**Pintos Project 4**

**(설계 프로젝트 수행 결과)**

과목 명 : [CSE4070]운영체제

담당 교수 : 소정민 교수님

조 / 조원 : 41조 20130859 송민규, 20130956 장동욱

개발 기간 : 2018.12.23~12.26

**프로젝트 제목 : pintos project 4**

**제출일 : 2018.12.26**

**참여 조원 : 20130859 송민규, 20130956 장동욱**

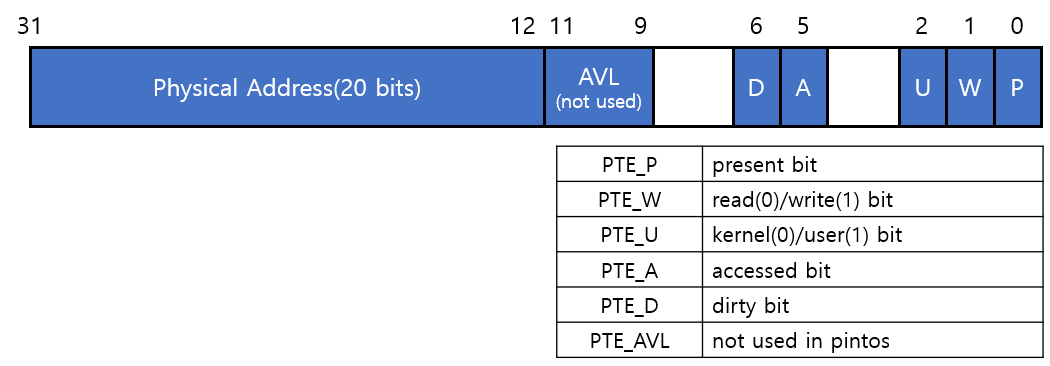
1. **개발 목표**

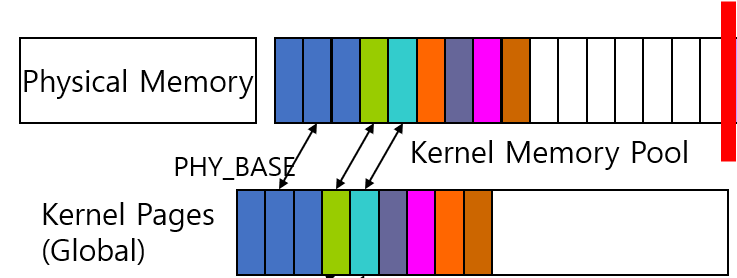
* **page table implementation & page table management**
* **page fault handler**
* **page replacement(frame eviction) using LRU approximation(clock)**
* **stack growth**

**이때까지 구현했던 프로젝트 내용에 더해 위 사항들을 구현한다. 단, 프로젝트 3의 구현 내용이 test case 수행 결과에 side effect를 미칠 수 있기 때문에, 프로젝트 2의 구현에 이어서 프로젝트를 수행한다.**

1. **개발 범위 및 내용**
   1. **개발 범위**

* **frame  
  physical memory의 일부분으로, page size에 맞게 사용되어야 한다. page에 할당되는 용도로 사용된다.**
* **page  
  virtual memory의 일부분으로 그 길이는 기본값으로 4,096 bytes이다. user가 직접 할당받아서 접근 및 사용할 수 있는 메모리다.**
* **swap slot  
  frame의 eviction을 위한 공간으로, page size에 맞게 사용되어야 한다.**
* **frame table  
  entry로서 frame을 점유하는 page에 대한 pointer를 포함한다. frame table의 가장 중요한 기능은 empty frame을 가져오는 것인데, 만약 그러한 frame이 없다면 eviction을 사용해야 한다. evict를 하고자 하는데 empty swap slot이 없다면 kernel을 panic시켜야 한다. supplementary table의 accessed와 dirty bit을 사용하여 eviction될 frame을 선택한다. 본 프로젝트에서는 LRU 알고리즘의 approximation 기법인 second chance(clock) 알고리즘을 사용한다.**
* **second chance algorithm  
  free frame이 있다면 할당한다. clock hand는 다음 free frame을 가리키고 있으며, 해당 frame을 다음 page fault 시 할당한다. free frame이 없다면 evict한다. clock hand가 돌아가면서 accessed bit이 0인 frame을 evict하고, 해당 frame을 할당한다. 만약 accessed bit이 0인 frame이 없다면 clock hand를 계속 돌리면서 1인 accessed bit를 0으로 수정하고, 가장 먼저 나오는 accessed bit이 0인 frame을 evict하고 할당한다.**
* **page table  
  virtual address를 physical address로 변환하는데 참조한다. virtual address에서 page number에 해당하는 값을 index로 삼아 caller인 thread의 page table의 base로부터 참조할 entry를 찾고, 해당 entry의 내용이 변환 후의 frame number가 된다. frame number와 변환되지 않은 부분인 offset의 조합으로 physical memory를 참조한다.**
* **supplementary page table  
  page table에 기본적으로 포함되지 않는 정보를 담아서 기존 보다 더 sophisticated한 page fault handling을 가능케 한다. page fault 발생시 해당 memory reference가 valid한지의 여부와 해당 page에 들어갈 data의 위치를 포함하고 있다.**



* **swap table  
  사용 중인 swap slot과 비어있는 swap slot을 관리한다.**
* **page fault  
  이전 프로젝트까지는 page fault는 kernel 혹은 user space 상에서 발생한 bug를 나타냈었다. user space에서 kernel space에 접근하고자 하면 page fault handler인 exception.c의 page\_fault 함수가 interrupt에 의해 수집된 정보를 바탕으로 page fault를 발생시켰다. 하지만 이번 프로젝트에서 page fault는 file 또는 swap으로부터 page를 가져와야함을 의미한다.**
* **page table management  
  수업 시간에 배운 추상적 개념과 달리 컴퓨터 아키텍처에 의존하는 구현상의 차이점이 있기에, 이를 유념하여 구현한다:  
  1. 80x86 아키텍처에서는 physical memory에 직접 접근할 수 없다.  
  이 때문에 kernel virtual memory에 physical memory를 직접 매핑하여 kernel virtual memory를 통하여 간접적으로 physical memory에 접근한다. 이에 대한 묘사는 강의자료에 첨부된 diagram에서도 확인할 수 있다.** **2. page table entry에 존재하는 내용만으로는 불충분하다.  
  수업시간에는 별도의 자료구조를 부여하여 설명하지 않았지만 page table management를 위해 여러가지 정보가 필요하고 이를 별도의 supplementary page table 자료구조를 구현하여 저장한다.  
  3. swap disk라는 swap을 위한 별도의 공간을 구현한다.  
  역시 수업시간에 직접적으로는 다루지 않았으나 page replacement를 위해 별도 구현한다. pintos-mkdisk swap.dsk --swap-size=n 명령어를 사용해서 디스크를 생성한다.**
* **Stack Growth  
  stack access인 경우 stack grwth가 허용되어야 하기 때문에 stack access를 판별할 수 있어야 한다. 일반적인 운영체제의 경우, stack pointer 위에 data를 push하기 때문에 stack pointer 아래에 존재하는 시도는 올바른 stack access라고 볼 수 없다. 하지만 80x86 아키텍처의 PUSH instruction과 PUSHA instruction은 각각 4 byte와 32 byte를 push하는데, 해당 아키텍처에서는 stack pointer의 adjustment보다 접근 권한을 먼저 체크하기 때문에 해당 push들이 허용되어야 한다. 크기 32인 전역변수를 선언하여 해당 상황에 대한 예외처리를 한다.**
  1. **개발 내용**
* **frame.c / frame.h  
  - clock algorithm 구현  
  clock 함수가 victim pointer인 clock hand를 frame table을 따라 돌려가면서 victim frame을 찾아낸다. dirty bit과 accessed bit을 사용하여 점차 조건을 강화해 나간다. 이 때, clock hand를 돌리는 함수, 즉, frame table을 탐색하는 역할을 clock이 수행하되, victim frame을 찾을 때까지 매 탐색 마다 victim frame을 선택하는 조건이 달라진다. 여러 조건들을 별도의 함수[[1]](#footnote-1)로 구현하고, 함수 포인터를 clock 함수의 인자를 넘겨주는 방식으로 탐색문을 여러 번 작성하지 않고도 여러 번의 탐색에 각각 다른 조건문을 적용시킬 수 있다.**
* **page.c / page.h  
  stack growth 함수를 통해 stack growth를 구현한다. 인자 인 upage에 상응하는 supplementary page table entry와 frame table entry를 생성, 초기화 후 각각 상응하는 table에 삽입한다. install page를 통해 stack을 grow시키고, 실패할 경우 관련 자료구조를 free 및 release한다.**
* **swap.c / swap.h  
  swap을 위한 초기화 swap\_init 함수와 실제 swapping을 위한 swap\_in, swap\_out 함수로 구성된다. page 사이즈와 block sector 사이즈를 가리키는 전역변수 간의 연산을 통하여 block을 읽고 block에 쓸 위치를 계산한다. for문을 통해 페이지 사이즈에 상응하는 block sector만큼 읽거나 쓰도록 하고, 인자 index를 이용하여 매번 읽고 쓸 위치를 계산한다.**

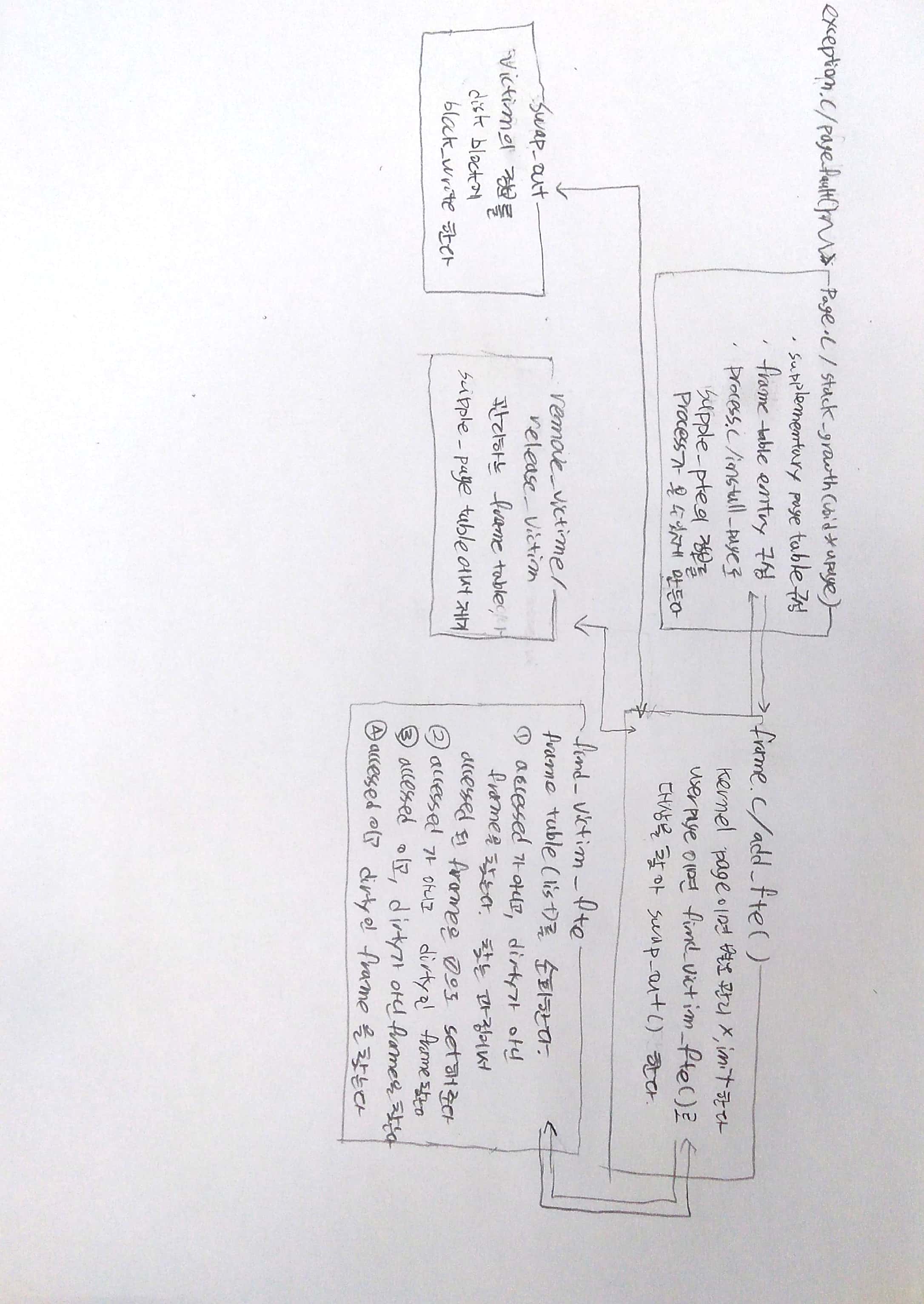
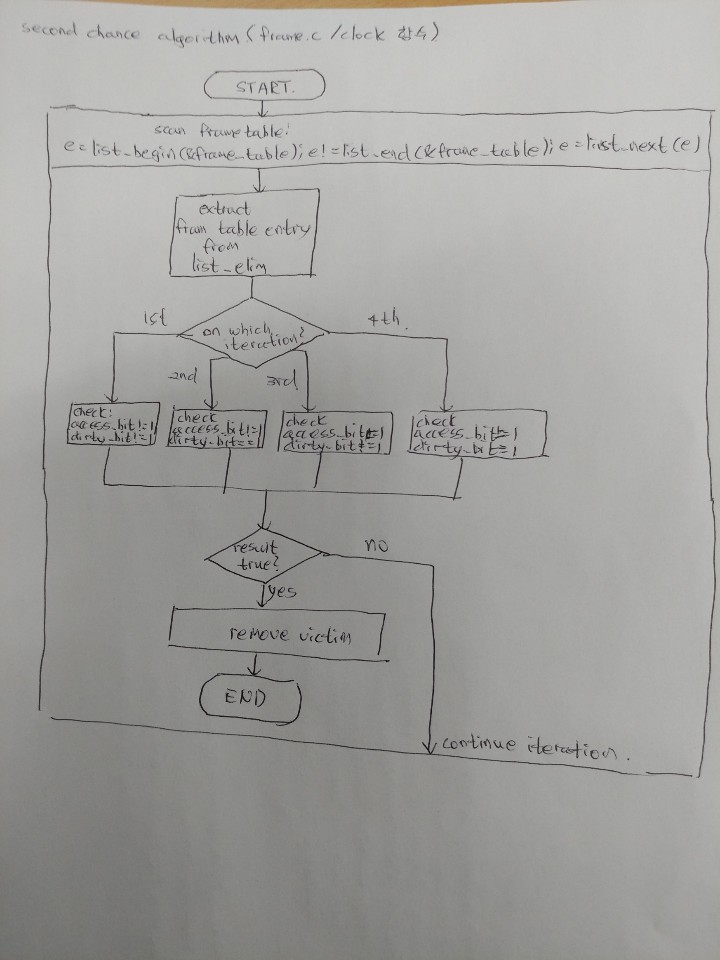
1. **추진 일정 및 개발 방법**
   1. **추진 일정**

* **11.22–11.23 명세서 탐독 및 자료조사**
* **11.23-11.25 코드 작성**
* **11.25-11.26 디버깅 및 보고서 작성**
* **12.26 사이버캠퍼스 제출**
  1. **개발 방법**
* **코드 구현과 수정은 vim으로 수행했다. 디버깅은 hex\_dump와 출력문을 활용했다. 테스트와 디버깅은 make test, make grade, backtrace 명령문과 output파일, result파일을 활용했다.**
  1. **연구원 역할 분담**
* **각기 다른 방식의 구현을 시도하다가 test case를 가장 많이 통과하는 것으로 통일했다.**

1. **연구 결과**
   1. **합성 내용**

**프로젝트 구현사항을 구조도로 나타내면 다음과 같다.**

* 1. **제작 내용**
* **userprog 디렉토리의 process.c, process.h, exception.c, exception.h, threads 디렉토리의 thread.c, thread.h, init.c, init.h를 수정하고 프로젝트 ppt와 pintos 매뉴얼에 나와있는 구현 방식을 따라 vm폴더의 frame.c, frame.h, page.c, page.h, swap.c, swap.h를 추가하여 개발 내용의 내용을 구현하였다.**
  1. **시험 및 평가 내용**

**명세서에 주어진 일부 요구사항을 만족하고, make check 및 make grade 결과 일부 테스트 케이스를 통과했다.  
  
**

1. **기타**
   1. **연구 조원 기여도**

* **송민규(50%), 장동욱(50%)**
  1. **소감**
* **프로그램의 여러 부분이 복잡하게 상호작용을 하여, 디버깅이 어렵고 수정 사항의 연쇄작용을 파악하는 것이 어려웠다. 테스트 케이스가 많아 테스트 시간이 오래 걸려 전체 프로젝트 소요 시간이 길어졌다.**
* **마지막 프로젝트인 만큼 어렵고, 기말고사 기간과 겹쳐서 시간이 많이 부족했지만 많은 것을 배울 수 있는 프로젝트였다.**

1. no\_access\_no\_dirt(): access도 없고 modify도 된 적이 없는 경우, no\_access\_dirt(): access는 없었으나 modify된 적은 있는 경우, access\_no\_dirt(): access는 있었으나 modify된 적이 없는 경우, access\_dirt(): access도 된 적이 있고, modify된 적도 있는 경우 true를 return [↑](#footnote-ref-1)