과제 #4 : Snapshot(Checkpointing) in xv6

○ 과제 목표
- 파일 시스템의 스냅샷(snapshopt, checkpoint)은 특정 시점의 파일시스템 상태를 보존하여 이후 해당 시점으로 복원하거나 참조할 수 있게 해주는 기능 (프로세스의 체크포인팅 기법도 별도로 있음)
- xv6의 파일 시스템은 파일 시스템의 스냅샷(snapshot, checkpoint, 복사본)이나 incremental 백업 기능을 지원하지 않음
- 스냅샷은 특정 시점의 파일시스템을 읽기 전용으로 저장하여 이후 언제든 그 시점의 상태로 복구할 수 있게 하는 기능
- 본 설계 과제에서는 Copy-On-Write(COW) 기법을 활용하여 xv6 파일 시스템에 스냅샷 기능을 추가하고, 이를 통해 저장장치 데이터의 백업과 복구 기법, 그리고 COW 기반 멀티 버전 파일시스템을 설계하는 능력을 배양
- COW를 사용하면 새로운 데이터 작성 시 기존 블록을 보존하고 새 블록에 기록함으로써, 스냅샷 생성 시 전체 데이터를 복사하지 않고도 공간 효율적인 스냅샷을 구현할 수 있음. Oracle의 ZFS 등의 실제 파일시스템도 이러한 COW를 통해 신속하고 공간 효율적인 스냅샷을 제공하며, 스냅샷이 없는 경우와 비교해도 스냅샷 파일시스템의 성능의 크게 저하되지 않는다고 알려져 있음

○ 과제 내용
- A. block COW 구현
  - ✓ 기존 xv6 구조
    - ☞ xv6는 inode를 통해 파일의 메타데이터와 데이터 블록의 위치를 관리하지만, 이 설계는 하나의 블록이 여러 파일에 의해 공유되는 상황을 고려하지 않음
    - ☞ 파일 시스템의 특정 시점을 보존하려면 디스크 블록 전체를 복사해야 함
    - ☞ 이러한 방식은 스냅샷 생성 시 모든 블록을 복사해야 하므로 일부만 수정이 있는 상황에서 동일한 내용의 블록이 여러 개 생길 수 있다는 단점이 있음
    - ☞ 이를 해결하기 위해, 이번 과제에서는 스냅샷 생성 시 실제 블록 복사 대신 기존 블록을 참조하는 방식을 도입
    - ☞ 이 때, 여러 스냅샷이 같은 블록을 공유하는 상태에서 해당 블록의 수정을 진행한다면 새로운 문제가 발생
    - ☞ 하나의 블록을 수정할 때 참조하고 있는 다른 파일들의 내용 또한 변경되기 때문에 스냅샷이 제대로 복구되지 않음
    - ☞ 따라서 이 문제를 방지하기 위해 블록에 대하여 COW 방식을 적용하여 문제를 해결
  - ✓ 구현 내용
    - ☞ 어떤 블록이 얼마나 많은 파일에 의해 참조되어 있는지 확인하고, 이를 토대로 bcow 및 bfree의 작동을 결정하는 토대가 되는 메타데이터 파일 작성 (파일 형식 자유)
    - ☞ 메타 데이터 파일의 위치는 /snapshot 디렉토리 아래에 작성
    - ☞ snapshot 파일은 수정 불가능, 현재 파일 시스템은 수정 가능.
    - ☞ snapshot 파일이 참조하고 있는 모든 블록은 불변 블록으로 현재 파일 시스템에서 해당 블록을 참조하고 있다면 수정 요청 발생 시 해당 블록에 대해 COW를 적용해야 함
    - ☞ 블록에 대한 참조가 모두 없어질 때 해당 블록을 할당 해제
- B. 스냅샷 구현
  - ✓ int snapshot_create(void)
    - ☞ 현재 파일시스템 상태의 스냅샷을 생성하고 새로운 스냅샷 식별자(ID)를 할당하여 리턴
    - ☞ 이 시스템 콜의 호출이 성공하면 ID에 대응하는 스냅샷이 읽기 전용으로 생성.
    - ☞ 또한 /snapshot/[ID] 디렉토리를 root 디렉토리로 하여 현재 파일 시스템을 그대로 캡처.
    - ☞ 즉, root 디렉토리 안의 모든 파일과 하위 폴더를 재귀적으로 /snapshot/[ID]에 복사(블록 변경 없음).
    - ☞ 이번 과제에서는 내용 복구에 초점을 맞춘 구현이기 때문에 디렉토리의 데이터 블록은 bcow를 적용하지 않아도 됨
    - ☞ 모든 스냅샷 관련 작업에서 /snapshot 디렉토리와 T_DEV 파일은 캡처하지 않음.
    - ☞ 스냅샷 생성은 현재 디스크 상의 모든 파일과 디렉토리 상태를 기록하지만 COW를 사용하므로 데이터 블록은 복사하지 않음.
    - ☞ 스냅샷 생성 실패시 0보다 작은 값 리턴
  - ✓ int snapshot_rollback(int snap_id)
    - ☞ 현재 파일시스템을 id의 스냅샷으로 복구함
    - ☞ 복구시 inode 번호를 구현할 필요 없이 새로운 inode를 할당받아 복구
    - ☞ 올바르지 않은 id 입력시 0보다 작은 값 리턴
  - ✓ int snapshot_delete(int snap_id)
    - ☞ 지정한 ID의 스냅샷을 삭제.
    - ☞ 올바르지 않은 id 입력시 0보다 작은 값 리턴

```
// user.h

int snapshot_create(void);
int snapshot_rollback(int);
int snapshot_delete(int);
```

- C. 테스트용 유저 프로세스 및 시스템콜 구현
  - ✓ 파일 addr 출력 테스트
    - ☞ mk_test_file : 테스트용 파일을 생성하는 유저 프로세스 (코드 제공, 사용법은 코드 참고)

```
#include "types.h"
#include "fcntl.h"
#include "user.h"
int main(int argc, char *argv[]) {
    int fd;
    if (argc < 2) {
        printf(1, "need argv[1]\n");
        exit();
    }
    if ((fd = open(argv[1], O_CREATE | O_WRONLY)) < 0) {
        printf(1, "open error for %s\n", argv[0]);
        exit();
    }
    char buf[513];
    for (int i = 1; i < 511; i++) buf[i] = 0;
    buf[511] = '\n';
    for (int i = 0; i < 12; i++) {
        buf[0] = i % 10 + '0';
        write(fd, buf, 512);
    }
    char *str = "hello\n";
    write(fd, str, 6);
    close(fd);
    exit();
}
```

☞ append : 파일 수정을 위한 유저 프로세스 (코드 제공)

```
#include "types.h"
#include "stat.h"
#include "user.h"
#include "fcntl.h"
int
main(int argc, char *argv[])
{  if(argc != 3){
    printf(2, "Usage: append filename string\n");
    exit();
  }
  // open the file for read+write, create if needed
  int fd = open(argv[1], O_RDWR | O_CREATE);
  if(fd < 0){
    printf(2, "append: cannot open %s\n", argv[1]);
    exit();
  }
  // move offset to the end by reading until EOF
  char buf[1];
  while(read(fd, buf, 1) == 1); // drain to EOF
  // now at end, write the string
  if(write(fd, argv[2], strlen(argv[2])) < 0){
    printf(2, "append: write failed\n");
    close(fd);
    exit();
  }
  close(fd);
  exit();
}
```

☞ print_addr : 파일명을 인자로 입력받아 해당 파일이 참조하는 블록의 주소를 출력하는 유저 프로세스

```
$ print_addr README
addr[0] : 3c
addr[1] : 3d
addr[2] : 3e
addr[3] : 3f
addr[4] : 40

$ mk_test_file hi
$ print_addr hi
addr[0] : 33d
```

```
addr[1] : 33e
addr[2] : 33f
addr[3] : 340
addr[4] : 341
addr[5] : 342
addr[6] : 343
addr[7] : 344
addr[8] : 345
addr[9] : 346
addr[10] : 347
addr[11] : 348
addr[12] : 349 (INDIRECT POINTER)
addr[12] -> [0] (bn : 12) : 34a
```

✓ 스냅샷 생성, 백업, 삭제 테스트

☞ snap_create : 시스템콜 snapshot_create()를 호출하여 스냅샷을 만드는 프로그램 (id는 알아서)

```
$ snap_test
$ ls /snapshot/01/
.              1 27 384
..             1 25 64
README         2 28 2286
cat            2 29 15684
echo           2 30 14560
forktest       2 31 8996
grep           2 32 18520
init           2 33 15456
kill           2 34 14648
ln             2 35 14544
ls             2 36 17116
mkdir          2 37 14668
rm             2 38 14648
sh             2 39 28700
stressfs       2 40 15580
usertests      2 41 63072
wc             2 42 16096
zombie         2 43 14220
snap_create      2 44 14112
print_addr     2 45 14360
append         2 46 15132
mk_test_file   2 47 15332
snap_rollback  2 49 14800
```

☞ snap_rollback : 시스템콜 snapshot_rollback()을 호출하여 인자로 받은 ID의 스냅샷으로 현재 파일 시스템을 복구하는 프로그램

☞ snap_delete : 시스템콜 snapshot_delete()을 호출하여 인자로 받은 ID의 스냅샷 파일을 삭제하는 프로그램

○ 과제 제출 마감

- 2025년 11월 26일(수) 23시 59분까지 구글클래스룸으로 제출
- 보고서 (hwp, doc, docx 등으로 작성 - 프로그램이 수행된 결과 캡쳐 등 포함)
- 가산점 2 부여 마감 기한 : 25년 11월 19일(수) 23시 59분까지 구글클래스로 제출 1초라도 늦으면 가산점2 없음
- xv6에서 변경한 소스코드 및 테스트 쉘 프로그램 소스코드(본 과제의 경우 테스트를 위해 제공한 코드 모두 포함)

○ 필수 구현

- A, B, C

○ 배점 기준

- 100점 (부분점수 없음)