Test TLB

TLB miss의 비용을 측정하고 현상을 설명하는 숙제이다.

```
int jump = PAGESIZE / sizeof(int);
for (i = 0; i < NUMPAGES * jump; i += jump)
    a[i] += 1;
```

위 코드 조각처럼 페이지 당 한번 씩만 접근하면서 실행하면 매번 주소 변환이 일어나서 TLB miss의 비용이 선명하게 나타난다. 이를가상 주소 공간의 크기를 늘리면서 메모리 접근 시간을 측정하면 아래 그림 처럼 16개 페이지 근처에서 20ns로 급격히 증가하고 1024개 근처에서 70ns 이상으로 급격히 증가하는 것을 관찰할 수 있다. 이를 통해 2 level TLB가 존재하는 것을 짐작할 수 있다.(자세한 내용은 교재 Homework(Measurement) 참고: : http://pages.cs.wisc.edu/~remzi/OSTEP/vm-tlbs.pdf)

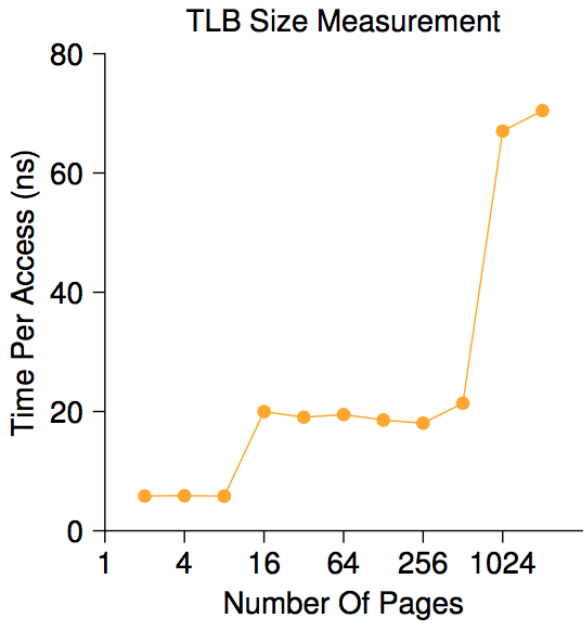


Figure 19.5: **Discovering TLB Sizes and Miss Costs**

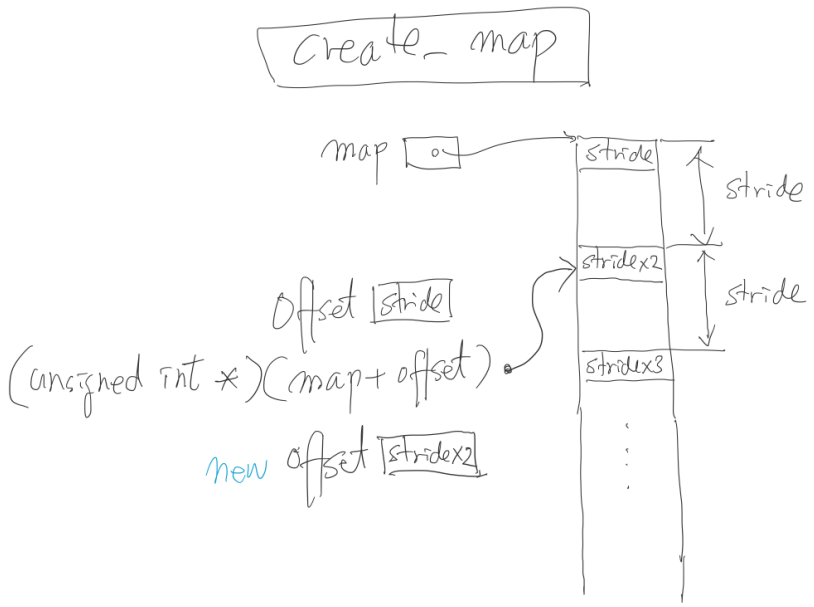
이런 측정을 Linus Torvalds가 작성한 아래 코드를 이용하여 실행해본다.
https://github.com/torvalds/test-tlb/blob/master/test-tlb.c

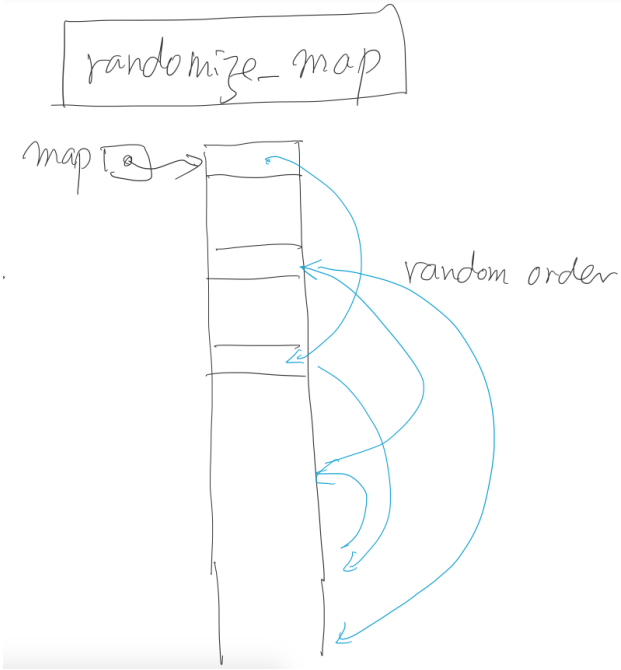
실행 환경은 리눅스로 제한되며 컴파일하려면 'make', 실행하려면 'make run'을 하면 된다.

위 교재의 코드 조각에 해당되는 부분은 아래와 같은데 mmap() 함수로 큰 메모리를 할당 받고 그 안에 메모리 캐시 크기보다 큰 보폭(stride)으로 다음에 접근할 위치(offset) 값을 미리 써 놓고 이를 링크처럼 따라가면서 읽는 프로그램이다. 따라서 메모리 캐싱에 의한 평균 메모리 접근 시간 단축 효과는 기대할 수 없고 주소 변환 과정에서 TLB miss가 발생했을 때 얼마마 느려지는지가 더 잘 드러난다.

```
95          gettimeofday(&start, NULL);
96          do {
97                  count++;
98                  offset = *(unsigned int *)(map + offset);
99          } while (!stop);
100         gettimeofday(&end, NULL);
101         usec = usec_diff(&start, &end);
```

Map에는 아래 그림 처럼 다음으로 접근할 위치(offset) 정보가 기록되어 있고 이를 randomize 해서 이용한다.





이를 실행할 때 결과를 아래 처럼 컴마(,)로 구별해서 CSV 화일을 생성하면 엑셀이나 구글시트에서 읽을 때 컬럼을 구별할 수 있어서 그래프 등을 그릴 때 편할 것이다.

```
$ for i in 4k 8k 16k 32k 64k 256k 512k 1M 2M 4M 6M 8M 16M 64M 128M 256M ; do echo -n "$i, "; ./test-tlb -r $i 64 ; done
```

```
// In test-tlb.c: line 283,284
printf("%6.2f , ns (~%.1f cycles)\n",
                cycles, cycles*FREQ);
```

./test-tlb 의 옵션은 -r 만 사용하기로 한다.

[보고서 내용]

1. 자신의 실행 환경
    a. 알아낸 만큼 설명
2. 여러 경우로 실행
    a. map size 를 여러가지로 변경
    b. stride를 여러가지로 변경 (메모리 캐싱 효과가 나타나게도 변경해 본다)
3. 결과를 그래프로 그림 (교재의 그래프 처럼)
4. 분석 내용을 설명
    a. 실행 환경의 TLB 구조

b. 실행 환경의 캐시라인의 크기