H/W 4

Homework in Chapt 19 (TLB Measurement)

본 과제는 Ubuntu 20.04 운영체제에서 진행되었습니다.

Question 1.

본문의 내용으로 짐작해보아 nanosecond 단위의 시간 측정이 필요하다는 고 생각해 clock_gettime()이라는 함수에 대해 알아보았다. 구조는 아래와 같다.

```
int clock_gettime(clockid_t clk_id, struct timespec tp);
struct timespec {
    time_t   tv_sec;     /* seconds */
    long     tv_nsec;    /* nanoseconds */
};
```

clk_id 에는 clock_realtime, clock_monotonic 이 존재하며 정확한 측정을 위해 real time 의 시간이 아닌 clock_monotinic 을 사용해 코드를 작성해 보았다.

```c
#include <stdio.h>
#include <time.h>

int main(){
    struct timespec startTime, endTime;
    /*
    struct timespec{

        time_t tv_sec; //seconds
        long tv_nsec;  // nanoseconds
    }
    */
    clock_gettime(CLOCK_MONOTONIC, &startTime);
    //some operation
    int sum = 0;
    int itr = 100000;
    for (int i=0;i<itr;i++){
        sum += 1;
    }
    clock_gettime(CLOCK_MONOTONIC, &endTime);
    long sec = endTime.tv_sec - startTime.tv_sec;
    long nsec = endTime.tv_nsec - startTime.tv_nsec;
    double diffTime = sec + nsec*1e-9;
    printf("time measured : %.9f s \n", diffTime);

    return 0;
}
```

이에 대한 결과는 아래와 같았고 nanosecond 단위까지 측정 가능하다는 것을 알아내었다.

Question 2.

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <unistd.h>

int main(int argc, char *argv[]){
    struct timespec start, end;
    double diffTime = 0;

    int NUMPAGES = atoi(argv[1]);
    printf("NUMPAGES : %d \n", NUMPAGES);
    int TRIALS = atoi(argv[2]);
    printf("TRIALS : %d \n", TRIALS);
    int PAGESIZE = sysconf(_SC_PAGESIZE);
    printf("PAGESIZE : %d \n", PAGESIZE);
    int *array = (int*)malloc(NUMPAGES*PAGESIZE);
    int jump = PAGESIZE / sizeof(int);
    for (int i = 0; i<TRIALS;i++){
        clock_gettime(CLOCK_MONOTONIC, &start);
        for (int j=0;j<NUMPAGES * jump;j+=jump){
            array[j] += 1;
        }
        clock_gettime(CLOCK_MONOTONIC, &end);
        diffTime += (end.tv_sec-start.tv_sec) +(end.tv_nsec-start.tv_nsec)*1e-9;
        //printf("%.9f s", diffTime);
    }
    printf("avg time for NUMPAGES(%d), TRIALS(%d) : %.9f s\n", NUMPAGES, TRIALS, diffTime/TRIALS);
    return 0;
}
```

10 ~ 13줄 : NUMPAGES 와 TRIALS 를 각각 command line 에서 받는다.

14~15줄 : 현 운영체제인 리눅스에서의 PAGESIZE 를 가져온다. 확인 결과, 4096이었다.

16줄 : NUMPAGES * PAGESIZE 의 갯수의 integer 를 가진 array 를 malloc 을 통해 할당했다.

18줄 : 한 반복마다 PAGESIZE/sizeof(int) 를 jump 한다. 즉 한 개의 page 씩을 건너뛰는 셈이다.

19 ~ 26줄 : 총 TRIAL 의 횟수만큼 array 의 element 마다 접근하는 부분이다. Command line 에서 지정한 NUMPAGES 과 jump 를 곱한 값, 즉, 전체 설정한 갯수의 PAGE 를 전부 확인해보는 루프이다. 만약 TLB 의 크기가 설정한 NUMPAGES 보다 작다면 TLB miss 가 날 것이다.

20, 24 25 줄 : clock_gettime 함수를 이용해 각 TRIAL 당 걸리는 시간을 측정해 diffTime 에 모두 더해 저장한다.

28 : 모든 TRIAL 의 걸린 시간들을 TRIAL 로 나누어 해당 NUMPAGES 를 갖고 있을 때 평균적으로 얼마의 시간이 걸렸는 지 콘솔에 출력 해준다.

Question 3.

NUMPAGES 의 값을 1부터 2배로 증가시켜가며 위의 tlb.c 를 실행하는 bash scipt 를 아래와 같이 작성했다.

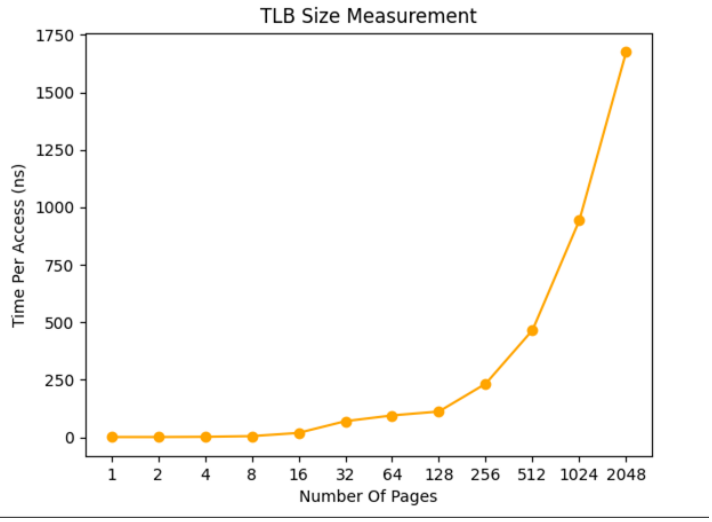
```bash
#!/bin/bash

for((i =1;i<$1;i*=2))
do
    ./a.out $i $2
    wait
done
```

위 스크립트를 실행할 때의 첫 argument 값까지 NUMPAGES 를 1부터 2배씩 증가한다.
TRIALS 는 두 번째 argument 에서 받는다.
예시로 ./run 2048 1000으로 실행한 결과는 아래와 같았다.

```
ch19hw [main] ⚡ ./run 2048 1000
NUMPAGES : 1
TRIALS : 1000
PAGESIZE : 4096
avg time for NUMPAGES(1), TRIALS(1000) : 0.000000088 s
NUMPAGES : 2
TRIALS : 1000
PAGESIZE : 4096
avg time for NUMPAGES(2), TRIALS(1000) : 0.000000106 s
NUMPAGES : 4
TRIALS : 1000
PAGESIZE : 4096
avg time for NUMPAGES(4), TRIALS(1000) : 0.000000075 s
NUMPAGES : 8
TRIALS : 1000
PAGESIZE : 4096
avg time for NUMPAGES(8), TRIALS(1000) : 0.000000100 s
NUMPAGES : 16
TRIALS : 1000
PAGESIZE : 4096
avg time for NUMPAGES(16), TRIALS(1000) : 0.000000226 s
NUMPAGES : 32
TRIALS : 1000
PAGESIZE : 4096
avg time for NUMPAGES(32), TRIALS(1000) : 0.000000744 s
NUMPAGES : 64
TRIALS : 1000
PAGESIZE : 4096
avg time for NUMPAGES(64), TRIALS(1000) : 0.000001543 s
NUMPAGES : 128
TRIALS : 1000
PAGESIZE : 4096
avg time for NUMPAGES(128), TRIALS(1000) : 0.000003062 s
NUMPAGES : 256
TRIALS : 1000
PAGESIZE : 4096
avg time for NUMPAGES(256), TRIALS(1000) : 0.000003590 s
NUMPAGES : 512
TRIALS : 1000
PAGESIZE : 4096
avg time for NUMPAGES(512), TRIALS(1000) : 0.000005148 s
NUMPAGES : 1024
TRIALS : 1000
PAGESIZE : 4096
avg time for NUMPAGES(1024), TRIALS(1000) : 0.000010558 s
```

Question 4.
아래와 같이 시각화를 위해 python 으로 코드를 작성했고 그에 따른 그래프이다.

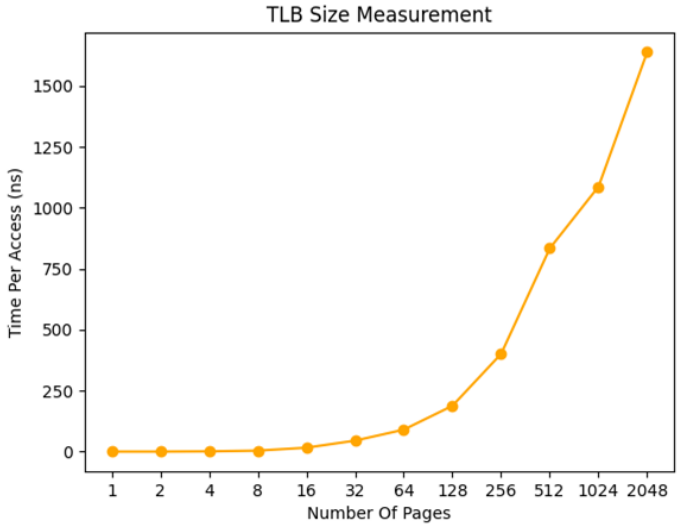
```python
from matplotlib import pyplot as plt

file = open("result.txt", "r")
List = list()
for line in file:
    line = line[:-2]
    List.append(int(float(line)*100000000))
Nlist = list()
for i in range(12):
    Nlist.append(str(pow(2,i)))

print(List)
print(Nlist)
plt.plot(Nlist, List, marker='o', color='orange')
plt.title('TLB Size Measurement')
plt.xlabel('Number Of Pages')
plt.ylabel('Time Per Access (ns)')
plt.xticks(Nlist)
plt.show()
```



수치만으로 볼 때는 보이지 않았던 페이지의 수가 늘어나며 급증하는 접근 시간에 대한 추세를 볼 수 있었다. 책에 나온 그래프처럼 TLB hierarchy level 에 따른 급격한 차이는 볼 수 없었지만, 16과 256에서 상승이 크게 일어난 것을 볼 수 있었다.

Question 5.

gcc 를 이용해 컴파일할 때, -O0 옵션을 추가해 컴파일 한 후, 결과를 살펴보았다.



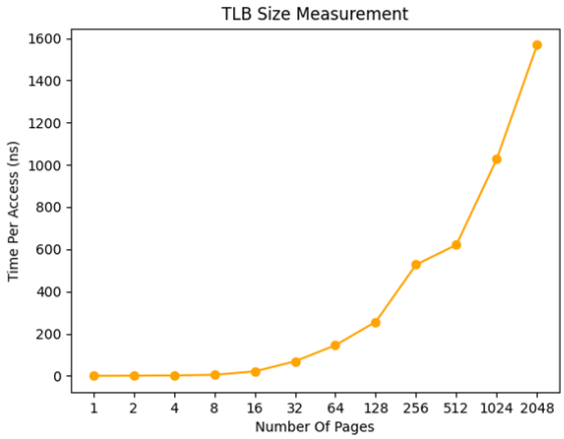
Question 6.

`pthread_setaffinity_np` 를 사용했다.

```
1 #define _GNU_SOURCE
2 #include <sched.h>
3 #include <stdio.h>
4 #include <stdlib.h>
5 #include <time.h>
6 #include <unistd.h>
7 #include <pthread.h>
8
9 int main(int argc, char* argv[]){
10     cpu_set_t cpuset;
11     pthread_t thread = pthread_self();
12     CPU_ZERO(&cpuset);
13     int a;
14     a = pthread_setaffinity_np(thread, sizeof(cpu_set_t), &cpuset);
15
```

위와 같이 10 ~ 15줄을 추가해 cpu 하나에서만 작업이 일어날 수 있게 고정하였다.

그에 따른 결과를 그래프로 나타내면 아래와 같다.



그 이전에는 2048개의 페이지 경우에서 1500 ns 정도였지만 이 경우에는 1600 ns 정도를 기록했다.

Question 7.
clock_gettime 함수를 for 루프 안에서만 측정했기 때문에 동적 할당에 대한 시간은 포함시키지 않았다. 따라서 초기화하는 부분에 대한 시간은 포함되지 않아, 위의 그래프들은 순전히 접근 시간이라고 보아도 무방하다.
calloc 을 사용해 처음 모든 array 의 값들을 0으로 만들어 코드를 돌려보았지만 결과는 같았다.