Project 2

Hazael Magino

2/23/2024

VV12417

The Goal of this program is to practice the development of file I/O and arithmetic calculations in low level architecture(x86).

Milestone 1:

In this program the desired goal was to show the implementation of the following tasks:

- Be able to open read and write from a file.
- Sum up integers and output the result.
- Sort the numbers in a programmable fashion such that they output in ascending order.

File I/O

While developing this program, I learned to utilize file I/O in assembly language. This process required extensive use of system calls and memory management. To calculate sums, it is necessary to open the file and set the appropriate system calls for reading. I employed the stack to establish predetermined buffer sizes and to designate storage locations for the data read from the file, as well as for the filename that the program will access. After determining the buffer sizes, system calls were executed to establish the necessary file permissions. Each register, such as EAX, had its own specific calls to achieve the desired outcomes. These calls were denoted by single digits ranging from 1 to 9, with each number assigned a specific task for the register. In addition to using system calls, a special type of pointer known as a descriptor was utilized to read each bit. In this code, the variable “ltoread” was responsible for this task, aiding in buffer management. Below is the code that I used for this part of the project:

```
start:
    jmp open



open:
;open the text file to read from:


    mov ebx, filename ; const char *filename
    mov eax, 5 ;set to open file
    mov ecx, 0 ; set to read only mode
    int 80h

    jmp read
```

Project 2

Hazael Magino

2/23/2024

VV12417

```
read:

    mov [readp], ebx ; allows to stores the file descriptor into the memory
location

    mov eax, 3 ; sys_read call
    mov ebx, eax ; set readp as input for read operation
    mov ecx, buffer ;load memory address of where to store data
    mov edx, ltoread ; Max amount of bytes to read
    int 80h

    jmp write

convert:
;convert each string to a number by parsing and shifting for numeric value

    mov esi, edx
    lodsb               ; Load the next character into AL
    sub al, '0'         ; Convert ASCII to integer ('0' -> 0, '1' -> 1, ..., '9'
-> 9)


    test al, al         ; Check if we've reached the end of the string (null
terminator)

    jnz convert         ; If not, continue the loop

write:

    mov eax, 4 ;call for sys_write
    mov ebx, 1 ;set readp for standard output
    mov ecx, buffer ; load data read to be written to output
    mov edx, ltoread ; Number of bytes to write
    int 80h
```

Sum to print:

In this segment of the program, arithmetic computations and strategic logic were necessary for the program to function as expected. Since we are reading strings, it was important to account for the fact that the numbers being read were strings, not decimal values. Therefore, a conversion between ASCII

Project 2

Hazael Magino

2/23/2024

VV12417

and decimal values was required. This part of the program aimed to manipulate bits and ASCII values to accumulate the sum. This process is similar to calculating the Hamming distance, except now we must anticipate the value of each bit position. Due to the nature of x86 architecture, we can only read one bit at a time. One challenge was anticipating the value of the number while managing the storage of these values and tracking the buffer's current position. This portion of the program was not fully completed, as difficulties arose when attempting to loop under certain conditions. External resources like Stack Overflow and YouTube were consulted during implementation, but a complete solution to the problem remained elusive. Below are the implementations of this portion, as well as a screenshot showing how sum values were retained and accumulated in specific registers:

```
resetReg:

;Needed in order to effectively convert asci values

    xor dx, dx ; initialize for division DX: AX/BX
    mov bx, 10
    xor cx, cx ; act as a counter, need to reset
    mov esi, buffer ;point esi to buffer


convertA:

    div bx ; AX is quotient, remainder is in DX (0-9)
    push dx ; save remainder to memory
    test ax, ax ; see if zero
    je exit ; if ax is 0, jump to exit
    jl exit ;
    jnz convertA ; No, use as the number to divide by on next loop

convertB:

    ;convert each string to a number by parsing and shifting for numeric value
    ;This portion is the dividedend

    pop dx ;should be first on the stack
    add dl, "0" ;Turn into character [0,9] -> ["0", "9"]
    mov eax, 4 ;sys_write call
    mov ebx, 1; file descriptor 1 for output
    mov [buffer], dl ; move the character to the buffer
```

Project 2

Hazael Magino

2/23/2024

VV12417

```
    int 0x80
    loop convertB
```

```
convertA:

    div bx ; AX is quotient, remainder is in DX (0-9)
    push dx ; save remainder to memory
    test ax, ax ; see if zero
    je exit ; if ax is 0, jump to exit
    jl exit ;
    jnz convertA ; No, use as the number to divide by on next loop

convertB:

    ;convert each string to a number by parsing and shifting for numeric value
    ;This portion is the dividedend
```

```
BLEMS   OUTPUT   DEBUG CONSOLE   PORTS
MINAL

83          test ax, ax ; see if zero
(gdb) s
84          je exit ; if ax is 0, jump to exit
(gdb) s
85          jl exit ;
(gdb) i r eax
eax            0xcc               204
(gdb) i r bx
bx             0xa                10
(gdb) snip
Undefined command: "snip".  Try "help".
(gdb)
```

Sort Methodology:

The final part of this project involves sorting the data in ascending order. Although I did not complete this part, I can suggest a potential implementation. To sort the data, it's crucial to monitor our position within the sort and the highest number encountered in the transposition of data. Bubble sort would be the optimal sorting method for this task. It is the simplest sorting algorithm to implement and manage, especially with limited low-level resources and constrained syntax.

Milestone 2

Code Implementation:

```
section .data:

    fmt2 db "The total value of all the sums is: $d"
    filename db "./randomInt10.txt"
    bufsize dw 1024
```

Project 2

Hazael Magino

2/23/2024

VV12417

```
section .bss

readp resb 4
ltoread resb 2000 ;for the file name
buffer: resb 2000 ;the max amount of character to be read from file
sum resb 4 ;the amount of space to calculate the sum


section .text
    global _start

_start:
    jmp open



open:
;open the text file to read from:


    mov ebx, filename ; const char *filename
    mov eax, 5 ;set to open file
    mov ecx, 0 ; set to read only mode
    int 80h

    jmp read

read:

    mov [readp], ebx ; allows to stores the file descriptor into the memory
location

    mov eax, 3 ; sys_read call
    mov ebx, eax ; set readp as input for read operation
    mov ecx, buffer ;load memory address of where to store data
    mov edx, ltoread ; Max amount of bytes to read
    int 80h

    jmp write
```

Project 2

Hazael Magino

2/23/2024

VV12417

```
convert:
;convert each string to a number by parsing and shifting for numeric value

    mov esi, edx
    lodsb               ; Load the next character into AL
    sub al, '0'         ; Convert ASCII to integer ('0' -> 0, '1' -> 1, ..., '9'
-> 9)


    test al, al         ; Check if we've reached the end of the string (null
terminator)

    jnz convert         ; If not, continue the loop

write:

    mov eax, 4 ;call for sys_write
    mov ebx, 1 ;set readp for standard output
    mov ecx, buffer ; load data read to be written to output
    mov edx, ltoread ; Number of bytes to write
    int 80h

resetReg:

;Needed in order to effectively convert asci values

    xor dx, dx ; initialize for division DX: AX/BX
    mov bx, 10
    xor cx, cx ; act as a counter, need to reset
    mov esi, buffer ;point esi to buffer


convertA:

    div bx ; AX is quotient, remainder is in DX (0-9)
    push dx ; save remainder to memory
    test ax, ax ; see if zero
    je exit ; if ax is 0, jump to exit
    jl exit ;
    jnz convertA ; No, use as the number to divide by on next loop
```

Project 2

Hazael Magino

2/23/2024

VV12417

```
convertB:

    ;convert each string to a number by parsing and shifting for numeric value
    ;This portion is the dividedend

    pop dx ;should be first on the stack
    add dl, "0" ;Turn into character [0,9] -> ["0", "9"]
    mov eax, 4 ;sys_write call
    mov ebx, 1; file descriptor 1 for output
    mov [buffer], dl ; move the character to the buffer

    int 0x80
    loop convertB

write2:
    add [sum], dx
    add [sum], bx
    mov eax, 4  ; System call number for sys_write
    mov ebx, 1  ; File descriptor for standard output
    mov ecx, fmt2  ; Move the address of the format string into ECX
    int 0x80

    mov eax, 4  ; System call number for sys_write
    mov ebx, 1  ; File descriptor for standard output
    mov edx, [sum]  ; Move the sum into EDX
    int 80h  ; Interrupt to invoke the system call

exit:

    mov eax, 6  ;close the file
    mov ebx, [readp]
    mov eax, 1  ;exit
    mov ebx, 0
    int 0x80
```

Project 2

Hazael Magino

2/23/2024

VV12417

Output:

*This was the final outcome of the produced code with being able to print file stream. Calculating the sum was attempted but output was not able to be produced due to internal logic. Sort method was not attempted.

Project 2

Hazael Magino

2/23/2024

VV12417