta를 사용하도록 하였습니다.



QEMU에서 사용하는 kvssd\_state structure입니다.



여기서 key[4], kvssd\_value[1024], value\_pos, key\_pos는 현재 QEMU에서 사용하는 key와 block의 value값을 읽고 쓰는 역할을 수행합니다. pbn은 kvssd.data에서 읽을 data 위치를 나타내고, wait은 GET, EXIST를 수행하였을 때 state가 WAIT상태로 된 것을 따로 표기하기 위해 사용하였습니다. 아래 3가지 array는 metadata에 저장된 data를 관리하는 용도로 사용됩니다.

텍스트이(가) 표시된 사진

자동 생성된 설명

metadata의 data를 읽거나 수정해야할 때 위의 함수들을 수행하여 갱신해줍니다.

본격적으로 read와 write함수를 설명하겠습니다.

텍스트이(가) 표시된 사진

자동 생성된 설명

read함수에서는 command가 STATUS\_REG, KEY\_REG, VALUE\_REG일 경우에 수행합니다. STATUS\_REG일 경우에는 현재 kvssd의 status를 return하였습니다.

KEY\_REG를 read할 경우에는 metadata를 읽은 후에 입력 받은 key값과 비교하여 존재유무를 판단합니다. 만약 keypos array, 입력 받았던 key의 값을 저장하는 array에 동일한 key가 있으면 1을 return하고, 아니면 0을 return합니다.

VALUE\_REG을 read할 경우에는 입력 받은 key에 해당하는 value값을 차례로 return합니다.

다음으로 write함수에 대해 설명하겠습니다. 우선 command가 GET일 경우입니다.

텍스트이(가) 표시된 사진

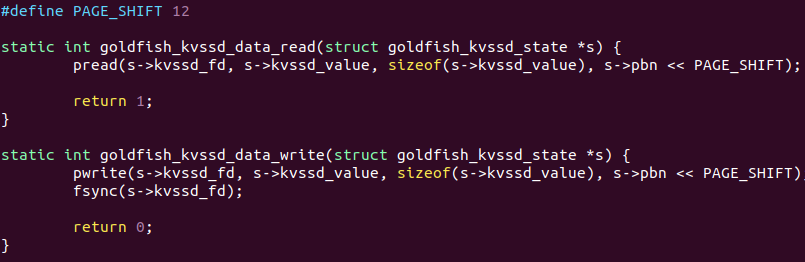
자동 생성된 설명

kvssd의 state를 BUSY로 바꾼 후 metadata를 읽고 입력 받은 key의 존재를 확인합니다.

만약 없으면(key\_pos array의 MAX\_BLOCK까지 비교) pbn에 0을 넣고 data read를 수행하여 data의 첫번째 값을 읽도록 수행하였습니다. (특별한 행동 지침이 없음)

key가 존재하면, pbn에 해당 key와 연결된 block 위치 kvssd\_key[i]값을 넣고 data read를 수행하도록 하였습니다.

아래는 data\_read, data\_write 함수입니다. 저는 s->pbn을 기준으로 data의 위치를 정해주었습니다.



다음으로 PUT Command의 경우입니다. 가장 먼저 기존에 ket를 입력 받은 적이 있을 경우입니다.

텍스트이(가) 표시된 사진

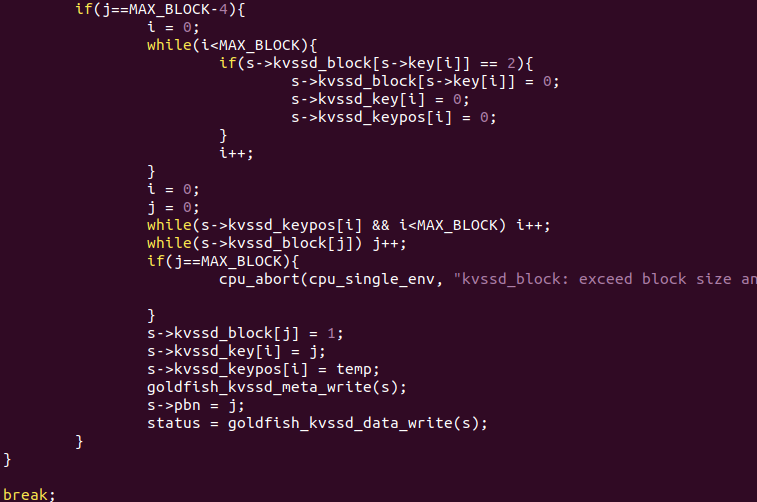
자동 생성된 설명

만약 입력 받은 key와 동일한 key가 이전에 있을 경우에는 해당 block을 2(invalid) 시키고, 이후로 빈 block을 찾은 후 새로 연결시켜줍니다. 그리고 metadata를 update해주고, 변경된 block에 data를 write해줍니다.

동일한 key가 없을 경우입니다. keypos array의 빈 곳을 찾아보고, 만약 block이 (MAX\_BLOCK-4)만큼 (GC 기준점) 사용되지 않았으면, keypos와 key, block을 새로 연결해주고, metadat를 업데이트한 후 data를 입력해줍니다.

텍스트이(가) 표시된 사진

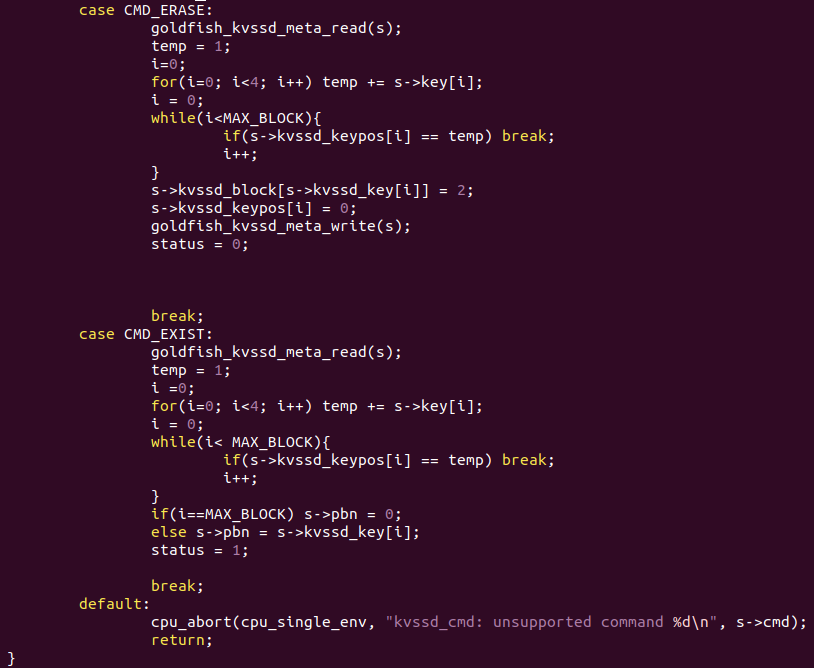
자동 생성된 설명



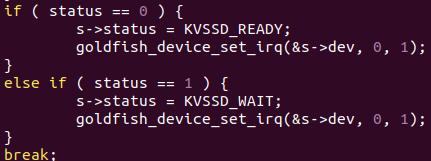
만약 GC 기준점인 (MAX\_BLOCK-4) 까지 빈 block이 없으면 전체 block을 다시 돌면서 invalid 된 block을 unused로 바꾸어주고, 연결된 key와 keypos 또한 0으로 초기화 해줍니다.

GC를 한 이후 unused된 block을 새로 할당하고, key와 keypos와 연결해주고, metadata를 update한 후 data를 입력해줍니다. 만약 GC를 한 이후에도 할당할 block이 없다는 것은 모든 block이 valid하다는 것이므로 cpu\_abort를 일으키게 해주었습니다.

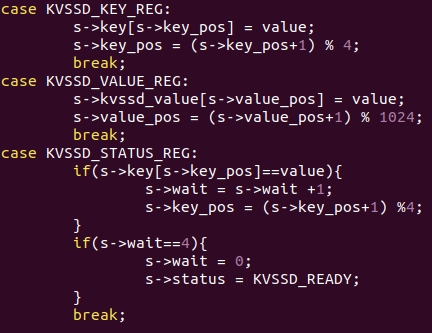
ERASE Command입니다. 입력 받은 key의 위치에 해당하는 block을 찾은 후 invalid 시켜줍니다. 저는 추가적으로 keypos 또한 0으로 초기화해주었습니다. ( PUT 할 때 구분하기 위해)

EXIST Command입니다. 입력 받은 key가 존재할 경우에는 pbn에 현재 block의 위치를 저장하였고 없으면 0을 저장하였습니다. status는 나중에 WAIT로 바꾸기 위해 1을 넣어주었습니다.

위에 경우에서 status 상황에 따라 s->status의 값을 정해주었습니다. PUT Command와 ERASE Command일 때는 status가 0으로 되어 READY 상태가 되고, GET Command와 EXIST Command일 때는 status가 1으로 되어 WAIT 상태가 됩니다.



write 함수에서 Register가 Command가 아닐 경우입니다.



KEY\_REG일 경우에는 s->key와 s-> key\_pos를 통해 user로부터 입력받은 key를 s->key로 옮겨주었습니다.

VALUE\_REG일 경우에는 s->kvssd\_value와 value\_pos를 통해 user로부터 입력받은 data를 s->kvssd\_value로 옮겨주었습니다.

STATUS\_REG일 경우에는 현재 key가 일치하는지 key[0], key[1], key[2], key[3] 4회 확인하면서 wait값을 변화시켜 wait가 4가 되었을 때 READY STATE로 변하게 해주었습니다.

이상으로 QEMU code 설명은 마치겠습니다.

텍스트이(가) 표시된 사진

자동 생성된 설명다음은 제가 test하기 위해 사용한 user code입니다.

텍스트이(가) 표시된 사진

자동 생성된 설명

실행 결과입니다. 현재 상황은 temp는 GET하려는 key 값이고, ERASE한 key 위치를 읽으려고 하기 때문에 pbn이 0으로 출력 되었습니다. 오른쪽 화면은 GET이 실패하였음으로, 마지막으로 env.value[0]에 저장되었던 1023이 출력되는 것입니다.

텍스트, 컴퓨터, 모니터, 스크린샷이(가) 표시된 사진

자동 생성된 설명