

PEDRO HENRIQUE RESENDE RIBEIRO
12011BCC004

ATIVIDADE PRÁTICA 02

MODELO DE PROCESSOS E THREADS



UNIVERSIDADE FEDERAL DE UBERLÂNDIA
FACULDADE DE COMPUTAÇÃO
2021

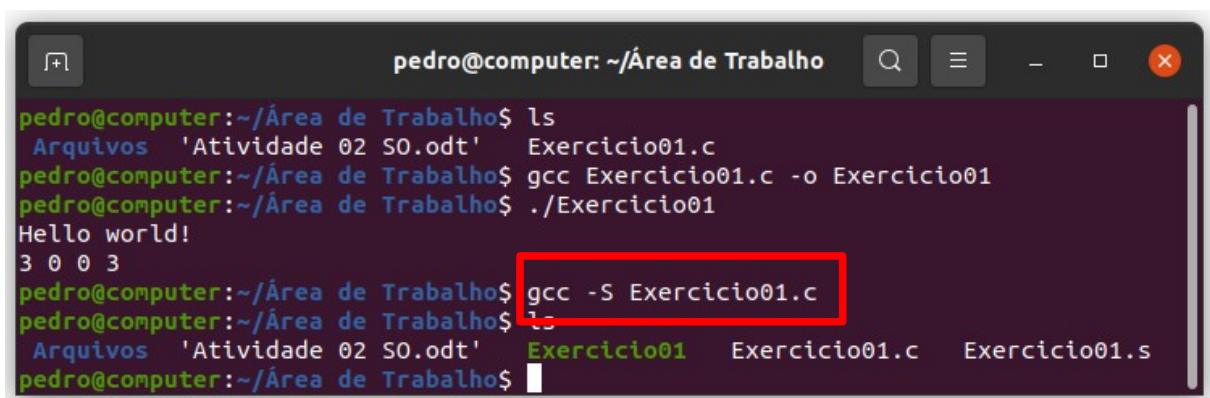
01) Utilizando dois compiladores diferentes de linguagem C, produza o código Assembly para o programa abaixo. Analise as diferenças de cada código Assembly produzido em comparação ao código C. A plataforma de SO é de livre escolha.

```
#include <stdio.h>

int i = 3;
int j;
main( ){
    int w;
    int z = 3;
    printf("Hello world!\n");
    printf("%d%d%d%d", i, j, w, z);
}
```

Solução: Para realizar a tarefa, foi gerado um arquivo chamado “Exercicio01.c” contendo o código fonte do enunciado. Em seguida, para compilar o código fonte em C utilizando o compilador gcc, o comando mostrado abaixo foi digitado no terminal do Ubuntu 20.04.2 LTS:

```
gcc -S Exercicio01.c
```

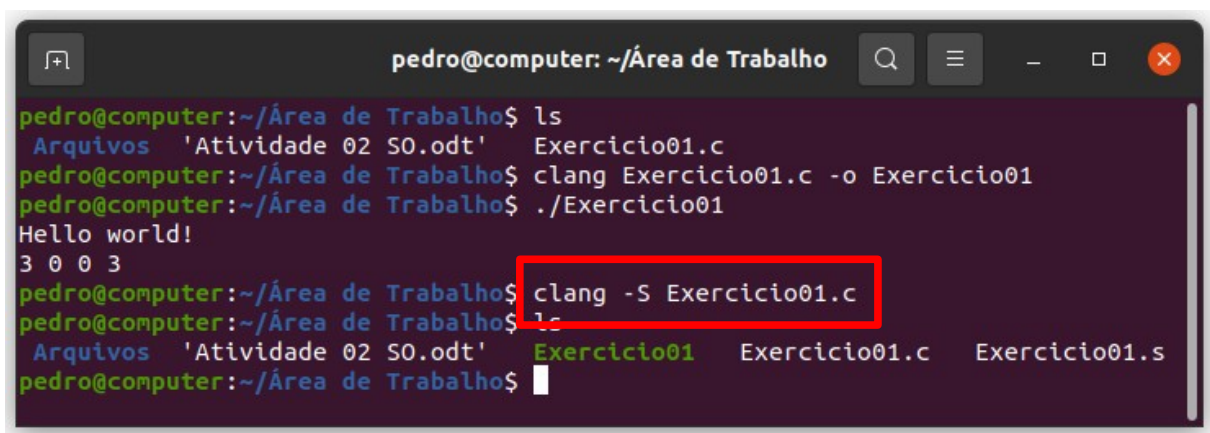


```
pedro@computer: ~/Área de Trabalho
pedro@computer:~/Área de Trabalho$ ls
Arquivos 'Atividade 02 SO.odt'  Exercicio01.c
pedro@computer:~/Área de Trabalho$ gcc Exercicio01.c -o Exercicio01
pedro@computer:~/Área de Trabalho$ ./Exercicio01
Hello world!
3 0 0 3
pedro@computer:~/Área de Trabalho$ gcc -S Exercicio01.c
pedro@computer:~/Área de Trabalho$ ls
Arquivos 'Atividade 02 SO.odt'  Exercicio01  Exercicio01.c  Exercicio01.s
pedro@computer:~/Área de Trabalho$
```

É obtido um warning durante o processo, mas é pelo fato de a função main não contém o int antes de sua declaração. Ao adicionar o tipo int, a mensagem do

warning é eliminada e o processo de compilação ocorre sem nenhum problema. Como resultado, obtém-se o código Assembly do programa.

O outro compilador utilizado foi o Clang (versão 10.0.0-4ubuntu1), também no sistema operacional Ubuntu. É possível observar que os dois compiladores produzem códigos Assembly do mesmo tamanho, porém a forma como cada um gera o código é diferente. O código gerado pelo GCC (Exercicio01_GCC.s) faz a declaração da variável `i` e armazena o espaço de memória para as strings antes da `main`, enquanto que o código gerado pelo Clang (Exercicio01_Clang.s) realiza esta atividade após a `main`.



```
pedro@computer: ~/Área de Trabalho
pedro@computer:~/Área de Trabalho$ ls
Arquivos 'Atividade 02 SO.odt' Exercício01.c
pedro@computer:~/Área de Trabalho$ clang Exercício01.c -o Exercício01
pedro@computer:~/Área de Trabalho$ ./Exercício01
Hello world!
3 0 0 3
pedro@computer:~/Área de Trabalho$ clang -S Exercício01.c
pedro@computer:~/Área de Trabalho$ ls
Arquivos 'Atividade 02 SO.odt' Exercício01 Exercício01.c Exercício01.s
pedro@computer:~/Área de Trabalho$
```

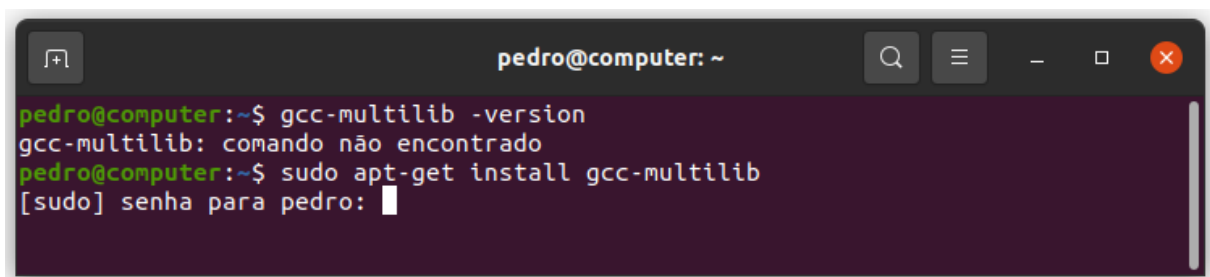
02) A partir do código fonte abaixo, gere o programa executável, execute-o e anote a senha que será exibida na tela. Observe que a rotina `passcode()` não faz parte do programa abaixo e nem da linguagem C. Sua implementação está disponível no arquivo objeto denominado “`passcode.o`” que acompanha esta lista. Para usar essa rotina é necessário incluir o arquivo de cabeçalho “`passcode.h`” que acompanha o “`passcode.o`”. O arquivo “`passcode.o`” foi criado para funcionar apenas no sistema operacional Linux.

```
#include <stdio.h>
#include "passcode.h"
main( ){
    char code[11];
    passcode(code);
    printf("%s", code);
}
```

Solução: Para gerar o arquivo executável, é necessário realizar o processo de linkagem do arquivo “passcode.o”. O sistema operacional utilizado é 64 bits, enquanto que o arquivo citado é de 32 bits. Para convertê-lo em um arquivo 64 bits, é necessário utilizar a biblioteca “gcc-multilib”.

Em primeiro lugar, foi realizada a conferência se a biblioteca citada estava presente na máquina. Como o resultado foi negativo, para instalar a biblioteca, utilizou-se o seguinte comando no terminal:

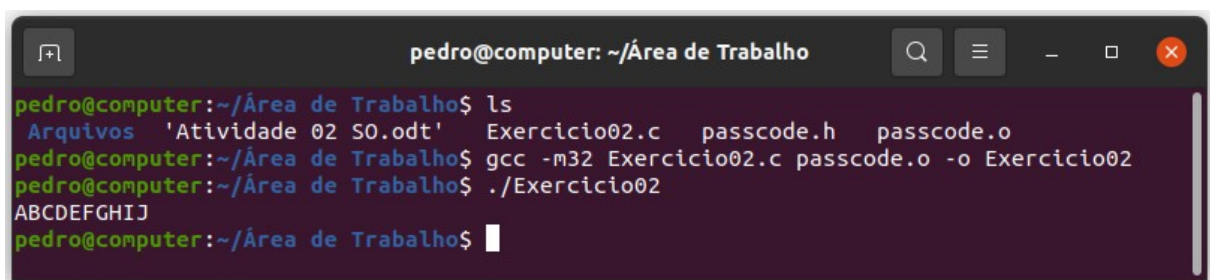
```
sudo apt-get install gcc-multilib
```

A terminal window titled 'pedro@computer: ~' with search, menu, and window control icons. The user enters 'gcc-multilib -version' and receives the message 'gcc-multilib: comando não encontrado'. Then, the user enters 'sudo apt-get install gcc-multilib' and is prompted for a password, indicated by a cursor and a small white box.

```
pedro@computer:~$ gcc-multilib -version
gcc-multilib: comando não encontrado
pedro@computer:~$ sudo apt-get install gcc-multilib
[sudo] senha para pedro: 
```

Após a instalação da biblioteca, para fazer a linkagem do arquivo objeto do código fonte e do “passcode.o”, o seguinte comando foi digitado no terminal:

```
gcc -m32 Exercicio02.c passcode.o -o Exercicio02
```

A terminal window titled 'pedro@computer: ~/Área de Trabalho' with search, menu, and window control icons. The user runs 'ls' and sees a list of files including 'Exercicio02.c', 'passcode.h', and 'passcode.o'. Then, the user runs 'gcc -m32 Exercicio02.c passcode.o -o Exercicio02' and './Exercicio02', which outputs 'ABCDEFGHIJ'.

```
pedro@computer:~/Área de Trabalho$ ls
Arquivos 'Atividade 02 SO.odt'  Exercicio02.c  passcode.h  passcode.o
pedro@computer:~/Área de Trabalho$ gcc -m32 Exercicio02.c passcode.o -o Exercicio02
pedro@computer:~/Área de Trabalho$ ./Exercicio02
ABCDEFGHIJ
pedro@computer:~/Área de Trabalho$ 
```

Após executar o programa, obtém-se o passcode, que é ABCDEFGHIJ.

03) O arquivo de programa “prog01.exe” (no diretório desta lista) possui três funções, main(), f1() e f2(), cujo código fonte em C está listado abaixo. Faça alterações diretamente no arquivo de programa (prog01.exe) para que ao ser executado a função main() chame primeiro f2() e depois f1(). O arquivo de

programa prog01.exe foi criado para funcionar apenas no sistema operacional Linux.

```
#include <stdio.h>

f1( ){ printf("F1"); }

f2( ){ printf("F2"); }

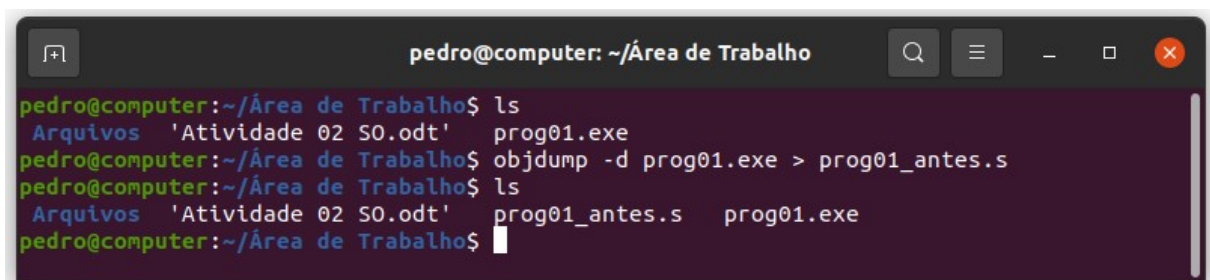
main( ){
    f1( );
    f2( );
}
```

Solução: Para realizar a modificação do programa “prog01.exe”, em primeiro lugar foi necessário obter o código Assembly para visualizar as chamadas das funções e os endereços correspondentes a elas.

Para realizar o “disassembly” do programa “prog01.exe”, é necessário utilizar o comando “objdump”. De acordo com o Man do Linux, ao utilizar “-d” juntamente com o comando “objdump”, obtém-se o código Assembly. Dessa forma, o seguinte comando foi digitado no terminal:

```
objdump -d prog01.exe > prog01_antes.s
```

É importante observar que a saída padrão do comando “objdump” é a tela do terminal. Dessa forma, foi feita uma saída redirecionada para um arquivo de texto denominado “prog01_antes.s”.

A screenshot of a Linux terminal window with a dark background. The title bar at the top reads "pedro@computer: ~/Área de Trabalho". The terminal shows the following commands and output: 1. Command: `ls`. Output: `Arquivos 'Atividade 02 SO.odt' prog01.exe`. 2. Command: `objdump -d prog01.exe > prog01_antes.s`. 3. Command: `ls`. Output: `Arquivos 'Atividade 02 SO.odt' prog01_antes.s prog01.exe`. The cursor is positioned at the end of the last line.

```
pedro@computer: ~/Área de Trabalho
pedro@computer:~/Área de Trabalho$ ls
Arquivos 'Atividade 02 SO.odt' prog01.exe
pedro@computer:~/Área de Trabalho$ objdump -d prog01.exe > prog01_antes.s
pedro@computer:~/Área de Trabalho$ ls
Arquivos 'Atividade 02 SO.odt' prog01_antes.s prog01.exe
pedro@computer:~/Área de Trabalho$
```

Ao analisar o trecho da função main do código Assembly, notou-se que as chamadas das funções eram realizadas nas linhas 127 e 128 do arquivo. A Figura

abaixo ilustra as chamadas das funções e o código hexadecimal correspondente a cada uma delas.

```
119 08048228 <main>:
120 8048228: 8d 4c 24 04      lea     0x4(%esp),%ecx
121 804822c: 83 e4 f0         and     $0xffffffff0,%esp
122 804822f: ff 71 fc         pushl   -0x4(%ecx)
123 8048232: 55              push    %ebp
124 8048233: 89 e5           mov     %esp,%ebp
125 8048235: 51              push    %ecx
126 8048236: 83 ec 04         sub     $0x4,%esp
127 8048239: e8 0e 00 00 00   call    804824c <f1>
128 804823e: e8 21 00 00 00   call    8048264 <f2>
129 8048243: 83 c4 04         add     $0x4,%esp
130 8048246: 59              pop     %ecx
131 8048247: 5d              pop     %ebp
132 8048248: 8d 61 fc         lea     -0x4(%ecx),%esp
133 804824b: c3              ret
```

O código hexadecimal “E8”, de acordo com uma consulta feita no site Stack Overflow e C9X.me, se refere a uma chamada relativa à instrução anterior. Isso quer dizer que o endereço utilizado para chamar cada função está relacionado com a posição de memória da instrução anterior. Ainda, de acordo com o site Stack Overflow, a chamada da instrução é <some address> + 5 bytes.

O segundo par hexadecimal “0E” para a chamada de “f1” e “21” para a chamada de “f2” se refere a quantidade de bytes relativos à instrução anterior. Para alterar o “prog01.exe”, deve-se modificar esses dois valores.

Para realizar a modificação, foi realizado o seguinte procedimento:

- Subtrair o endereço da tag da função 02 com o endereço mostrado na linha 127 (pois a chamada da função f2 deve ocorrer primeiro).
- Subtrair 5 bytes do valor encontrado.
- Subtrair o endereço da tag da função 01 com o endereço mostrado na linha 128 (pois a chamada da função f1 deve ocorrer depois de f2).
- Subtrair 5 bytes do valor encontrado.

Para a função f2 tem-se: $8048264 - 8048239 = 2B$ (lembrar que é uma operação com números hexadecimais). Subtraindo 5 do valor encontrado anteriormente, obtêm-se o valor 26. Dessa forma, na linha 127, deve-se trocar o valor “0E” por “26”.

Para a função f1 tem-se: $804824C - 804823E = 0E$. Subtraindo 5 do valor encontrado obtêm-se o valor 09. Dessa forma, na linha 128, deve-se trocar o valor “21” por “09”. Isso pode ser visto no editor hexadecimal Bless Hex Editor.

prog01.exe	
00000000	7F 45 4C 46 01 01 01 00 00 00 00 00 00 00 02 00 03 00 01 00 00 00 30 81 04 08 34
0000001d	00 00 00 18 9E 08 00 00 00 00 00 34 00 20 00 05 00 28 00 21 00 1E 00 01 00 00 00 00
0000003a	00 00 00 80 04 08 00 80 04 08 FB 91 08 00 FB 91 08 00 05 00 00 00 10 00 00 01 00 00
00000057	00 FC 91 08 00 FC 21 0D 08 FC 21 0D 08 C4 07 00 00 54 23 00 00 06 00 00 00 10 00 00
00000074	04 00 00 00 D4 00 00 00 D4 80 04 08 D4 80 04 08 20 00 00 00 20 00 00 00 04 00 00 04
00000091	00 00 00 07 00 00 00 FC 91 08 00 FC 21 0D 08 FC 21 0D 08 10 00 00 00 28 00 00 00 04 00
000000ae	00 00 04 00 00 00 51 E5 74 64 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000cb	00 06 00 00 00 04 00 00 00 04 00 00 10 00 00 00 01 00 00 00 47 4E 55 00 00 00 00
000000e8	02 00 00 00 06 00 00 12 00 00 00 55 89 E5 53 83 EC 04 E8 00 00 00 00 5B 81 C3 5C A1
00000105	08 00 8B 93 FC FF FF FF 85 D2 74 05 E8 EA 7E FB F7 E8 B5 00 00 00 E8 10 8A 06 00 58 5B
00000122	C9 C3 00 00 00 00 00 00 00 00 00 00 00 31 ED 5E 89 E1 83 E4 F0 50 54 52 68 B0 8B 04
0000013f	08 68 10 8B 04 08 51 56 68 28 82 04 08 E8 2F 01 00 00 F4 90 90 90 90 90 90 90 90 90
0000015c	90 90 90 90 55 89 E5 53 83 EC 14 80 3D C0 29 0D 08 00 75 54 A1 C4 29 0D 08 BB 1C 22 0D
00000179	08 81 EB 14 22 0D 08 C1 FB 02 83 EB 01 39 D8 73 1E 8D B6 00 00 00 00 83 C0 01 A3 C4 29
00000196	0D 08 FF 14 85 14 22 0D 08 A1 C4 29 0D 08 39 D8 72 E8 B8 E0 FF 0A 08 85 C0 74 0C C7 04
000001b3	24 84 B0 0C 08 E8 23 7E 06 00 C6 05 C0 29 0D 08 01 83 C4 14 5B 5D C3 8D B6 00 00 00 00
000001d0	55 B8 A0 FD 0A 08 89 E5 83 EC 18 E8 00 00 00 5A 81 C2 7C A0 08 00 85 C0 74 20 89 54
000001ed	24 0C C7 44 24 08 00 00 00 C7 44 24 04 C8 29 0D 08 C7 04 24 84 B0 0C 08 E8 95 7B 06
0000020a	00 A1 20 22 0D 08 85 C0 74 12 B8 00 00 00 00 85 C0 74 00 07 04 24 20 22 0D 08 FF 50 C9
00000227	C3 8D 4C 24 04 83 E4 F0 FF 71 FC 55 89 E5 51 83 EC 00 E8 0E 00 00 00 E8 21 00 00 00 83
00000244	C4 04 59 5D 8D 61 FC C3 55 89 E5 83 EC 08 83 EC 0C 6 10 2A 0B 08 E8 19 0F 00 00 83 C4 10 C9 C3 90 90
00000261	10 C9 C3 55 89 E5 83 EC 08 83 EC 0C 68 4B 2A 0B 08 E8 19 0F 00 00 83 C4 10 C9 C3 90 90
0000027e	90 90 55 B8 00 00 00 89 E5 57 56 53 83 EC 4C 85 C0 8B 7D 14 8B 75 18 8B 5D 1C 0F 84

prog01.exe	
00000000	7F 45 4C 46 01 01 01 00 00 00 00 00 00 00 02 00 03 00 01 00 00 00 30 81 04 08 34
0000001d	00 00 00 18 9E 08 00 00 00 00 00 34 00 20 00 05 00 28 00 21 00 1E 00 01 00 00 00 00
0000003a	00 00 00 80 04 08 00 80 04 08 FB 91 08 00 FB 91 08 00 05 00 00 00 10 00 00 01 00 00
00000057	00 FC 91 08 00 FC 21 0D 08 FC 21 0D 08 C4 07 00 00 54 23 00 00 06 00 00 00 10 00 00
00000074	04 00 00 00 D4 00 00 00 D4 80 04 08 D4 80 04 08 20 00 00 00 20 00 00 00 04 00 00 04
00000091	00 00 00 07 00 00 00 FC 91 08 00 FC 21 0D 08 FC 21 0D 08 10 00 00 00 28 00 00 00 04 00
000000ae	00 00 04 00 00 00 51 E5 74 64 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000cb	00 06 00 00 00 04 00 00 00 04 00 00 10 00 00 00 01 00 00 00 47 4E 55 00 00 00 00
000000e8	02 00 00 00 06 00 00 12 00 00 00 55 89 E5 53 83 EC 04 E8 00 00 00 00 5B 81 C3 5C A1
00000105	08 00 8B 93 FC FF FF FF 85 D2 74 05 E8 EA 7E FB F7 E8 B5 00 00 00 E8 10 8A 06 00 58 5B
00000122	C9 C3 00 00 00 00 00 00 00 00 00 00 00 31 ED 5E 89 E1 83 E4 F0 50 54 52 68 B0 8B 04
0000013f	08 68 10 8B 04 08 51 56 68 28 82 04 08 E8 2F 01 00 00 F4 90 90 90 90 90 90 90 90 90
0000015c	90 90 90 90 55 89 E5 53 83 EC 14 80 3D C0 29 0D 08 00 75 54 A1 C4 29 0D 08 BB 1C 22 0D
00000179	08 81 EB 14 22 0D 08 C1 FB 02 83 EB 01 39 D8 73 1E 8D B6 00 00 00 00 83 C0 01 A3 C4 29
00000196	0D 08 FF 14 85 14 22 0D 08 A1 C4 29 0D 08 39 D8 72 E8 B8 E0 FF 0A 08 85 C0 74 0C C7 04
000001b3	24 84 B0 0C 08 E8 23 7E 06 00 C6 05 C0 29 0D 08 01 83 C4 14 5B 5D C3 8D B6 00 00 00 00
000001d0	55 B8 A0 FD 0A 08 89 E5 83 EC 18 E8 00 00 00 5A 81 C2 7C A0 08 00 85 C0 74 20 89 54
000001ed	24 0C C7 44 24 08 00 00 00 C7 44 24 04 C8 29 0D 08 C7 04 24 84 B0 0C 08 E8 95 7B 06
0000020a	00 A1 20 22 0D 08 85 C0 74 12 B8 00 00 00 00 85 C0 74 00 07 04 24 20 22 0D 08 FF 50 C9
00000227	C3 8D 4C 24 04 83 E4 F0 FF 71 FC 55 89 E5 51 83 EC 00 E8 26 00 00 00 E8 09 00 00 00 83
00000244	C4 04 59 5D 8D 61 FC C3 55 89 E5 83 EC 08 83 EC 0C 6 10 2A 0B 08 E8 19 0F 00 00 83 C4 10 C9 C3 90 90
00000261	10 C9 C3 55 89 E5 83 EC 08 83 EC 0C 68 4B 2A 0B 08 E8 19 0F 00 00 83 C4 10 C9 C3 90 90
0000027e	90 90 55 B8 00 00 00 89 E5 57 56 53 83 EC 4C 85 C0 8B 7D 14 8B 75 18 8B 5D 1C 0F 84

Para realizar o teste do programa “prog01.exe”, em primeiro lugar é necessário digitar o seguinte comando no terminal para permitir a execução.

```
chmod +x prog01.exe
```

Em seguida, ao executar o programa, a saída mostra que as chamadas das funções foram invertidas.

```
pedro@computer: ~/Área de Trabalho
pedro@computer:~/Área de Trabalho$ chmod +x prog01.exe
pedro@computer:~/Área de Trabalho$ ls
Arquivos 'Atividade 02 SO.odt'  prog01_antes.s  prog01.exe
pedro@computer:~/Área de Trabalho$ ./prog01.exe
F2
F1
pedro@computer:~/Área de Trabalho$
```

Para confirmar que tudo ocorreu corretamente, foi gerado o arquivo “prog01_depois.s” para conferir as linhas 127 e 128 do arquivo. Como pode ser notado, a inversão das chamadas, de fato foi realizada.

```
119 08048228 <main>:
120 8048228: 8d 4c 24 04      lea    0x4(%esp),%ecx
121 804822c: 83 e4 f0         and    $0xffffffff0,%esp
122 804822f: ff 71 fc        pushl  -0x4(%ecx)
123 8048232: 55              push   %ebp
124 8048233: 89 e5           mov    %esp,%ebp
125 8048235: 51              push   %ecx
126 8048236: 83 ec 04        sub    $0x4,%esp
127 8048239: e8 26 00 00 00   call   8048264 <f2>
128 804823e: e8 09 00 00 00   call   804824c <f1>
129 8048243: 83 c4 04        add    $0x4,%esp
130 8048246: 59              pop     %ecx
131 8048247: 5d              pop     %ebp
132 8048248: 8d 61 fc        lea    -0x4(%ecx),%esp
133 804824b: c3              ret
```

04) A partir do código fonte abaixo, crie os arquivos de programas “prog02.exe” e “prog03.exe”. Execute cada programa comparando seus tempos de execução. Para isso, utilize o comando “time” (Linux) ou “PowerShell Measure-Command” (Windows). Abaixo exemplos de utilização.

```
/* prog02.c */
#include <stdio.h>
main( ){
    int i;
    for(i = 0; i < 10000; i++)
        printf("A\n");
}
```



```

/* prog03.c */
#include <stdio.h>
#include <string.h>
main( ){
    int i;
    char str[20001] = "";
    for(i = 0; i < 10000; i += 2){
        *(str + i) = 'A';
        *(str + i + 1) = '\\n';
    }
    *(str + i) = '\\0';
    printf("%s", str);
}

```

C:\Users\johndoe>powershell -command "Measure-Command" {prog02.exe}"

C:\Users\johndoe>powershell -command "Measure-Command" {prog03.exe}"

Obs: Execute 11 vezes cada programa, descarte a primeira execução de cada programa e tire a média dos demais 10 resultados de cada programa para ter um valor aproximado dos tempos de execução de cada um.

Solução: Após compilar os programas, para utilizar o comando time, deve-se digitar o comando da seguinte forma no terminal:

time ./prog02 ou time ./prog03

A tabela a seguir ilustra o tempo das 11 execuções em milissegundos para o programa "prog02".

Tipo	1	2	3	4	5	6	7	8	9	10	11
Real	0,033	0,032	0,031	0,031	0,030	0,029	0,030	0,028	0,029	0,032	0,034
User	0,012	0,008	0,012	0,004	0,004	0,000	0,013	0,005	0,004	0,000	0,012
Sys	0,020	0,024	0,019	0,027	0,026	0,029	0,018	0,023	0,025	0,032	0,021

A primeira medida de tempo da execução do “prog02” será descartada e a média será calculada da medida 2 até a 11. Dessa forma, a média e o desvio padrão dos tempos em milissegundos é:

	Real [ms]	User [ms]	Sys [ms]
Média	0,031	0,006	0,024
Desvio Padrão	0,002	0,005	0,004

A tabela a seguir ilustra o tempo das 11 execuções em milissegundos para o programa “prog03”.

Tipo	1	2	3	4	5	6	7	8	9	10	11
Real	0,008	0,008	0,007	0,007	0,008	0,007	0,007	0,006	0,007	0,007	0,007
User	0,004	0,004	0,000	0,000	0,000	0,003	0,000	0,000	0,000	0,000	0,000
Sys	0,004	0,004	0,007	0,007	0,008	0,004	0,008	0,007	0,008	0,007	0,007

A primeira medida de tempo da execução do “prog03” será descartada e a média será calculada da medida 2 até a 11. Dessa forma, a média e o desvio padrão dos tempos em milissegundos é:

	Real [ms]	User [ms]	Sys [ms]
Média	0,0071	0,0007	0,0067
Desvio Padrão	0,0006	0,0015	0,0015

Como discutido na plataforma MS Teams, o comando time é uma forma rudimentar de medição do tempo de execução e não é recomendada para medidas que exigem precisão. De maneira geral, percebe-se que o programa “prog03” possui uma eficiência maior, visto que o tempo de execução real é cerca de 3 vezes menor do que o tempo gasto pelo “prog02”.

05) Fazer um programa em linguagem C para contar e imprimir o número total de arquivos armazenados em um disco rígido. Implementar e comparar o tempo de execução de três versões desse programa. A primeira versão deve ser programada como um único processo singlethreaded. A segunda versão deve

ser programada com múltiplos processos singlethreaded, onde o número de processos (n) deve corresponder ao número de processadores do computador. Caso o computador tenha apenas um processador, então utilize $n = 2$. A terceira versão deve ser programada como múltiplos processos, tal como a segunda versão, contudo cada processo deve utilizar múltiplas threads (mt). O valor de mt deve ser 2. Na segunda e terceira versões, o algoritmo de busca e contagem de arquivos deve ser paralelizado; por exemplo, enquanto um processo conta os arquivos em uma parte do disco (ex: C:\ no Windows ou /dev/sda1 no Linux) o outro processo conta os arquivos em outra parte (ex: D:\ ou /dev/sda2). O mesmo aplica-se para múltiplas threads. A estratégia de paralelização do algoritmo de contagem de arquivos é de livre escolha, assim como a plataforma de SO escolhida para realizar esse exercício.