CPE 435: OPERATING SYSTEMS LABORATORY.

Lab09

Code Testing & Performance Analysis and Energy Consumption.

Submitted by: Dan Otieno.

Date of Experiment: 03/10/23.

Report Deadline: 03/24/23.

Demonstration Deadline: 03/24/23.

Introduction:

The purpose of this lab was to understand the utilities we use to debug and analyze software.

PART 1:

This part involves compiling a few test codes and analyzing them using Valgrind. As stated in the lab manual, Valgrind is a tool suite that provides a number of debugging and profiling tools. You can make your program faster and more correct using Valgrind.

Results & Observation:

Assignment 1:

Description:

The goal for this assignment was to run and analyze code samples provided in the lab manual. The outputs for those samples are shown below:

```
dpo0002prac@DESKTOP-140DFI9:/mnt/c/Users/d_oti/Desktop/CPE_CLASSES/CPE_435/Lab_9$ ./test1
array[0][0] was 0
dpo0002prac@DESKTOP-140DFI9:/mnt/c/Users/d_oti/Desktop/CPE_CLASSES/CPE_435/Lab_9$ ./test2
array[0][0] was 0
dpo0002prac@DESKTOP-140DFI9:/mnt/c/Users/d_oti/Desktop/CPE_CLASSES/CPE_435/Lab_9$
```

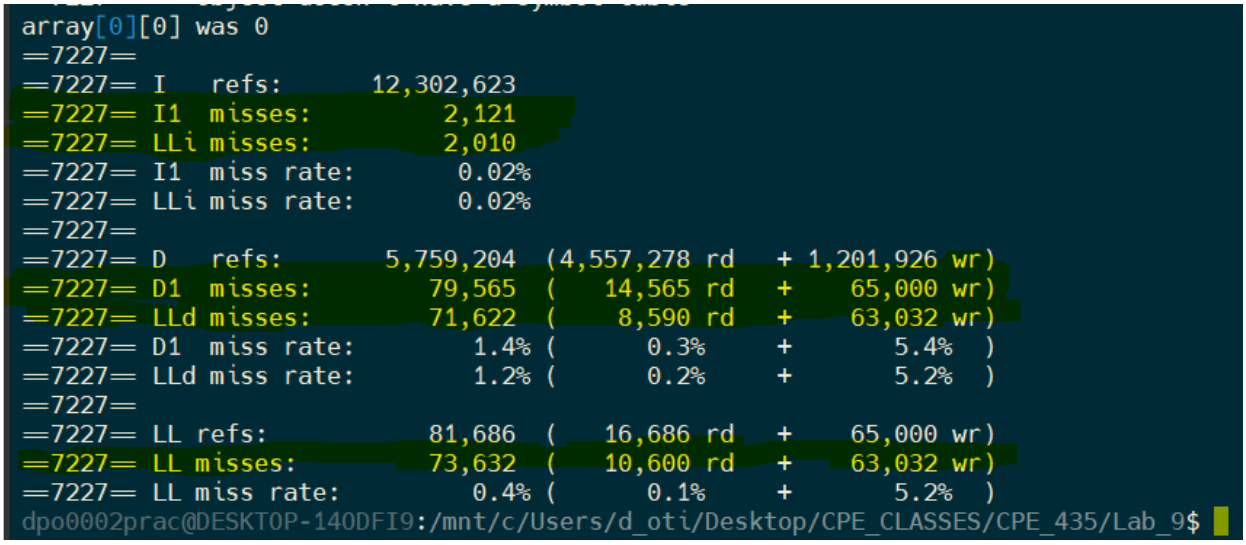
Assignment 2:

Description:

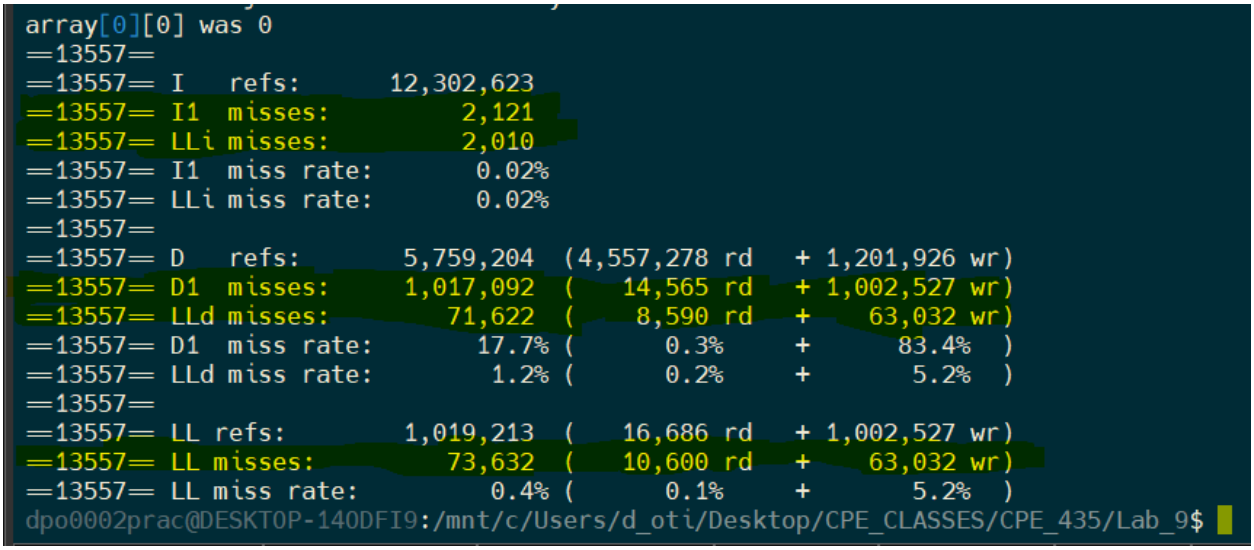
For this assignment, we run the same programs in assignment 1 with Valgrind, using the cachegrind tool. The following screenshots show the output results, which highlighted misses:

Program Outputs:

- Test1.



- Test2.



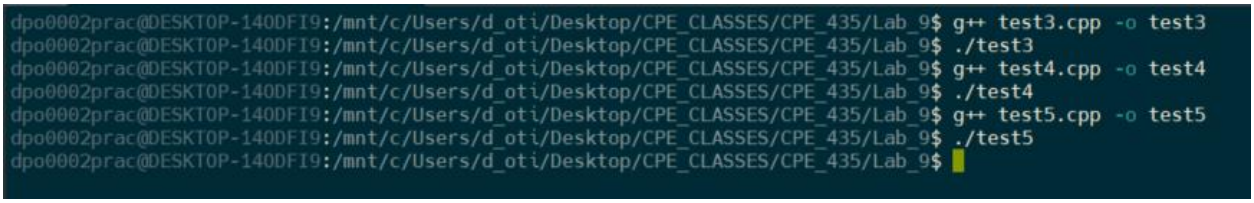
From the numbers above, test2 has more D1 misses compared to test1, therefore test1 is the better program (fewer misses) in terms of performance, other than that, there are no differences between other highlighted misses. We can also see that test2 has a much higher miss rate (17.7%) compared to that of test1(1.4%).

Assignment 3:

Description:

For this assignment, we run the codes provided in the lab manual, the following observation was made from the terminal when the files for test3, test4 and test5 were compiled and ran:

Program Outputs:



The following explain issues with the programs:

- For test3: Memory is allocated but there are no values stored in the array to carry out any operations and achieve outputs.
- For test4: We have memory allocated for an array of 10 elements (indexes 0,1…..9), however, in the array x, we are trying to store the character ‘a’ in index 10, which is out of bounds.
- For test5: x is declared as an integer datatype, but it is not initialized, so the condition defined in the if statement does not have enough criteria, i.e x is not initialized to any value, which would be the comparison with 0 to complete the if-conditions.

Assignment 4:

Description:

For this assignment, we run the same programs in assignment 3 with Valgrind, using the memcheck tool. The following screenshots show the output results:

Program Outputs:

Mecheck output for test3:

```
dpo0002prac@DESKTOP-140DFI9:/mnt/c/Users/d_oti/Desktop/CPE_CLASSES/CPE_435/Lab_9$ valgrind --tool=memcheck --leak-check=yes
 ./test3
==20936== Memcheck, a memory error detector
==20936== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==20936== Using Valgrind-3.18.1 and LibVEX; rerun with -h for copyright info
==20936== Command: ./test3
==20936==
==20936==
==20936== HEAP SUMMARY:
==20936==     in use at exit: 100 bytes in 1 blocks
==20936==   total heap usage: 2 allocs, 1 frees, 72,804 bytes allocated
==20936==
==20936== 100 bytes in 1 blocks are definitely lost in loss record 1 of 1
==20936==    at 0x484A2F3: operator new[](unsigned long) (in /usr/libexec/valgrind/vgpreload_memcheck-amd64-linux.so)
==20936==    by 0x10915E: main (in /mnt/c/Users/d_oti/Desktop/CPE_CLASSES/CPE_435/Lab_9/test3)
==20936==
==20936== LEAK SUMMARY:
==20936==    definitely lost: 100 bytes in 1 blocks
==20936==    indirectly lost: 0 bytes in 0 blocks
==20936==      possibly lost: 0 bytes in 0 blocks
==20936==    still reachable: 0 bytes in 0 blocks
==20936==         suppressed: 0 bytes in 0 blocks
==20936==
==20936== For lists of detected and suppressed errors, rerun with: -s
==20936== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
dpo0002prac@DESKTOP-140DFI9:/mnt/c/Users/d_oti/Desktop/CPE_CLASSES/CPE_435/Lab_9$
```

Memcheck output for test4:

```
dpo0002prac@DESKTOP-140DFI9:/mnt/c/Users/d_oti/Desktop/CPE_CLASSES/CPE_435/Lab_9$ valgrind --tool=memcheck --leak-check=yes
 ./test4
==22349== Memcheck, a memory error detector
==22349== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==22349== Using Valgrind-3.18.1 and LibVEX; rerun with -h for copyright info
==22349== Command: ./test4
==22349==
==22349== Invalid write of size 1
==22349==    at 0x10916B: main (in /mnt/c/Users/d_oti/Desktop/CPE_CLASSES/CPE_435/Lab_9/test4)
==22349==  Address 0x4a9104a is 0 bytes after a block of size 10 alloc'd
==22349==    at 0x4848899: malloc (in /usr/libexec/valgrind/vgpreload_memcheck-amd64-linux.so)
==22349==    by 0x10915E: main (in /mnt/c/Users/d_oti/Desktop/CPE_CLASSES/CPE_435/Lab_9/test4)
==22349==
==22349==
==22349== HEAP SUMMARY:
==22349==     in use at exit: 10 bytes in 1 blocks
==22349==   total heap usage: 1 allocs, 0 frees, 10 bytes allocated
==22349==
==22349== 10 bytes in 1 blocks are definitely lost in loss record 1 of 1
==22349==    at 0x4848899: malloc (in /usr/libexec/valgrind/vgpreload_memcheck-amd64-linux.so)
==22349==    by 0x10915E: main (in /mnt/c/Users/d_oti/Desktop/CPE_CLASSES/CPE_435/Lab_9/test4)
==22349==
==22349== LEAK SUMMARY:
==22349==    definitely lost: 10 bytes in 1 blocks
==22349==    indirectly lost: 0 bytes in 0 blocks
==22349==      possibly lost: 0 bytes in 0 blocks
==22349==    still reachable: 0 bytes in 0 blocks
==22349==         suppressed: 0 bytes in 0 blocks
==22349==
==22349== For lists of detected and suppressed errors, rerun with: -s
==22349== ERROR SUMMARY: 2 errors from 2 contexts (suppressed: 0 from 0)
dpo0002prac@DESKTOP-140DFI9:/mnt/c/Users/d_oti/Desktop/CPE_CLASSES/CPE_435/Lab_9$
```

Memcheck output for assignment 5:

```
dpo0002prac@DESKTOP-140DFI9:/mnt/c/Users/d_oti/Desktop/CPE_CLASSES/CPE_435/Lab_9$ valgrind --tool=memcheck --leak-check=yes
 ./test5
==23063== Memcheck, a memory error detector
==23063== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==23063== Using Valgrind-3.18.1 and LibVEX; rerun with -h for copyright info
==23063== Command: ./test5
==23063==
==23063== Conditional jump or move depends on uninitialised value(s)
==23063==    at 0x1091D9: main (in /mnt/c/Users/d_oti/Desktop/CPE_CLASSES/CPE_435/Lab_9/test5)
==23063==
X is zeroX is zero
==23063==
==23063== HEAP SUMMARY:
==23063==     in use at exit: 0 bytes in 0 blocks
==23063==   total heap usage: 2 allocs, 2 frees, 73,728 bytes allocated
==23063==
==23063== All heap blocks were freed -- no leaks are possible
==23063==
==23063== Use --track-origins=yes to see where uninitialised values come from
==23063== For lists of detected and suppressed errors, rerun with: -s
==23063== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
dpo0002prac@DESKTOP-140DFI9:/mnt/c/Users/d_oti/Desktop/CPE_CLASSES/CPE_435/Lab_9$
```

PART 2:

This part using a power measurement tool to determine which is the best sorting algorithm based on power consumption and work performed.

Assignment 5:

Description:

For this assignment, use C/C++ to write and implement the Quick, Merge and Insertion sorting algorithms. The source codes are included in the appendix section of this report. The programs are used for Assignments 6 and 7 as further explained in those assignment descriptions. (Source for Sorting codes: My CS 317 project from a previous semester).

Assignment 6:

Description:

For this assignment, we use the provided rapl-read.c file to implement the sorting algorithms written for Assignment 5. The minimum sample requirements are at least 1,000,000, except for insertion sort. We are required to also run a timer to make sure we achieve a 5-second run time. The following table below details the data collected from 5 trial runs. For Insertion sort, we use 210,000 samples, MergeSort, 35,000,000 samples, and 48,000,000 samples for QuickSort. In the command terminal, we compile rapl-read with the -O flag, i.e. “gcc -O0 -Wall -o rapl-read rapl-read.c -lm.”

Program Outputs:

Trial	Algorithm					
	Insertionsort (210,000)		Mergesort (35,000,000)		Quicksort(48,000,000)	
	Energy(J)	Time(s)	Energy(J)	Time(s)	Energy(J)	Time(s)
1	120.750	5.337	103.230	5.366	98.163	5.142
2	119.984	5.340	102.983	5.356	97.116	5.159
3	126.233	5.343	102.915	5.366	97.086	5.123
4	119.795	5.312	102.717	5.366	97.736	5.183
5	120.908	5.322	104.762	5.379	97.163	5.149
Average Energy	121.534		103.3214		97.4528	
Energy per element	0.000579		2.952E-06		2.030E-06	
Average Time	5.3308		5.3666		5.1512	
Time per element	2.538E-05		1.533E-07		1.073E-07	

Assignment 7:

Description:

This assignment is similar to Assignment 6, we use the provided rapl-read.c file to implement the sorting algorithms written for Assignment 5. The minimum sample requirements are at least 1,000,000, except for insertion sort. We are required to also run a timer to make sure we achieve a 5-second run time. The following table below details the data collected from 5 trial runs. For Insertion sort, we use 250,000 samples, MergeSort, 35,000,000 samples, and 50,000,000 samples for QuickSort. In the command terminal, we compile rapl-read with the -O flag, i.e. "gcc -O3 -Wall -o rapl-read rapl-read.c -lm."

Program Outputs:

Trial	Algorithm					
	Insertionsort (250,000)		Mergesort (35,000,000)		Quicksort(50,000,000)	
	Energy(J)	Time(s)	Energy(J)	Time(s)	Energy(J)	Time(s)
1	120.602	5.659	104.470	5.483	97.608	5.156
2	120.646	5.673	103.643	5.481	96.552	5.124
3	120.639	5.679	103.982	5.472	98.698	5.139
4	121.874	5.676	104.231	5.469	96.373	5.118
5	121.379	5.671	103.906	5.476	96.070	5.116
Average Energy	121.028		104.046		97.060	
Energy per element	0.000484		2.973E-06		1.941E-06	
AverageTime	5.6716		5.4762		5.1306	
Time per element	2.269E-05		1.565E-07		1.026E-07	

Assignment 8:

Response:

In Assignments 6 and 7, Quicksort is the better algorithm, both in terms of energy consumption per element and time per element. The compiler flag does not make much of a difference, at least not from my trial runs, the energy and time consumption is almost the same, albeit with very slight differences. The fact that Quicksort has the best energy and time consumption is not surprising, considering it is the fastest sorting algorithm of the three with a smaller best-case time complexity (see table below). Table source: https://www.codeproject.com/Articles/5308420/Comparison-of-Sorting-Algorithms

Sorting Algorithm	Average Case	Best Case	Worst Case
Bubble Sort	$O(n^2)$	O(n)	$O(n^2)$
Insertion Sort	$O(n^2)$	O(n)	$O(n^2)$
Selection Sort	$O(n^2)$	$O(n^2)$	$O(n^2)$
Quick Sort	O(n.log(n))	O(n.log(n))	$O(n^2)$
Merge Sort	O(n.log(n))	O(n.log(n))	O(n.log(n))
Heap Sort	O(n.log(n))	O(n.log(n))	O(n.log(n))
Counting Sort	O(n+k)	O(n+k)	O(n+k)
Radix Sort	O(n*k)	O(n*k)	O(n*k)
Bucket Sort	O(n+k)	O(n+k)	$O(n^2)$

Extra Credit Research:

MSR (Model-Specific Register) registers are control registers in the x86 architecture that are used for debugging, program execution tracing, computer performance monitoring, and toggling certain CPU features. The "rdmsr" is used for instructions to read into the MSR, while the "wrmsr" handles write instructions. A user with permissions to read and/or write to this file can use the file I/O API to access these registers. /dev/cpu/CPUNUM/msr provides an interface to read and write the model-specific registers (MSRs) of an x86 CPU. The register access is done by opening the file and seeking to the MSR number as offset in the file, and then reading or writing in chunks of 8 bytes. An I/O transfer of more than 8 bytes means multiple reads or writes of the same register. This file is protected so that it can be read and written only by the user root, or members of the group root. (SOURCE 1, SOURCE 2).

Conclusion:

All programs worked as expected. I learned something new about performance analysis, I also did not know about rapl-read, as I have always used HW Monitor, but rapl-read is quite easy to use, at least as was the case of this lab.

Appendix:

Assignment 5 – Sorting/Top Level/Verification code.

```
/*
DAN OTIENO
CPE 435-01
LAB 9
Assignment V, VI, VII.
*/

#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <sys/time.h>
#include <ctype.h>
#include <unistd.h>
#include <string.h>

#define TIMER_CLEAR (tv1.tv_sec = tv1.tv_usec = tv2.tv_sec = tv2.tv_usec =
0)
#define TIMER_START gettimeofday(&tv1, (struct timezone*)0)
#define TIMER_ELAPSED (double) (tv2.tv_usec-
tv1.tv_usec)/1000000.0+(tv2.tv_sec-tv1.tv_sec)
#define TIMER_STOP gettimeofday(&tv2, (struct timezone*)0)
struct timeval tv1,tv2;
long int array1[50000000]; /* this is ok it is from global memory */
long int array2[35000000];
long int array3[250000];

/* Function declarations */
void quick_sort(long int arr[], int left, int right);
void q_verify(long int arr[], int n);
void merge_sort(int arr[], int l, int r);
void merge(int arr[], int l, int m, int r);
void m_verify(int arr[], int n);
void insertion_sort(long arr[], long n);
int i_verify(int arr[], int n);

void quick_sort_top_level(long int n)
{
    array1[n];
    int i;
    srand(time(NULL)); // seed the random number generator with the current
time
    for (i = 0; i < n; i++)
    {
        array1[i] = rand();
    }

    TIMER_CLEAR;
    TIMER_START;
    quick_sort(array1, 0, n-1);
    q_verify(array1, n);
    TIMER_STOP;
```

```
    printf("Time elapsed = %f seconds\n", TIMER_ELAPSED);
}

/* Top level function */
void merge_sort_top_level(long int n)
{
    array2[n];
    int i;
    /* Generate n random integers */
    srand(time(NULL));
    for (i = 0; i < n; i++)
    {
        array2[i] = rand();
    }
    TIMER_CLEAR;
    TIMER_START;
    merge_sort(array2, 0, n - 1);
    m_verify(array2, n);
    TIMER_STOP;
    printf("Time elapsed = %f seconds\n", TIMER_ELAPSED);
}

void insertion_sort_top_level(long int n)
{
    long int i;
    srand(time(NULL));

    // Generate n random integers
    for (i = 0; i < n; i++)
    {
        array3[i] = rand();
    }

    TIMER_CLEAR;
    TIMER_START;
    insertion_sort(array3, n);
    i_verify(array3, n);
    TIMER_STOP;
    printf("Time elapsed = %f seconds\n", TIMER_ELAPSED);
}

void quick_sort(long int arr[], int left, int right)
{
    if (left < right)
    {
        long int pivot = arr[(left + right) / 2];
        int i = left, j = right;
        while (i <= j)
        {
            while (arr[i] < pivot) i++;
            while (arr[j] > pivot) j--;
            if (i <= j)
            {
                long int tmp = arr[i];
                arr[i] = arr[j];
                arr[j] = tmp;
                i++;
```

```
                j--;
            }
        }
        quick_sort(arr, left, j);
        quick_sort(arr, i, right);
    }
}

void q_verify(long int arr[], int n)
{
    int i;
    for (i = 0; i < n-1; i++)
    {
        if (arr[i] > arr[i+1])
        {
            printf("Sorting failed at index %d\n", i);
            return;
        }
    }
    printf("Sorting succeeded!\n");
}


/* Merge sort implementation */
void merge_sort(int arr[], int l, int r)
{
    if (l < r)
    {
        int m = (l + r) / 2;
        merge_sort(arr, l, m);
        merge_sort(arr, m + 1, r);
        merge(arr, l, m, r);
    }
}

/* Merge two sorted subarrays into a larger sorted array */
void merge(int arr[], int l, int m, int r)
{
    int i, j, k;
    int n1 = m - l + 1;
    int n2 = r - m;

    //int L[n1], R[n2];
    int* L = (int*)malloc(sizeof(long int) * n1);
    int* R = (int*)malloc(sizeof(long int) * n2);

    for (i = 0; i < n1; i++)
    {
        L[i] = arr[l + i];
    }
    for (j = 0; j < n2; j++)
    {
        R[j] = arr[m + 1 + j];
    }

    i = 0;
    j = 0;
```

```
    k = l;

    while (i < n1 && j < n2)
    {
        if (L[i] <= R[j])
        {
            arr[k] = L[i];
            i++;
        }
        else
        {
            arr[k] = R[j];
            j++;
        }
        k++;
    }

    while (i < n1)
    {
        arr[k] = L[i];
        i++;
        k++;
    }

    while (j < n2)
    {
        arr[k] = R[j];
        j++;
        k++;
    }
}

/* Verify if the array is sorted */
void m_verify(int arr[], int n)
{
    int i;
    for (i = 1; i < n; i++)
    {
        if (arr[i] < arr[i - 1])
        {
            printf("Error: Sorting failed!\n");
            return;
        }
    }
    printf("Sorting successful!\n");
}

void insertion_sort(long int arr[], long int n)
{
   int i, j, key;
   for (i = 1; i < n; i++)
   {
       key = arr[i];
       j = i - 1;
       while (j >= 0 && arr[j] > key)
       {
           arr[j + 1] = arr[j];
```

```
            j = j - 1;
        }
        arr[j + 1] = key;
    }
}

int i_verify(int arr[], int n)
{
    int i;
    for (i = 1; i < n; i++)
    {
        if (arr[i] < arr[i - 1])
        {
            printf("Sorting successful!\n");
            return 0;
        }
    }
    printf("Error: Sorting failed!\n");
}
```