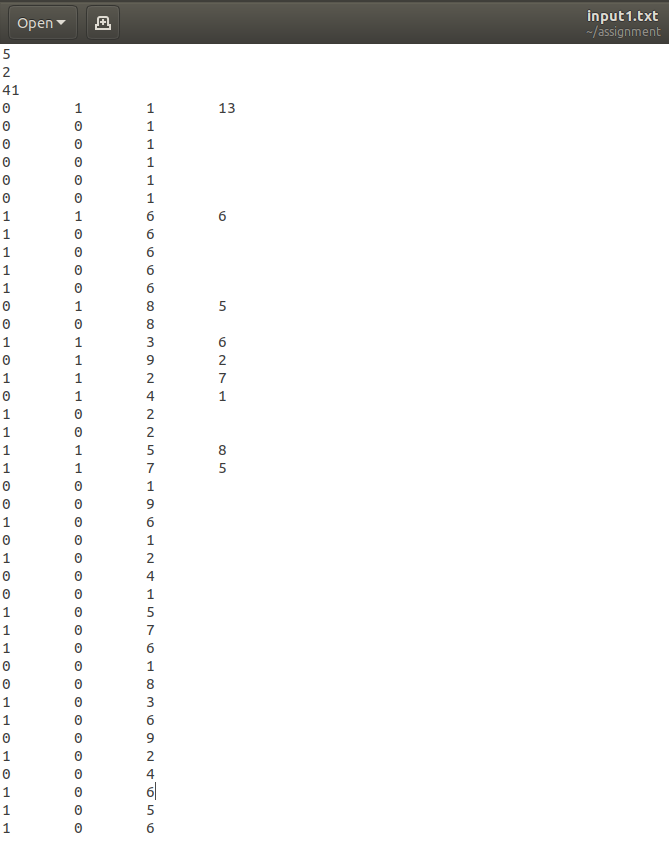
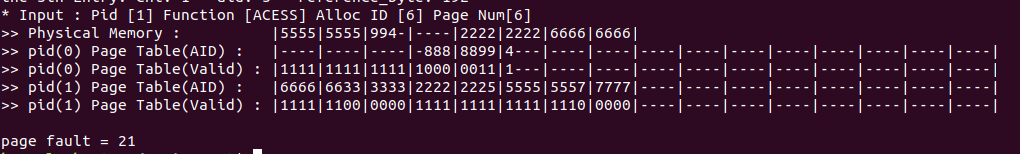
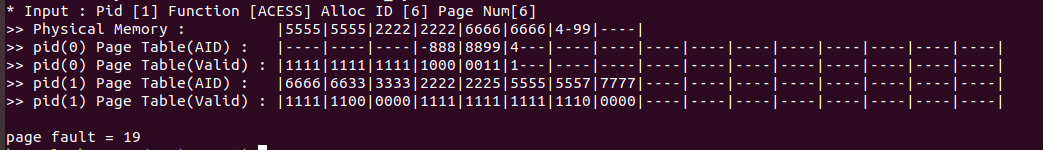
1. Input1



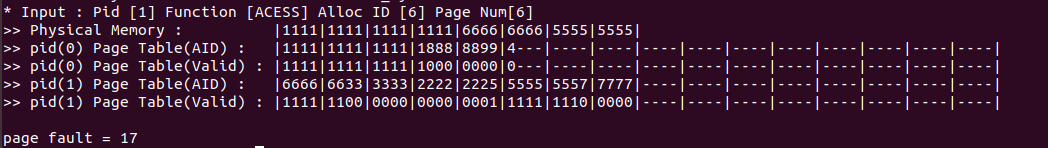
FIFO algorithm의 결과



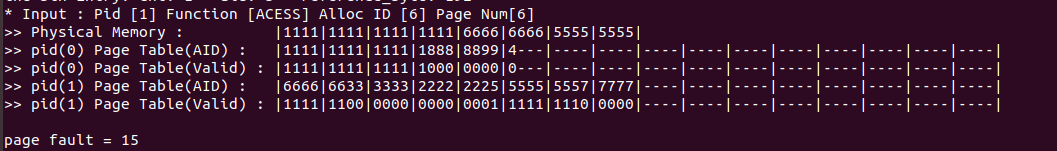
LRU algorithm의 결과



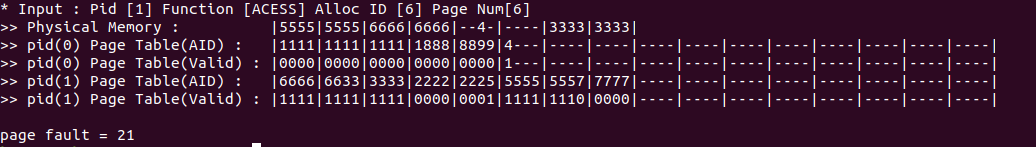
Sampled LRU의 결과



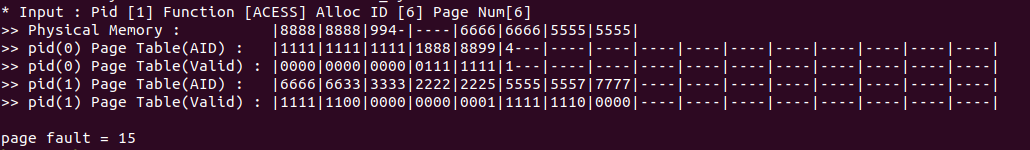
LFU algorithm의 결과



MFU algorithm의 결과

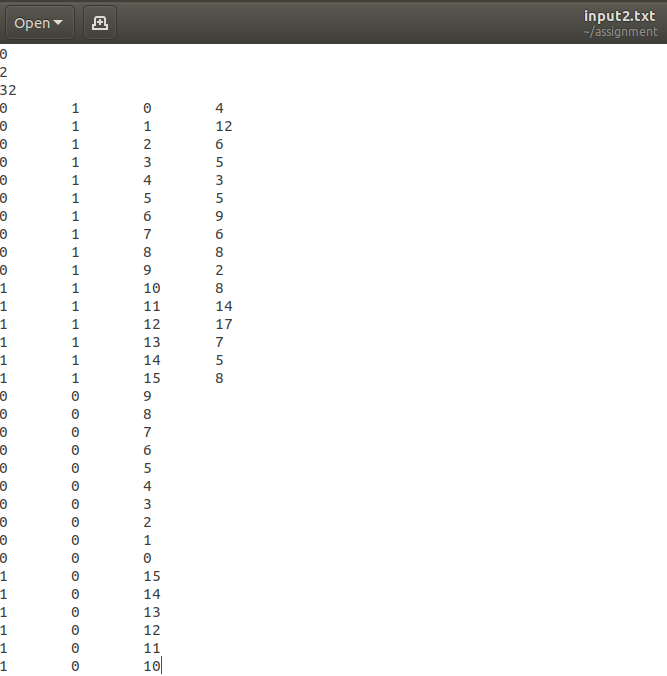


Optimal algorithm의 결과

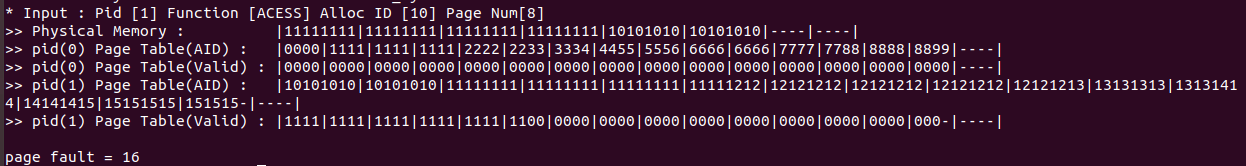


각 algorithm의 page fault rate는 아래와 같다.

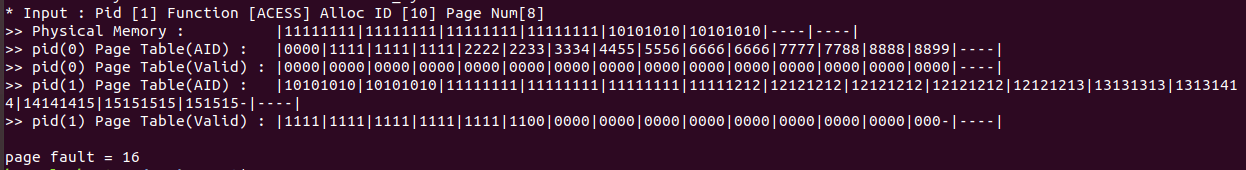
여기서 LFU algorithm이 가장 낮은 page fault rate를 가진다. 그 이유는 input1의 경우 빈번히 사용되는 page가 프로그램이 진행되면서 계속해서 access 되는 경우를 input으로 넣은 것인데, 이는 특정 함수의 빈번한 호출을 요구하는 프로그램이 예가 될 수 있다. 이 경우에는 count값이 높은 page를 physical memory에 오랫동안 유지해주는 LFU algorithm이 page fault rate가 낮게 나온다.

2. input2  


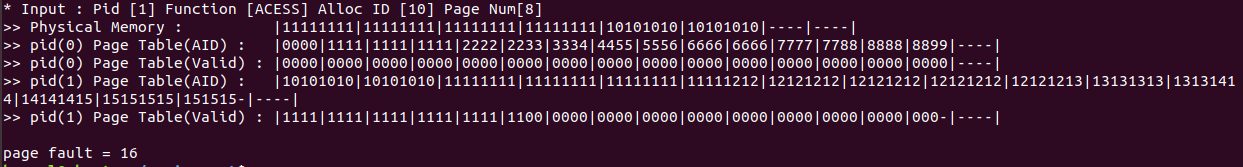
FIFO algorithm의 결과



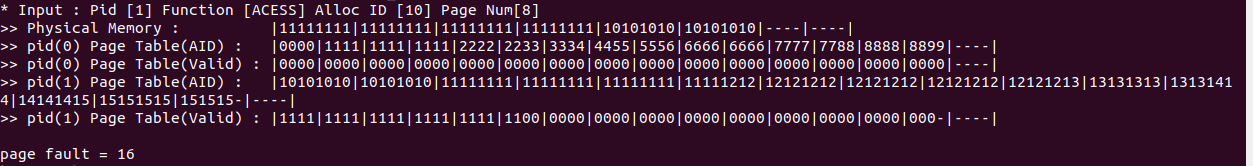
LRU algorithm의 결과



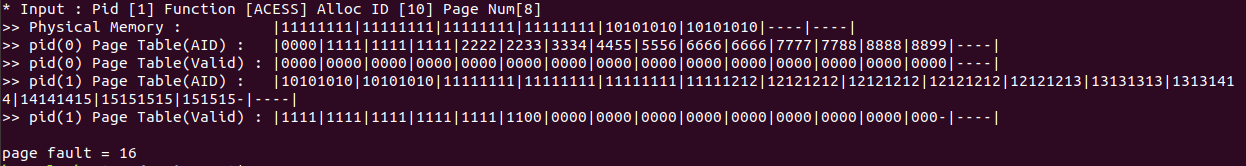
Sampled LRU의 결과



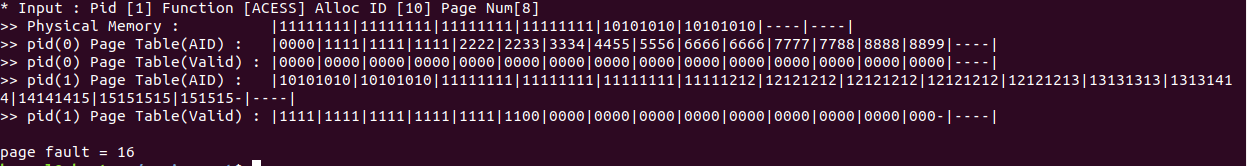
LFU algorithm의 결과



MFU algorithm의 결과

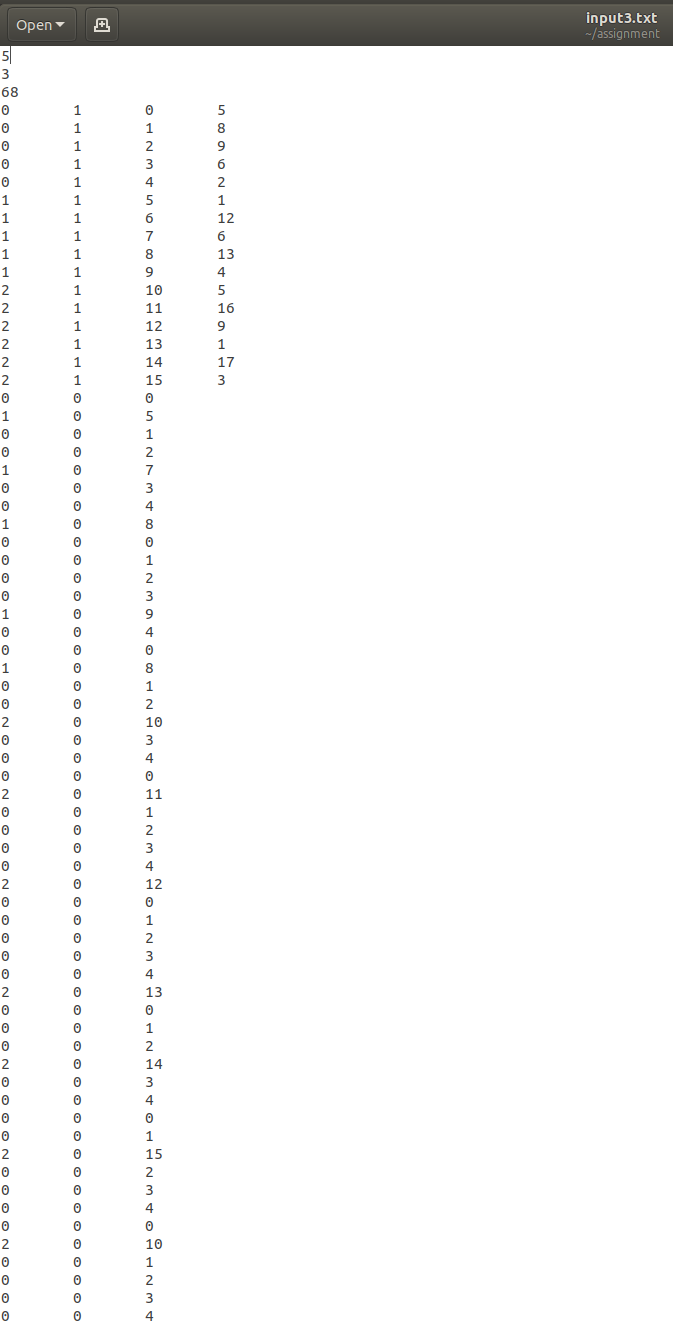


Optimal algorithm의 결과

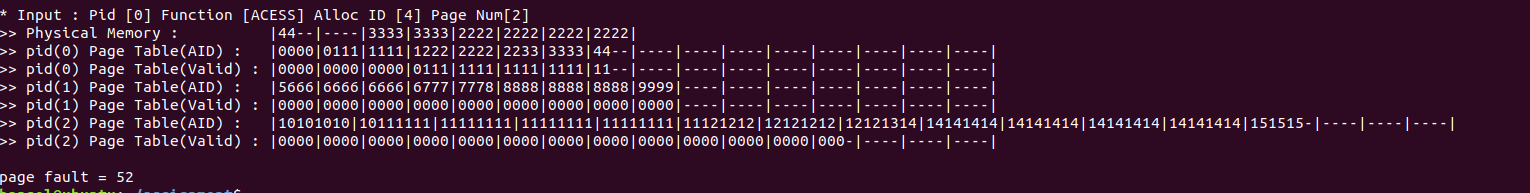


각 algorithm의 page fault rate는 아래와 같다.

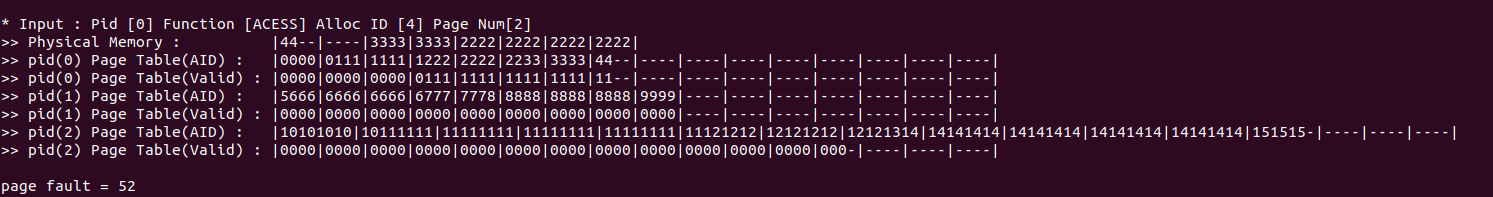
Input2의 경우에는 page fault rate가 모든 algorithm에 대해서 100%로 동일하다. 그 이유는 input2는 프로그램이 수행되면서 다른 함수의 호출이 적고 순환문의 크기가 매우 큰 program과 비슷한 page access를 input으로 넣은 것이다. 즉 프로그램에서 pc값은 대략적으로 계속 커지는 경우이다. 이 경우는 계속해서 새로운 page에 대한 access를 요청하게 된다. 그러므로 모든 access 요청이 page fault를 유발하는 것이다. 이 결과에서 메모리 관리에는 단순히 page replacement algorithm만이 아닌 앞으로 사용될 것 같은 page를 예측하고 미리 physical memory에 mapping 해놓는 기술 또한 필요함을 알 수 있다.

3. input3  


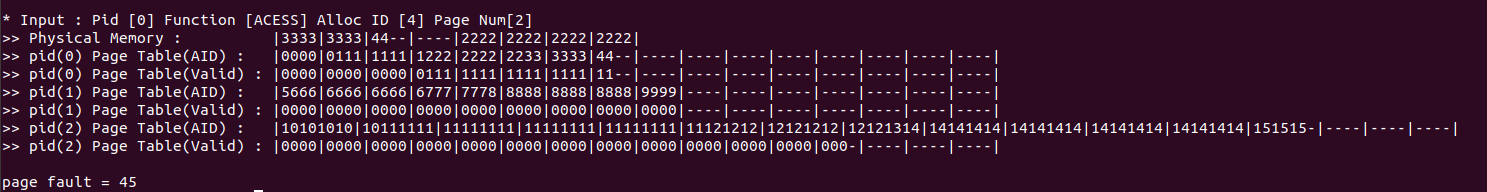
FIFO algorithm의 결과



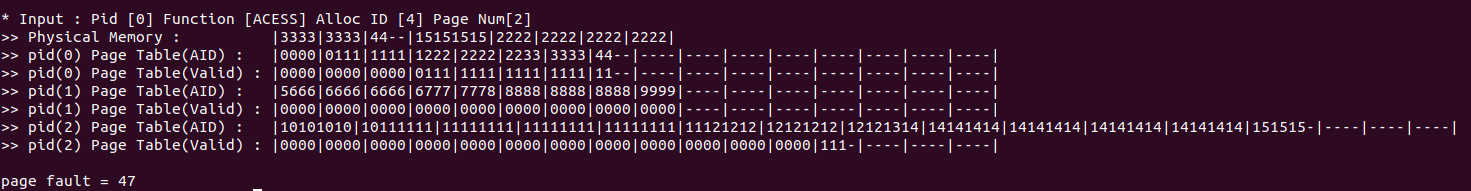
LRU algorithm의 결과



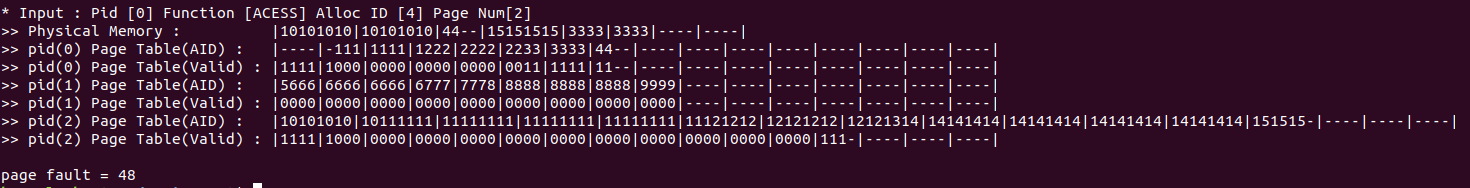
Sampled LRU의 결과



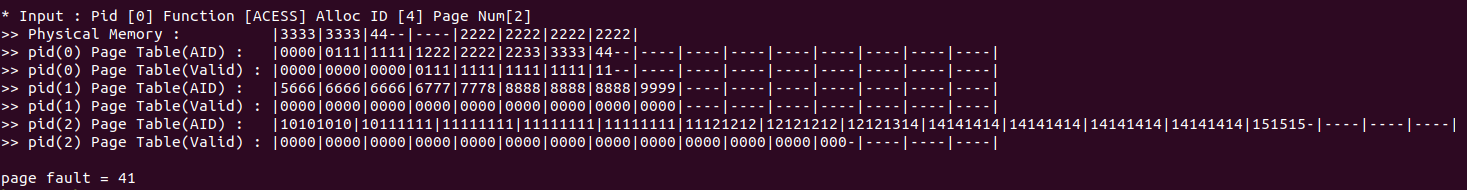
LFU algorithm의 결과



MFU algorithm의 결과



Optimal algorithm의 결과



각 algorithm의 page fault rate는 아래와 같다.

Input3에서는 optimal algorithm을 제외하면 sampled LRU가 가장 낮은 page fault rate를 보인다. 그 이유는 input3는 하나의 큰 프로그램이 계속해서 반복작업을 수행하는 동안 다른 프로그램이 한번씩 처리되는 경우를 나타낸 것인데, 이 때 sampled LRU에서는 반복해서 작업되고 있는 페이지의 reference byte값이 높아서 physical memory에 오랫동안 유지하게 된다. (이 과제에서 LRU가 Sampled LRU보다 낮은 효율을 보이는 이유는 physical memory가 상대적으로 작기 때문이다.)