

11711BCC008

Matheus José da Costa

1- Para esta questão, foram utilizados os seguintes compiladores a partir do Sistema Operacional Ubuntu 20.04: gcc e clang. Embora o gcc e o clang sejam programas diferentes, ambos tem o objetivo de converter o código fonte em código objeto.

Para a execução do programa e geração do Assembly, foi usado os seguintes comandos:

```
gcc -S atividade1.c -o atividade1-gcc.s
clang -S atividade1.c -o atividade1-clang.s
```

Durante este exercício, algo perceptível no Assembly gerado foi nas instruções básicas geradas por cada um e também o tamanho do arquivo, que no caso o tamanho do arquivo **atividade1-clang.s** é maior que o arquivo **atividade1-gcc.s**.

2-

Para executar o código desta questão, necessitei instalar algumas libs para sistema 32 bits, porque o meu sistema é de 64 bits e estava dando erro de output (utilizei o comando **sudo apt-get install gcc-multilib**). Após isso, ao compilar, necessitei especificar para o gcc que queria compilar o código para o formato de 32 bits, justamente porque o **passcode.o** fornecido está para o formato de 32 bits. O comando então ficou o seguinte: **gcc -m32 -o atividade2 passcode.o atividade2.c**

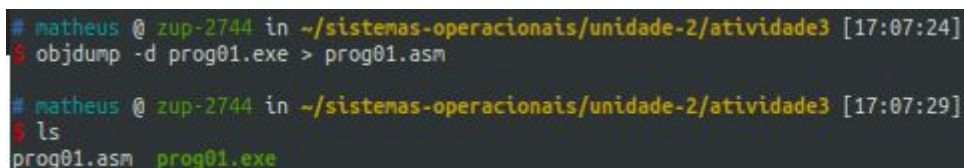
Executando o programa **atividade2** tive o resultado: **ABCDEFGHJIJ**.

Necessitamos especificar o **passcode.o** porque, dentro do arquivo **atividade2.c** necessitamos importar um arquivo de cabeçalho **passcode.h** que é implementado por esse **arquivo.o**.

3-

Para esta questão, necessitei converter o **prog01.exe** para assembly, para isso usei o comando:

```
objdump -d prog01.exe > prog01.asm
```



```
# matheus @ zup-2744 in ~/sistemas-operacionais/unidade-2/atividade3 [17:07:24]
$ objdump -d prog01.exe > prog01.asm

# matheus @ zup-2744 in ~/sistemas-operacionais/unidade-2/atividade3 [17:07:29]
$ ls
prog01.asm  prog01.exe
```

Após isso, entrei no arquivo **prog01.asm** com um editor de teste e identifiquei o código hexadecimal referente às chamadas **f1** e **f2**.

```

08048228 <main>:
8048228: 8d 4c 24 04      lea    0x4(%esp),%ecx
804822c: 83 e4 f0         and    $0xffffffff0,%esp
804822f: ff 71 fc         pushl  -0x4(%ecx)
8048232: 55              push   %ebp
8048233: 89 e5           mov    %esp,%ebp
8048235: 51              push   %ecx
8048236: 83 ec 04        sub    $0x4,%esp
8048239: e8 0e 00 00 00   call   804824c <f1>
804823e: e8 21 00 00 00   call   8048264 <f2>
8048243: 83 c4 04        add    $0x4,%esp
8048246: 59              pop    %ecx
8048247: 5d              pop    %ebp
8048248: 8d 61 fc        lea    -0x4(%ecx),%esp
804824b: c3             ret

```

Logo em seguida, abri o prog01.exe com o gdb, a partir do comando **`gdb prog01.exe`**.

Dando um disassemble na função main, obtive as instruções básicas em assembly deste programa.

```

# matheus @ zup-2744 in ~/sistemas-operacionais/unidade-2/atividade3 [17:07:38]
$ gdb prog01.exe
GNU gdb (Ubuntu 9.2-0ubuntu1~20.04) 9.2
Copyright (C) 2020 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from prog01.exe...
(gdb) disassemble main
Dump of assembler code for function main:
   0x08048228 <+0>:      lea    0x4(%esp),%ecx
   0x0804822c <+4>:      and    $0xffffffff0,%esp
   0x0804822f <+7>:      pushl  -0x4(%ecx)
   0x08048232 <+10>:     push   %ebp
   0x08048233 <+11>:     mov    %esp,%ebp
   0x08048235 <+13>:     push   %ecx
   0x08048236 <+14>:     sub    $0x4,%esp
   0x08048239 <+17>:     call   0x0804824c <f1>
   0x0804823e <+22>:     call   0x08048264 <f2>
   0x08048243 <+27>:     add    $0x4,%esp
   0x08048246 <+30>:     pop    %ecx
   0x08048247 <+31>:     pop    %ebp
   0x08048248 <+32>:     lea    -0x4(%ecx),%esp
   0x0804824b <+35>:     ret
End of assembler dump.
(gdb) █

```

A partir dessas instruções básicas, deu para perceber, desde o início da linha 0x08048228 a linha 0x0804824b, que há uma diferença de +5 Bytes de uma instrução para outra.

Sabendo disso, para que possamos inverter a ordem das instruções das chamadas, precisamos primeiro somar -5 ao binário da primeira chamada e +5 para a segunda chamada.

```
8048239:    e8 0e 00 00 00    call 804824c <f1>
804823e:    e8 21 00 00 00    call 8048264 <f2>
```

0e - 5 = 09

21 + 5 = 26

Editando o binário do prog01.exe com o hexedit e salvando:

```
00 00 00 C7 44 24 04 C8 29 00
04 E8 26 00 00 00 E8 09 00 00
```

Ao executar o programa, então temos:

```
# matheus @ zup-2744 in ~/sistemas-operacionais/unidade-2/atividade3 [19:08:02]
$ ./prog01.exe
F2
F1
```

O assembly após a modificação:

```
(gdb) disassemble main
Dump of assembler code for function main:
   0x08048228 <+0>:    lea    0x4(%esp),%ecx
   0x0804822c <+4>:    and    $0xffffffff0,%esp
   0x0804822f <+7>:    pushl  -0x4(%ecx)
   0x08048232 <+10>:   push   %ebp
   0x08048233 <+11>:   mov    %esp,%ebp
   0x08048235 <+13>:   push   %ecx
   0x08048236 <+14>:   sub    $0x4,%esp
   0x08048239 <+17>:   call   0x08048264 <f2>
   0x0804823e <+22>:   call   0x0804824c <f1>
   0x08048243 <+27>:   add    $0x4,%esp
   0x08048246 <+30>:   pop    %ecx
   0x08048247 <+31>:   pop    %ebp
   0x08048248 <+32>:   lea    -0x4(%ecx),%esp
   0x0804824b <+35>:   ret
End of assembler dump.
(gdb)
```

4-

Para o prog02.exe, o tempo de execução foi de 0,040s:

```
./prog02 0,01s user 0,03s system 98% cpu 0,040 total
```

Para o prog03.exe, o tempo de execução foi de 0,014s:

```
./prog03 0,00s user 0,01s system 94% cpu 0,014 total
```

Para a segunda parte da tarefa, executando ambos os programas 11 vezes e tirando a média dos valores fora a primeira execução, utilizei os seguintes comandos:

```
for i in {1..11}; do { time ./prog02 | tail -n 0 } 2>> prog02-execution-time.txt ; done
```

Resultado:

```
prog02-execution-time.txt
1 ./prog02 0,00s user 0,00s system 88% cpu 0,004 total$
1 ./prog02 0,00s user 0,00s system 79% cpu 0,004 total$
2 ./prog02 0,00s user 0,00s system 85% cpu 0,004 total$
3 ./prog02 0,00s user 0,00s system 82% cpu 0,004 total$
4 ./prog02 0,00s user 0,00s system 85% cpu 0,004 total$
5 ./prog02 0,00s user 0,00s system 85% cpu 0,003 total$
6 ./prog02 0,00s user 0,00s system 75% cpu 0,003 total$
7 ./prog02 0,00s user 0,00s system 76% cpu 0,003 total$
8 ./prog02 0,00s user 0,00s system 77% cpu 0,002 total$
9 ./prog02 0,00s user 0,00s system 79% cpu 0,002 total$
10 ./prog02 0,00s user 0,00s system 71% cpu 0,002 total$
```

Com média de execução de: 0,031 segundos

```
for i in {1..11}; do { time ./prog03 | tail -n 0 } 2>> prog03-execution-time.txt ; done
```

```
prog03-execution-time.txt
10 ./prog03 0,00s user 0,00s system 83% cpu 0,003 total$
9 ./prog03 0,00s user 0,00s system 85% cpu 0,002 total$
8 ./prog03 0,00s user 0,00s system 67% cpu 0,002 total$
7 ./prog03 0,00s user 0,00s system 37% cpu 0,003 total$
6 ./prog03 0,00s user 0,00s system 81% cpu 0,001 total$
5 ./prog03 0,00s user 0,00s system 70% cpu 0,001 total$
4 ./prog03 0,00s user 0,00s system 82% cpu 0,001 total$
3 ./prog03 0,00s user 0,00s system 82% cpu 0,001 total$
2 ./prog03 0,00s user 0,00s system 75% cpu 0,001 total$
1 ./prog03 0,00s user 0,00s system 69% cpu 0,001 total$
11 ./prog03 0,00s user 0,00s system 77% cpu 0,001 total$
```

Com média de execução de: 0,014 segundos

Dessa forma, conclui-se que o **prog03** é mais rápido que o **prog02**



5-

Para a atividade 5, foi necessário escrever um programa single threaded, um programa com multiprocessos single threaded e um programa multi processos multi threaded.

Segue o tempo de execução para a primeira abordagem:

Comando utilizado para gerar o executável:

**gcc -o count\_files\_singlethread.c count\_files\_singlethreaded**

```
# matheus @ zup-2744 in ~/sistemas-operacionais/unidade-2/atividade5 [18:38:06]
$ sudo time ./count_files_singlethreaded
Enter a directory to be calculated the size of files: /
Number of files: 1661168
0.06user 0.93system 0:02.82elapsed 35%CPU (0avgtext+0avgdata 1968maxresident)k
0inputs+0outputs (0major+341minor)pagefaults 0swaps
```

Foram gastos 35% de CPU e 0.93 segundos de processamento de sistema.

Já para a segunda abordagem foi necessário utilizar o vfork() e gravar o resultado das operações em um arquivo e posteriormente somar. Isso foi feito porque o processo pai não tem acesso aos valores que os filhos calculam. Segue o resultado:

Comando utilizado para gerar o executável:

**gcc -o multiprocess-singlethreaded count\_files\_multiprocess\_singlethread.c**

```
# matheus @ zup-2744 in ~/sistemas-operacionais/unidade-2/atividade5 [18:40:28] C:134
$ sudo time ./multiprocess-singlethreaded
PID=474961: Processo Pai
PID=474962: i=0
Finalizando - PID=474962: i=0
PID=474963: i=1
Finalizando - PID=474963: i=1
PID=474964: i=2
Finalizando - PID=474964: i=2
PID=474965: i=3
Finalizando - PID=474965: i=3
PID=474966: i=4
Finalizando - PID=474966: i=4
PID=474967: i=5
Finalizando - PID=474967: i=5
PID=474968: i=6
Finalizando - PID=474968: i=6
PID=474969: i=7
Finalizando - PID=474969: i=7
O numero total de arquivos da partição / é: 1536525
PID=474968: i=8
Finalizando - PID=474968: i=8
free(): double free detected in tcache 2
Command terminated by signal 6
0.00user 0.00system 0:01.07elapsed 0%CPU (0avgtext+0avgdata 2004maxresident)k
0inputs+0outputs (0major+63minor)pagefaults 0swaps
```

Foram gastos 0% de CPU e 0s de sistema. Esta abordagem já mostra um resultado bem melhor em relação ao single threaded.

Para a abordagem 3, necessitei utilizar o pthread create e o pthread join para realizar as operações, segue o tempo de execução:

Comando utilizado para gerar o executável:

```
gcc -o count_files_multiprocess_multithreaded count_files_multiprocess_multithreaded.c -pthread
```

```
# matheus @ zup-2744 in ~/sistemas-operacionais/unidade-2/atividade5 [18:42:47] C:139
$ sudo time ./count_files_multiprocess_multithreaded
Start to count files from /bin directory: Count directory /bin 2 1889
Command terminated by signal 11
0.00user 0.00system 0:00.12elapsed 3%CPU (0avgtext+0avgdata 1848maxresident)k
0inputs+0outputs (0major+90minor)pagefaults 0swaps
```