CS 322-Lab 6

Name: P.V. Sriram

Roll No.: 1801CS37

Task -1

# Multiplication

**Assembly**

**Compile:** gcc -m32 -o asm.out mult16m.c mult16inline.c

**Run:** ./asm.out

We shall perform the experiment for different count calls varying from 1 Million to 100 Millions and shall do every count thrice for better accuracy .

|  |  |
| --- | --- |
| Number of Calls (in Millions) | Time (Seconds) |
| 1 | Avg (0.005215, 0.004394, 0.004467) = 0.004692 |
| 10 | Avg (0.044866, 0.045073, 0.046061) = 0.045333 |
| 20 | Avg (0.089310, 0.088897, 0.089318) = 0.089175 |
| 40 | Avg (0.178046, 0.177199, 0.177253) = 0.177749 |
| 60 | Avg (0.266487, 0.265704, 0.271177) = 0.267789 |
| 80 | Avg (0.354266, 0.356430, 0.355106) = 0.355267 |
| 100 | Avg (0.441704, 0.441847, 0.441906) = 0.441819 |

**C**

**Compile:** gcc -O2 -o c.out mult16m.c mult16c.c

**Run:** ./c.out

|  |  |
| --- | --- |
| Number of Calls (in Millions) | Time (Seconds) |
| 1 | Avg (0.012545, 0.013112,0.012461) = 0.012706 |
| 10 | Avg (0.127199,0.128420,0.136273) = 0.130631 |
| 20 | Avg (0.252632,0.253093,0.265829) = 0.257185 |
| 40 | Avg (0.522774,0.540728,0.479266) = 0.514256 |
| 60 | Avg (0.802085,0.756405,0.772613) = 0.777034 |
| 80 | Avg (1.040835,1.065291,1.007769) = 1.037965 |
| 100 | Avg (1.342858,1.235574,1.293069) = 1.290500 |

We can clearly see the increase in performance in Assembly level language than in C. Almost thrice the performance boost every time. We can visualize the same in the following graph.

# Factorial

**Assembly**

**Compile:** gcc -O2 -o asm.out L6\_factorial\_main\_t.c L6\_factorial.o

**Run:** ./asm.out

We shall perform the experiment for different intervals of slots varying from 0 - 1 lakh to 5 – 6 lakhs.

|  |  |
| --- | --- |
| Factorials intervals (in Lakhs) | Time (Seconds) |
| 0 - 1 | 11.393353 |
| 1 - 2 | 62.090787 |
| 2 - 3 | 92.309978 |
| 3 - 4 | 141.249739 |
| 4 - 5 | 170.425378 |
| 5 - 6 | Segmentation Fault |

We can see that the segmentation fault occurs in the slot of 5 – 6 Lakhs

For more precision

5.5 - 198.148304

5.75 - 212.134723

5.875 - 251.752365

5.9375 - 240.476745

Segmentation Fault occurs in the interval 5,93,750 to 6,00,000.

We shall perform the experiment for different count calls varying from 0-1 Lakhs to 5-6 lakhs and shall do every count thrice for better accuracy .

|  |  |
| --- | --- |
| Factorials intervals (in 10,000s) | Time (Seconds) |
| 0 - 1 | Avg (0.102203, 0.102344, 0.101854) = 0.102134 |
| 1 - 2 | Avg (0.305439, 0.306417, 0.306355) = 0.306070 |
| 2 - 3 | Avg (0.511903, 0.513004, 0.523270) = 0.516059 |
| 3 - 4 | Avg (0.717578, 0.730165, 0.725215) = 0.724319 |
| 4 - 5 | Avg (0.930757, 0.939116, 0.927911) = 0.932595 |
| 5 - 6 | Avg (1.146066, 1.144193, 1.139747) = 1.143353 |

**C**

**Compile:** gcc -m32 -o c.out L6\_factorial\_main\_t\_c.c

**Run:** ./c.out

|  |  |
| --- | --- |
| Factorials intervals (in Lakhs) | Time (Seconds) |
| 0 - 1 | 25.701154 |
| 1 - 2 | 104.731994 |
| 2 - 3 | Segmentation Fault |
| 3 - 4 | Segmentation Fault |
| 4 - 5 | Segmentation Fault |
| 5 - 6 | Segmentation Fault |

We can see that we reach the segmentation fault much earlier on C.

For more precision:

2.5 - 110.310673

2.75 - 101.906232

2.875 - 102.274180

2.9375 – Segmentation Fault

Segmentation fault occurs in the interval 2,87,500 to 2,93,750.

|  |  |
| --- | --- |
| Factorials intervals (in 10,000s) | Time (Seconds) |
| 0-1 | Avg (0.216319, 0.218780 0.219218) = 0.218106 |
| 1-2 | Avg (0.669255, 0.668785, 0.665861) = 0.667967 |
| 2-3 | Avg (1.108237, 1.122435, 1.182598) = 1.137757 |
| 3-4 | Avg (1.597104, 1.565141, 1.599648) = 1.587298 |
| 4-5 | Avg (2.061260, 2.043004, 2.036578) = 2.046947 |
| 5-6 | Avg (2.544237, 2.582543, 2.615619) = 2.580800 |

We can clearly see the increase in performance in Assembly level language than in C. Almost twice the performance boost every time. We can visualize the same in the following graph.

Task - 2

# The Sorting Hat

The sorting hat game is inspired from the sorting hat in Harry potter. The Sorting hat is responsible to assign newly enrolled students into the Houses of Gryffindor, Slytherin, Ravenclaw, Hufflepuff. The hat is placed on the student’s head and based on the personality; it speaks out the best suited house. But in some cases, such as Harry Potter himself, the hat gets confused in between the best options. It couldn’t decide in between and “Slytherin” and “Gryffindor”, at the end decided to choose Gryffindor just because his father belonged to it.

We will help the Sorting hat to take its decision using an assembly level program. If there’s a confusion between Gryffindor and Slytherin.

# Algorithm

We first take the first name of the student. And then we search for the substring “ram” within the name. If there is a substring “ram” within the name, we assign Gryffindor, else Slytherin.

**Read Until Enter**: Calls system interrupt for keyboard input. We take in a maximum of 10 characters and place them in the reserved memory “buffer”.

The string “ram” is stored in a variable location “subs”.

**Find**: In this function, we loop through the characters of the first name and see for a match in the first character of the name and ram. If there is a match, we jump to eql function else, we jump to not\_eql function.

**Eql:** If there’s a match we see the next characters in both name, and ram. And do this three times for a continuous match for r, a, m by incrementing esi, edi both. If we don’t find such a match, we make edi 0.

**Not\_eql**: If the present characters don’t match, we make edi 0 and search for “r” all over.

**Passed**: Prints the string “Gryffindor” if a match is found

**Not** **Passed**: Prints the string “Slytherin” if a match is not found

Text

Description automatically generated

Graphical user interface, application

Description automatically generated

Text

Description automatically generated

Graphical user interface, text, application

Description automatically generated

# 64-Bit Assembly Code

section .data

subs db "ram"

start db "Please enter your first name...", 0, 10

Pass\_Okay db 0Dh , 0Ah ,"Gryffindor", 0, 10

Pass\_fail db 0Dh , 0Ah ,"Slytherin", 0, 10

len1 : equ $-start

len2 : equ $-Pass\_Okay

len3 : equ $-Pass\_fail

section .bss

buffer resb 10

buffB resb 1

section .text

global \_start

\_start:

mov rcx, 0

mov rsi, 0

mov rax, 1 ;Print starting message

mov rdi, 1

mov rsi, start

mov rdx, 31

syscall

readUntilEnter: ;Read input into buffer

mov rax, 0

mov rdi, 0

mov rsi, buffer

mov rdx, 10

syscall

PassOK: ;Check length ok

mov rdx, 10

mov rsi, 0

mov rdi, 0

mov rcx, 3

find: ;Start searching for first match

MOV al, byte[buffer + rsi]

MOV ah, byte[subs + rdi]

cmp al, ah

je eql

jne not\_eql

not\_eql: ;If not equal reset search

inc rsi

dec rdx

mov rdi, 0

jz not\_passed

jnz find

eql: ;If equal, search for continuity

inc rsi

inc rdi

dec rcx

jz passed

dec rdx

jz not\_passed

MOV al, [buffer + esi]

MOV ah, [subs + edi]

cmp al, ah

je eql

mov rdi, 0

jne find

passed: ;Print substring exist message

mov rax, 1

mov rdi, 1

mov rsi, Pass\_Okay

mov rdx, 12

syscall

jmp exit

not\_passed: ;Print substring not found message

mov rax, 1

mov rdi, 1

mov rsi, Pass\_fail

mov rdx, 12

syscall

jmp exit

exit: ;Terminate

mov rax, 60 ; return 0 status on exit - 'No Errors'

mov rdx, 0 ; invoke SYS\_EXIT (kernel opcode 1)

syscall

# C Code

#include <stdio.h>

#include <string.h>

int check\_sub(char\* name, char\* subs)

{

int j = 0;

for (int i = 0; i < strlen(name); i++)

{

if(name[i] == subs[j])

{

j++;

if(j == 3)

{

printf("Gryffindor\n");

return(0);

}

}

}

printf("Slytherin\n");

return(0);

}

int main()

{

char name[10];

char subs[3] = "ram";

scanf("%s", name);

check\_sub(name, subs);

}