Assignment 4 Report

COMP 3300

03/06/2020

Bilal Malik

104435995

Hasan Malik

10461046

Task 1

The program begins by taking in a the argument and converting it to a long long integer through the atoll() function, this way it can be stored in a 32-bit virtual address. An if statement is then ran, to check if any arguments were passed outside of the executable, and if this is the case; to notify the user. A bit shift occurs on the address variable and this is stored as the page number in pNum variable holder. The address is shifted to the right by 12 bits, 0xfff is the number 65535 in hexadecimal and & performs a bitwise and operation to get the value where it needs to be. The right shift occurs to set the least count number of bits to fetch offset with a for loop. Both these computations leave us with an accurate page number and offset from the beginning of that page number respectively. The values are finally printed out. In summary, a program that takes a decimal virtual address and outputs the page number and offset for the given address.

The following are the five different runs with varying arguments in the part 1 of the program, we start including a run that matches the sample execution in the assignment document, that is the value 19986.

A screenshot of a computer screen

Description automatically generated

The output for thus run matches as well. The next is run is by a reduction of 9000, and this can be seen by the smaller number of pages being necessary to search through.

A screenshot of a computer screen

Description automatically generated

The program is clearly working successfully, as the decrease in size of the virtual address means less defaulting occurs, and that less pages need to be occupied to completely fill the virtual address.

A screenshot of a computer screen

Description automatically generated

We can find the smallest offset in a page number; with this we can also see if the shifting is working correctly as reduction in memory by only one should understand reduce page number as it cannot reduce the offset.

A screenshot of a computer screen

Description automatically generated

Finally, a run much larger than the first, to show the page number and offset is truly relative to address space as it increases with this as input.

A screenshot of a computer screen

Description automatically generated

Task 2

The program begins by declare all the necessary variables, including the pagenumber offset and address all to be computed in a for loop. Srandom() is utilized give a sequence of random numbers. A clock is called before the loop executes; this returns the number of clock ticks elapsed since the beginning of the program at that time. Inside of the loop, a random number is given as address and the page number and offset a calculated from there in the same way they are calculated in part 1. Once the for-loop ends, the clock is called again. The two times are subtracted, leaving the time inside the loop. The clock function only returns the number of clocks ticks and needs to be divided by the CLOCKS\_PER\_SEC so find the time in seconds spent. Finally the CPU time is printed out, in summary after 1 million laps in the loop calculated offset and page number, the time it takes to complete this task is then printed, that is , this program generates the random virtual address between 0 and 2^32-1 and then computes the page number and offset for each address. For all the random virtual address, the program outputs the CPU time of.

Once the program was changed and saved as assignmentpart2, I ran it by first compiling through gcc and the name of the program. Since, I did not specify and executable, I will be given a.exe by default. I execute this executable through ./a.exe and received a total CPU time of 0.015000 seconds.

A screenshot of a computer screen

Description automatically generated

This time was figured by returning the clock() return value at two separate points in the function, before and after the for loop. Subtracting them to get the time inside the for loop, this time is clock ticks taken to complete the task and needs to be divided by CLOCKS\_PER\_SEC to get the number of seconds. Running through 1-million random addresses is a difficult task, on top of this finding its random location in a page and offset location while the random address can be anywhere from 1 to 1 million makes this a daunting task. Yet the CPU is able to continually clock in string times. From just over a tenth of a second, it can be seen that the Operating system does not hinder in performance from the page faulting as seriously as we may assume.

A screenshot of a computer screen

Description automatically generated

Other runs show the CPU time staying consistently only climbing up on cases where usage is heavier and busier.

Task3

The program begins by declaring all necessary variables, then taking in arguments as inputs fir address space, page size and address. It does this by converting the string from atoi() into an integer. Then a series of if statements follow to check whether the arguments are valid values, and whether address space and page size are numbers with base 2. Once all the inputs have met all necessary checks, the page size is multiplied by 1024 bytes (one kilobyte) then seeing if the page size is a power of two, by multiplying it by two until it reaches page size. Finally, the page offset and page number are calculated using shifting and AND Boolean logic like part one and two, finally these values are printed. In summary, the program takes three inputs, checks if the first two are powers of two, and other checks before doing the computation seen before and printing out values, that is page number and offset is computed based on input unlike the previous tasks, that is this is a modification of part 1, that takes an address-space, page size, and an address as an input address. This program also checks for powers of 2 for v and s, and if s is less than v, and whether v = 16, 32, 0r 64. If s is between 514 bytes and 1gb.

The following are five runs with varying inputs, to show the program handling of different parameters:

A screenshot of a computer screen

Description automatically generated

This execution involves 32 as an address space, 16 as page size, and 101245 as address. It responds with 6-page numbers, and an offset of 2952, which is correct. By decreasing page size, it can be seen that the page numbers decrease as well.

A screenshot of a computer screen

Description automatically generated

As well, we can see the error handling dealing with the wrong inputs, in this case, the number for address space must be greater than page size. As the case fails, the output notifies the user.

A screenshot of a computer screen

Description automatically generated

Another case is where either address space or page size is not a power of two. Both these situations are handled here.

A screenshot of a computer screen

Description automatically generated

Finally, putting the incorrect number of inputs, directs you to the correct method to input.

A screenshot of a computer screen

Description automatically generated

Thus it takes into account all the different situations with inputs ensuring they are correct when calculation is to occur.