Лабораторна робота №1

Аналіз програмного коду мов високого рівня

Виконав:
Студент 3 курсу ФТІ
групи ФІ-92
Поночевний Назар Юрійович
Варіант 6

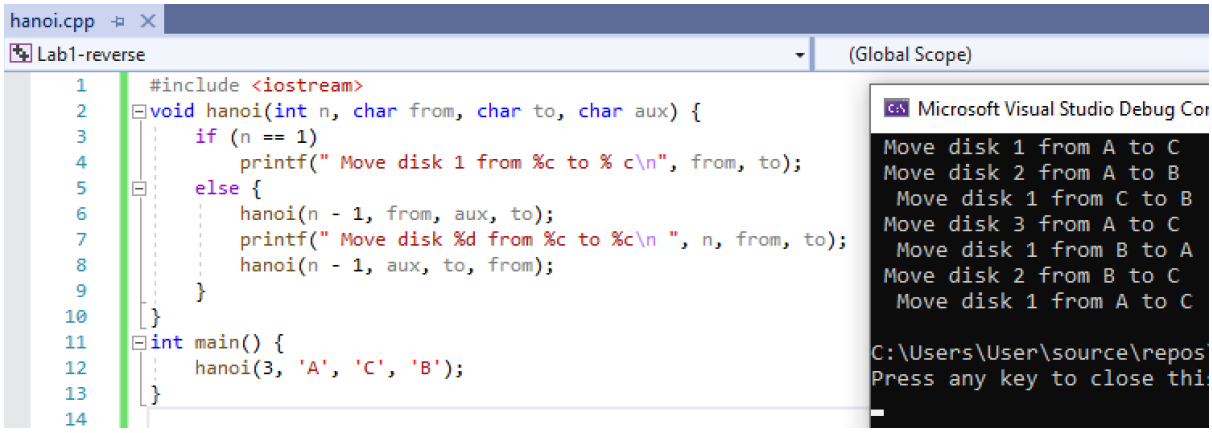
Мета роботи

Отримати навички розпізнавання констукцій мов високого рівня в машинному коді для архітектур x86/x64 та ARM/ARM64 на прикладі C/C++.

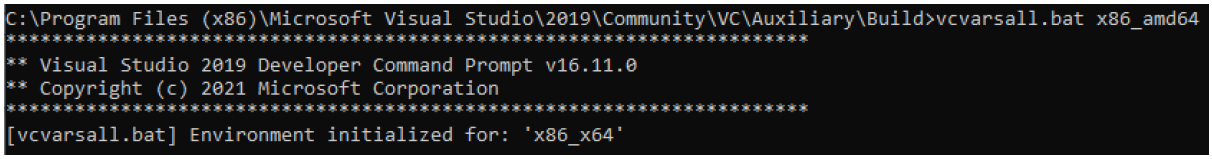
Завдання 1:

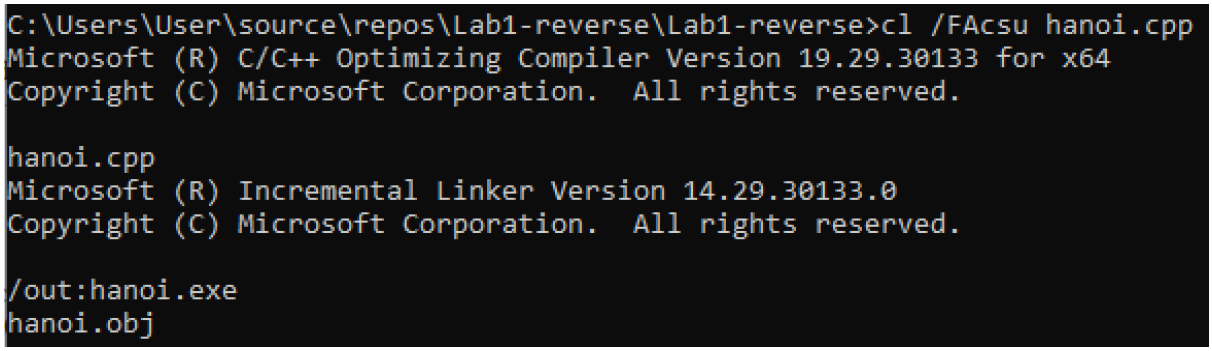
Проаналізувати машинний код прикладу hanoi.c для Windows x64, ARM, ARM64 (MSVC), для Linux amd64, arm, arm64 (GCC), для Linux amd64 (LLVM clang, https://llvm.org/);

1) Спочатку перевіримо реалізацію hanoi.cpp



2) Переводимо MSVC у режим x64 і компілюємо





3) Переводимо MSVC у режим ARM і компілюємо

```
C:\Program Files (x86)\Microsoft Visual Studio\2019\Community\VC\Auxiliary\Build>vcvarsall.bat x86_arm
**********************************************************************
** Visual Studio 2019 Developer Command Prompt v16.11.0
** Copyright (c) 2021 Microsoft Corporation
**********************************************************************
[vcvarsall.bat] Environment initialized for: 'x86_arm'

C:\Program Files (x86)\Microsoft Visual Studio\2019\Community\VC\Auxiliary\Build>
```

```
C:\Users\User\source\repos\Lab1-reverse\Lab1-reverse>cl /FAcsu hanoi.cpp
Microsoft (R) C/C++ Optimizing Compiler Version 19.29.30133 for x86
Copyright (C) Microsoft Corporation.  All rights reserved.

hanoi.cpp
Microsoft (R) Incremental Linker Version 14.29.30133.0
Copyright (C) Microsoft Corporation.  All rights reserved.

/out:hanoi.exe
hanoi.obj

C:\Users\User\source\repos\Lab1-reverse\Lab1-reverse>dir
 Volume in drive C has no label.
 Volume Serial Number is E411-13D6

 Directory of C:\Users\User\source\repos\Lab1-reverse\Lab1-reverse

10/13/2021  09:42 AM    <DIR>          .
10/13/2021  09:42 AM    <DIR>          ..
10/13/2021  09:21 AM    <DIR>          Debug
10/13/2021  09:42 AM             7,458 hanoi.cod
10/13/2021  09:20 AM               321 hanoi.cpp
10/13/2021  09:42 AM           102,400 hanoi.exe
10/13/2021  09:42 AM             2,063 hanoi.obj
10/13/2021  09:20 AM             7,195 Lab1-reverse.vcxproj
10/13/2021  09:20 AM               978 Lab1-reverse.vcxproj.filters
10/13/2021  09:18 AM               168 Lab1-reverse.vcxproj.user
               7 File(s)        120,583 bytes
               3 Dir(s)  78,482,485,248 bytes free
```

4) Переводимо MSVC у режим ARM64 і компілюємо

```
C:\Program Files (x86)\Microsoft Visual Studio\2019\Community\VC\Auxiliary\Build>vcvarsall.bat x86_arm64
**********************************************************************
** Visual Studio 2019 Developer Command Prompt v16.11.0
** Copyright (c) 2021 Microsoft Corporation
**********************************************************************
[vcvarsall.bat] Environment initialized for: 'x86_arm64'

C:\Program Files (x86)\Microsoft Visual Studio\2019\Community\VC\Auxiliary\Build>
```

```
C:\Users\User\source\repos\Lab1-reverse\Lab1-reverse>cl /FAcsu hanoi.cpp
Microsoft (R) C/C++ Optimizing Compiler Version 19.29.30133 for x86
Copyright (C) Microsoft Corporation.  All rights reserved.

hanoi.cpp
Microsoft (R) Incremental Linker Version 14.29.30133.0
Copyright (C) Microsoft Corporation.  All rights reserved.

/out:hanoi.exe
hanoi.obj
hanoi.obj : error LNK2019: unresolved external symbol ___acrt_iob_func referenced in function _printf
hanoi.obj : error LNK2019: unresolved external symbol ___stdio_common_vfprintf referenced in function __vfprintf_l
LINK : error LNK2001: unresolved external symbol _mainCRTStartup
C:\Program Files (x86)\Microsoft Visual Studio\2019\Community\VC\Tools\MSVC\14.29.30133\lib\x64\libcpmt.lib : warning LN
K4272: library machine type 'x64' conflicts with target machine type 'x86'
C:\Program Files (x86)\Microsoft Visual Studio\2019\Community\VC\Tools\MSVC\14.29.30133\lib\x64\LIBCMT.lib : warning LNK
4272: library machine type 'x64' conflicts with target machine type 'x86'
hanoi.exe : fatal error LNK1120: 3 unresolved externals

C:\Users\User\source\repos\Lab1-reverse\Lab1-reverse>
```

5) Всі коди додав у архів. Тепер спробуємо зробити те саме для Linux amd64 (gcc)

```
┌──(kali㉿kali)-[~/lab1-reverse]
└─$ gcc -g hanoi.c -o hanoi.amd64

┌──(kali㉿kali)-[~/lab1-reverse]
└─$ objdump  -drwC -Mintel  -S hanoi.amd64 > hanoi.amd64.2.lst
```

6) Тепер Linux ARM (gcc)

```
┌──(kali㉿kali)-[~/lab1-reverse]
└─$ arm-linux-gnueabi-gcc -g hanoi.c -o hanoi.arm

┌──(kali㉿kali)-[~/lab1-reverse]
└─$ arm-linux-gnueabi-objdump  -drwC  -S hanoi.arm > hanoi_gcc_arm.lst
```

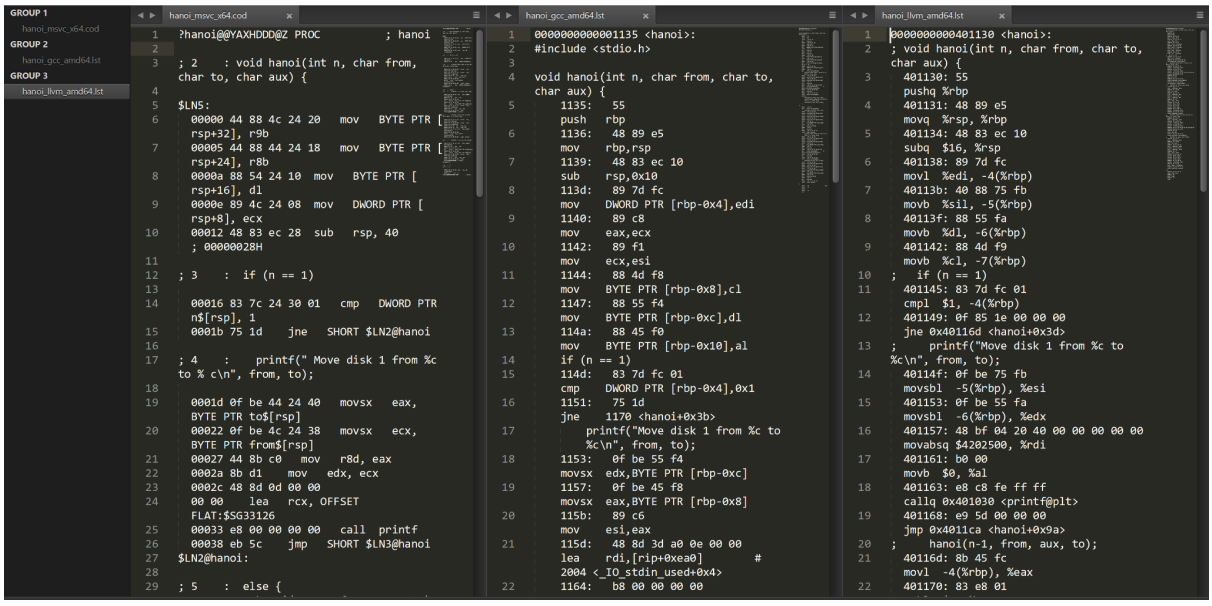
7) I Linux arm64 (gcc)

```
┌──(kali㉿kali)-[~/lab1-reverse]
└─$ aarch64-linux-gnu-gcc -g hanoi.c -o hanoi.aarch64

┌──(kali㉿kali)-[~/lab1-reverse]
└─$ aarch64-linux-gnu-objdump  -drwC  -S hanoi.aarch64 > hanoi_gcc_aarch64.lst
```

8) Файли зберігаємо у архів і спробуємо для Linux amd64 (llvm clang)

```
┌──(kali㉿kali)-[~/lab1-reverse]
└─$ clang -g hanoi.c -o hanoi.amd64

┌──(kali㉿kali)-[~/lab1-reverse]
└─$ llvm-objdump -drwC -S hanoi.amd64 > hanoi_llvm_amd64.lst
llvm-objdump: warning: 'hanoi.amd64': failed to parse debug information for hanoi.amd64
```

9) Тепер час порівняти як відрізняються машинний код у MSVC, GCC та LLVM



Завдання 2:

Реалізувати мовою C/C++, проаналізувати результати компіляції (за варіантом), для платформ i686, amd64, arm, aarch64:

– Комбінаторні алгоритми [25], будь-який на Ваш вибір з вказаного класу:

6. на графах – розфарбування графів;

– Криптографічні алгоритми, алгоритми кодування та контролю цілісності [26, 27]:

6. ARCfour / RC4;

10) Реалізуємо розфарбовування графу жадібним алгоритмом

```
┌──(kali㉿kali)-[~/lab1-reverse]
└─$ cat graph-coloring.cpp
// A C++ program to implement greedy algorithm for graph coloring
#include <iostream>
#include <list>
using namespace std;

// A class that represents an undirected graph
class Graph
{
        int V; // No. of vertices
        list<int> *adj; // A dynamic array of adjacency lists
public:
        // Constructor and destructor
        Graph(int V) { this->V = V; adj = new list<int>[V]; }
        ~Graph()         { delete [] adj; }

        // function to add an edge to graph
        void addEdge(int v, int w);

        // Prints greedy coloring of the vertices
        void greedyColoring();
};

void Graph::addEdge(int v, int w)
{
        adj[v].push_back(w);
        adj[w].push_back(v); // Note: the graph is undirected
}

// Assigns colors (starting from 0) to all vertices and prints
// the assignment of colors
void Graph::greedyColoring()
{
        int result[V];

        // Assign the first color to first vertex
        result[0] = 0;

        // Initialize remaining V-1 vertices as unassigned
        for (int u = 1; u < V; u++)
                result[u] = -1; // no color is assigned to u

        // A temporary array to store the available colors. True
        // value of available[cr] would mean that the color cr is
        // assigned to one of its adjacent vertices
        bool available[V];
        for (int cr = 0; cr < V; cr++)
                available[cr] = false;

        // Assign colors to remaining V-1 vertices
```

```
        // Assign colors to remaining V-1 vertices
        for (int u = 1; u < V; u++)
        {
                // Process all adjacent vertices and flag their colors
                // as unavailable
                list<int>::iterator i;
                for (i = adj[u].begin(); i != adj[u].end(); ++i)
                        if (result[*i] != -1)
                                available[result[*i]] = true;

                // Find the first available color
                int cr;
                for (cr = 0; cr < V; cr++)
                        if (available[cr] == false)
                                break;

                result[u] = cr; // Assign the found color

                // Reset the values back to false for the next iteration
                for (i = adj[u].begin(); i != adj[u].end(); ++i)
                        if (result[*i] != -1)
                                available[result[*i]] = false;
        }

        // print the result
        for (int u = 0; u < V; u++)
                cout << "Vertex " << u << " ---> Color "
                        << result[u] << endl;
}

// Driver program to test above function
int main()
{
        Graph g1(5);
        g1.addEdge(0, 1);
        g1.addEdge(0, 2);
        g1.addEdge(1, 2);
        g1.addEdge(1, 3);
        g1.addEdge(2, 3);
        g1.addEdge(3, 4);
        cout << "Coloring of graph 1 \n";
        g1.greedyColoring();

        Graph g2(5);
        g2.addEdge(0, 1);
        g2.addEdge(0, 2);
        g2.addEdge(1, 2);
        g2.addEdge(1, 4);
        g2.addEdge(2, 4);
        g2.addEdge(4, 3);
        cout << "\nColoring of graph 2 \n";
        g2.greedyColoring();
```

```
┌──(kali㉿kali)-[~/lab1-reverse]
└─$ g++ -g graph-coloring.cpp -o graph-coloring.amd64

┌──(kali㉿kali)-[~/lab1-reverse]
└─$ ./graph-coloring.amd64
Coloring of graph 1
Vertex 0 ⟶ Color 0
Vertex 1 ⟶ Color 1
Vertex 2 ⟶ Color 2
Vertex 3 ⟶ Color 0
Vertex 4 ⟶ Color 1

Coloring of graph 2
Vertex 0 ⟶ Color 0
Vertex 1 ⟶ Color 1
Vertex 2 ⟶ Color 2
Vertex 3 ⟶ Color 0
Vertex 4 ⟶ Color 3
```

11) Компілюємо для Linux i686 (gcc)

```
┌──(kali㉿kali)-[~/lab1-reverse]
└─$ g++ -m32 -g graph-coloring.cpp -o graph-coloring.i686 -march=i686

┌──(kali㉿kali)-[~/lab1-reverse]
└─$ i686-linux-gnu-objdump -drwC -Mintel -S graph-coloring.i686 > graph-coloring_gcc_i686.lst
```

12) Компілюємо для Linux amd64 (gcc)

```
┌──(kali㉿kali)-[~/lab1-reverse]
└─$ g++ -g graph-coloring.cpp -o graph-coloring.amd64

┌──(kali㉿kali)-[~/lab1-reverse]
└─$ objdump -drwC -Mintel -S graph-coloring.amd64 > graph-coloring_gcc_amd64.lst
```

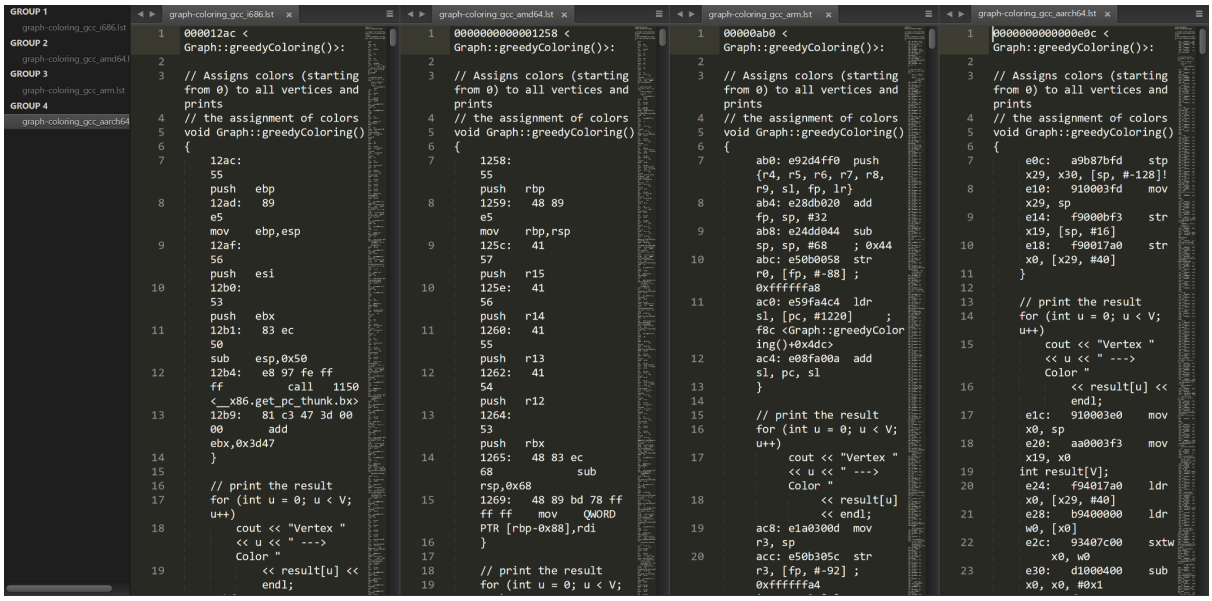
13) Компілюємо для Linux arm (gcc)

```
┌──(kali㉿kali)-[~/lab1-reverse]
└─$ arm-linux-gnueabi-g++ -g graph-coloring.cpp -o graph-coloring.arm

┌──(kali㉿kali)-[~/lab1-reverse]
└─$ arm-linux-gnueabi-objdump -drwC -S graph-coloring.arm > graph-coloring_gcc_arm.lst
```

14) Компілюємо для Linux aarch64 (gcc)

```
┌──(kali㉿kali)-[~/lab1-reverse]
└─$ aarch64-linux-gnu-g++ -g graph-coloring.cpp -o graph-coloring.aarch64

┌──(kali㉿kali)-[~/lab1-reverse]
└─$ aarch64-linux-gnu-objdump -drwC -S graph-coloring.aarch64 > graph-coloring_gcc_aarch64.lst
```

15) Всі файли кладемо в архів і порівнюємо



16) Тепер так само з RC4. Спочатку реалізуємо алгоритм

```
└─$ cat rc4.cpp
#include <iostream>
#include <fstream>
#include <cstdlib>
#include <cstdio>
#include <cstring>

class State
{
        unsigned char s[256];
        int i, j;

        void swap(int a, int b);

        public:
        unsigned char getbyte(void);
        State(unsigned char key[], int length );
};

State::State(unsigned char key[], int length)
{
        for(int k=0; k<256; k++)
        {
                s[k]=k;
        }

        j=0;

        for(i=0; i<256 ; i++)
        {
                j=(j + s[i] + key[i % length]) % 256;
                swap(i, j);
        }

        i=j=0;
}

void State::swap(int a, int b)
{
        unsigned char temp= s[i];
        s[i]=s[j];
        s[j]=temp;
}

unsigned char State::getbyte(void)
{
        i=(i+1)%256;
        j=(j+s[i])%256;
        swap(i, j);
        int index=(s[i]+s[j])%256;
        return s[index];
}
```

17) Компілюємо для Linux i686 (gcc)

```
┌──(kali㉿kali)-[~/lab1-reverse]
└─$ g++ -m32 -g rc4.cpp -o rc4.i686 -march=i686

┌──(kali㉿kali)-[~/lab1-reverse]
└─$ i686-linux-gnu-objdump -drwC -Mintel -S rc4.i686 > rc4_gcc_i686.lst
```

18) Компілюємо для Linux amd64 (gcc)

```
┌──(kali㉿kali)-[~/lab1-reverse]
└─$ g++ -g rc4.cpp -o rc4.amd64

┌──(kali㉿kali)-[~/lab1-reverse]
└─$ objdump -drwC -Mintel -S rc4.amd64 > rc4_gcc_amd64.lst
```

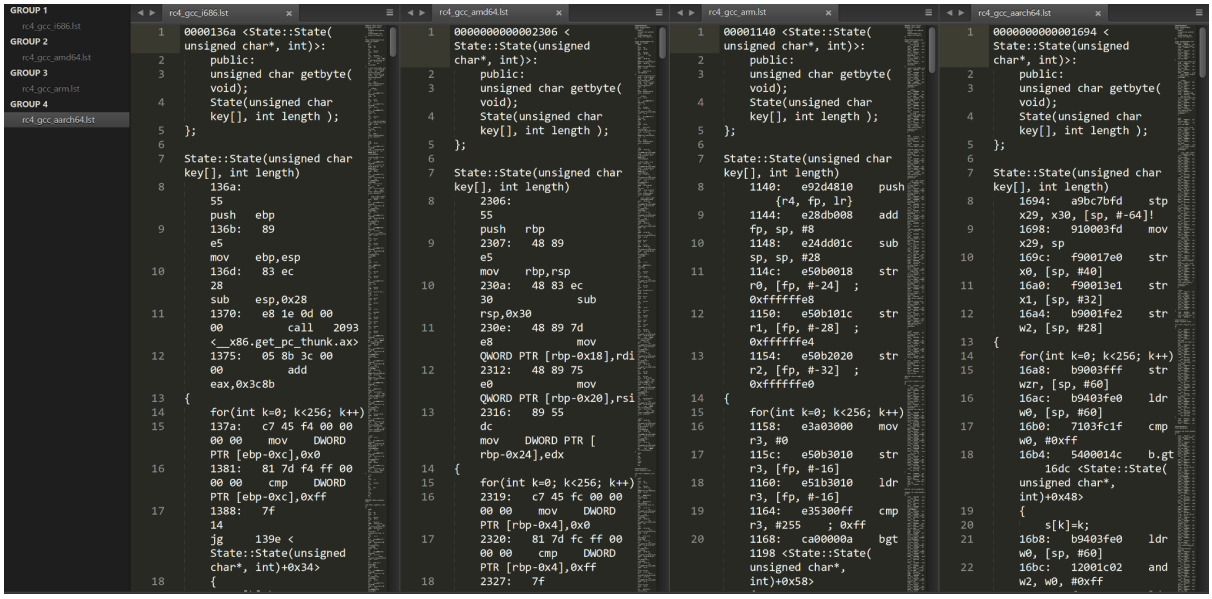
19) Компілюємо для Linux arm (gcc)

```
┌──(kali㉿kali)-[~/lab1-reverse]
└─$ arm-linux-gnueabi-g++ -g rc4.cpp -o rc4.arm

┌──(kali㉿kali)-[~/lab1-reverse]
└─$ arm-linux-gnueabi-objdump -drwC -S rc4.arm > rc4_gcc_arm.lst
```

20) Компілюємо для Linux aarch64 (gcc)

```
┌──(kali㉿kali)-[~/lab1-reverse]
└─$ aarch64-linux-gnu-g++ -g rc4.cpp -o rc4.aarch64

┌──(kali㉿kali)-[~/lab1-reverse]
└─$ aarch64-linux-gnu-objdump -drwC -S rc4.aarch64 > rc4_gcc_aarch64.lst
```

21) Всі файли кладемо в архів і порівнюємо



Завдання 3:

Реалізація функцій стандартної бібліотеки С. Бібліотека за варіантами, функції всі зазначені (за наявності реалізації), версія бібліотеки остання стабільна на момент початку курсу:

6. klibc [33];

Функції:

– стандартного ввод-виводу printf, puts;
– роботи з файлами fopen, fread, fwrite, feof, fclose;
– виконання команд операційної системи system.

Зверніть увагу на відмінності системних викликів у різних ОС Linux та Windows для різних архітектур (x86, x86_64, ARM) [35].

22) Візьмемо реалізацію функції printf у бібліотеці klibc

```
┌──(kali㉿kali)-[~/lab1-reverse]
└─$ cat vsnprintf.c
/*
 * vsnprintf.c
 *
 * vsnprintf(), from which the rest of the printf()
 * family is built
 */

#include <stdarg.h>
#include <stddef.h>
#include <inttypes.h>
#include <string.h>
#include <limits.h>
#include <stdio.h>

enum flags {
        FL_ZERO         = 0x01, /* Zero modifier */
        FL_MINUS        = 0x02, /* Minus modifier */
        FL_PLUS         = 0x04, /* Plus modifier */
        FL_TICK         = 0x08, /* ' modifier */
        FL_SPACE        = 0x10, /* Space modifier */
        FL_HASH         = 0x20, /* # modifier */
        FL_SIGNED       = 0x40, /* Number is signed */
        FL_UPPER        = 0x80  /* Upper case digits */
};

/* These may have to be adjusted on certain implementations */
enum ranks {
        rank_char       = -2,
        rank_short      = -1,
        rank_int        =  0,
        rank_long       =  1,
        rank_longlong   =  2
};

#define MIN_RANK        rank_char
#define MAX_RANK        rank_longlong

#define INTMAX_RANK     rank_longlong
#define SIZE_T_RANK     rank_long
#define PTRDIFF_T_RANK  rank_long

#define EMIT(x) ({ if (o<n){*q++ = (x);} o++; })

static size_t
format_int(char *q, size_t n, uintmax_t val, enum flags flags,
           int base, int width, int prec)
{
        char *qq;
        size_t o = 0, oo;
```

23) Компілюємо для Linux i686 (gcc)

```
┌──(kali㉿kali)-[~/lab1-reverse]
└─$ i686-linux-gnu-gcc vsnprintf.c -c -o vsnprintf.i686

┌──(kali㉿kali)-[~/lab1-reverse]
└─$ i686-linux-gnu-objdump -drwC -Mintel -S vsnprintf.i686 > vsnprintf_gcc_i686.lst
```

24) Компілюємо для Linux amd64 (gcc)

```
┌──(kali㉿kali)-[~/lab1-reverse]
└─$ gcc vsnprintf.c -c -o vsnprintf.amd64

┌──(kali㉿kali)-[~/lab1-reverse]
└─$ objdump -drwC -Mintel -S vsnprintf.amd64 > vsnprintf_gcc_amd64.lst
```

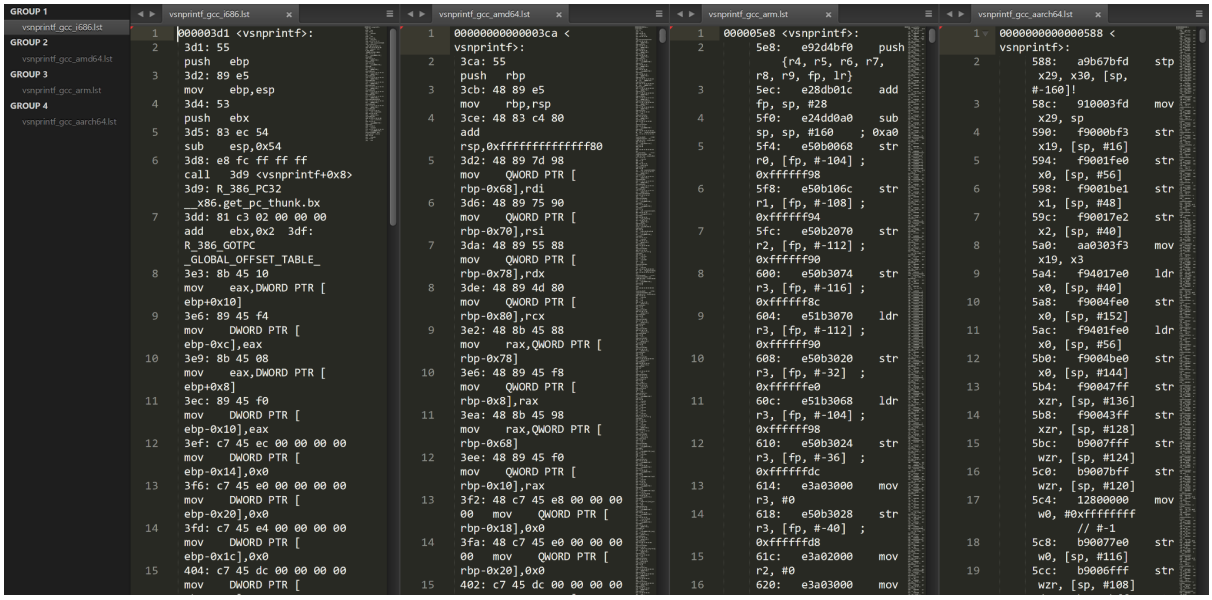
25) Компілюємо для Linux arm (gcc)

```
┌──(kali㉿kali)-[~/lab1-reverse]
└─$ arm-linux-gnueabi-gcc vsnprintf.c -c -o vsnprintf.arm

┌──(kali㉿kali)-[~/lab1-reverse]
└─$ arm-linux-gnueabi-objdump -drwC -S vsnprintf.arm > vsnprintf_gcc_arm.lst
```

26) Компілюємо для Linux aarch64 (gcc)

```
┌──(kali㉿kali)-[~/lab1-reverse]
└─$ aarch64-linux-gnu-gcc vsnprintf.c -c -o vsnprintf.aarch64

┌──(kali㉿kali)-[~/lab1-reverse]
└─$ aarch64-linux-gnu-objdump -drwC -S vsnprintf.aarch64 > vsnprintf_gcc_aarch64.lst
```

27) Всі файли кладемо в архів і порівнюємо



28) Візьмемо реалізацію функції fopen у бібліотеці klibc

```
┌──(kali㉿kali)-[~/lab1-reverse]
└─$ cat fxopen.c
/*
 * fxopen.c
 *
 * Common code between fopen(), fdopen() and the standard descriptors.
 */

#include "stdioint.h"

FILE *stdin, *stdout, *stderr;

/* This depends on O_RDONLY == 0, O_WRONLY == 1, O_RDWR == 2 */
int __parse_open_mode(const char *mode)
{
        int plus = 0;
        int flags = O_RDONLY;

        while (*mode) {
                switch (*mode++) {
                case 'r':
                        flags = O_RDONLY;
                        break;
                case 'w':
                        flags = O_WRONLY | O_CREAT | O_TRUNC;
                        break;
                case 'a':
                        flags = O_WRONLY | O_CREAT | O_APPEND;
                        break;
                case '+':
                        plus = 1;
                        break;
                }
        }

        if (plus)
                flags = (flags & ~(O_RDONLY | O_WRONLY)) | O_RDWR;

        return flags;
}

FILE *__fxopen(int fd, int flags, int close_on_err)
{
        FILE *f = NULL;

        f = malloc(sizeof *f);
        if (!f)
                goto err;
        /*
        f->fd = fd;
        f->filepos = lseek(fd, 0, SEEK_CUR);
```

29) Компілюємо для Linux i686 (gcc)

```
┌──(kali㉿kali)-[~/lab1-reverse]
└─$ i686-linux-gnu-gcc fxopen.c -c -o fxopen.i686

┌──(kali㉿kali)-[~/lab1-reverse]
└─$ i686-linux-gnu-objdump -drwC -Mintel -S fxopen.i686 > fxopen_gcc_i686.lst
```

30) Компілюємо для Linux amd64 (gcc)

```
┌──(kali㉿kali)-[~/lab1-reverse]
└─$ gcc fxopen.c -c -o fxopen.amd64

┌──(kali㉿kali)-[~/lab1-reverse]
└─$ objdump -drwC -Mintel -S fxopen.amd64 > fxopen_gcc_amd64.lst
```

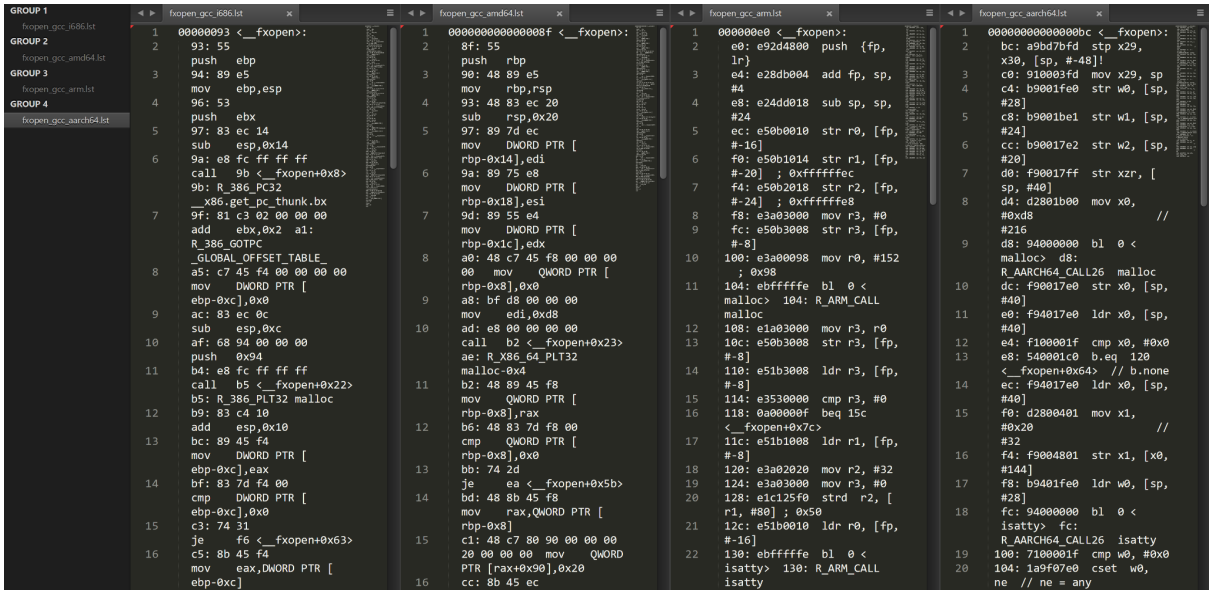
31) Компілюємо для Linux arm (gcc)

```
┌──(kali㉿kali)-[~/lab1-reverse]
└─$ arm-linux-gnueabi-gcc fxopen.c -c -o fxopen.arm

┌──(kali㉿kali)-[~/lab1-reverse]
└─$ arm-linux-gnueabi-objdump -drwC -S fxopen.arm > fxopen_gcc_arm.lst
```

32) Компілюємо для Linux aarch64 (gcc)

```
┌──(kali㉿kali)-[~/lab1-reverse]
└─$ aarch64-linux-gnu-gcc fxopen.c -c -o fxopen.aarch64

┌──(kali㉿kali)-[~/lab1-reverse]
└─$ aarch64-linux-gnu-objdump -drwC -S fxopen.aarch64 > fxopen_gcc_aarch64.lst
```

33) Всі файли кладемо в архів і порівнюємо



```
GROUP 1
  fxopen_gcc_i686.lst
GROUP 2
  fxopen_gcc_amd64.lst
GROUP 3
  fxopen_gcc_arm.lst
GROUP 4
  fxopen_gcc_aarch64.lst
```

fxopen_gcc_i686.lst

```
00000093 <__fxopen>:
 93: 55                   push   ebp
 94: 89 e5                mov    ebp,esp
 96: 53                   push   ebx
 97: 83 ec 14             sub    esp,0x14
 9a: e8 fc ff ff ff       call   9b <__fxopen+0x8>
                          9b: R_386_PC32
                          __x86.get_pc_thunk.bx
 9f: 81 c3 02 00 00 00    add    ebx,0x2  a1:
                          R_386_GOTPC
                          _GLOBAL_OFFSET_TABLE_
 a5: c7 45 f4 00 00 00 00 mov    DWORD PTR [
                          ebp-0xc],0x0
 ac: 83 ec 0c             sub    esp,0xc
 af: 68 94 00 00 00       push   0x94
 b4: e8 fc ff ff ff       call   b5 <__fxopen+0x22>
                          b5: R_386_PLT32 malloc
 b9: 83 c4 10             add    esp,0x10
 bc: 89 45 f4             mov    DWORD PTR [
                          ebp-0xc],eax
 bf: 83 7d f4 00          cmp    DWORD PTR [
                          ebp-0xc],0x0
 c3: 74 31                je     f6 <__fxopen+0x63>
 c5: 8b 45 f4             mov    eax,DWORD PTR [
                          ebp-0xc]
```

fxopen_gcc_amd64.lst

```
000000000000008f <__fxopen>:
  8f: 55                   push   rbp
  90: 48 89 e5             mov    rbp,rsp
  93: 48 83 ec 20          sub    rsp,0x20
  97: 89 7d ec             mov    DWORD PTR [
                           rbp-0x14],edi
  9a: 89 75 e8             mov    DWORD PTR [
                           rbp-0x18],esi
  9d: 89 55 e4             mov    DWORD PTR [
                           rbp-0x1c],edx
  a0: 48 c7 45 f8 00 00 00
  00   mov    QWORD PTR [
                           rbp-0x8],0x0
  a8: bf d8 00 00 00       mov    edi,0xd8
  ad: e8 00 00 00 00       call   b2 <__fxopen+0x23>
                           ae: R_X86_64_PLT32
                           malloc-0x4
  b2: 48 89 45 f8          mov    QWORD PTR [
                           rbp-0x8],rax
  b6: 48 83 7d f8 00       cmp    QWORD PTR [
                           rbp-0x8],0x0
  bb: 74 2d                je     ea <__fxopen+0x5b>
  bd: 48 8b 45 f8          mov    rax,QWORD PTR [
                           rbp-0x8]
  c1: 48 c7 80 90 00 00 00
  20 00 00 00  mov    QWORD
                           PTR [rax+0x90],0x20
  cc: 8b 45 ec
```

fxopen_gcc_arm.lst

```
000000e0 <__fxopen>:
  e0: e92d4800  push  {fp,
  lr}
  e4: e28db004  add fp, sp,
  #4
  e8: e24dd018  sub sp, sp,
  #24
  ec: e50b0010  str r0, [fp,
  #-16]
  f0: e50b1014  str r1, [fp,
  #-20]  ; 0xffffffec
  f4: e50b2018  str r2, [fp,
  #-24]  ; 0xffffffe8
  f8: e3a03000  mov r3, #0
  fc: e50b3008  str r3, [fp,
  #-8]
 100: e3a00098  mov r0, #152
  ; 0x98
 104: ebfffffe  bl  0 <
 malloc>  104: R_ARM_CALL
 malloc
 108: e1a03000  mov r3, r0
 10c: e50b3008  str r3, [fp,
 #-8]
 110: e51b3008  ldr r3, [fp,
 #-8]
 114: e3530000  cmp r3, #0
 118: 0a00000f  beq 15c
 <__fxopen+0x7c>
 11c: e51b1008  ldr r1, [fp,
 #-8]
 120: e3a02020  mov r2, #32
 124: e3a03000  mov r3, #0
 128: e1c125f0  strd  r2, [
 r1, #80] ; 0x50
 12c: e51b0010  ldr r0, [fp,
 #-16]
 130: ebfffffe  bl  0 <
 isatty>  130: R_ARM_CALL
 isatty
```

fxopen_gcc_aarch64.lst

```
00000000000000bc <__fxopen>:
  bc: a9bd7bfd  stp x29,
  x30, [sp, #-48]!
  c0: 910003fd  mov x29, sp
  c4: b9001fe0  str w0, [sp,
  #28]
  c8: b9001be1  str w1, [sp,
  #24]
  cc: b90017e2  str w2, [sp,
  #20]
  d0: f90017ff  str xzr, [
  sp, #40]
  d4: d2801b00  mov x0,
  #0xd8                  //
  #216
  d8: 94000000  bl  0 <
  malloc>  d8:
  R_AARCH64_CALL26  malloc
  dc: f90017e0  str x0, [sp,
  #40]
  e0: f94017e0  ldr x0, [sp,
  #40]
  e4: f100001f  cmp x0, #0x0
  e8: 54000140  b.eq  120
  <__fxopen+0x64>  // b.none
  ec: f94017e0  ldr x0, [sp,
  #40]
  f0: d2800401  mov x1,
  #0x20                  //
  #32
  f4: f9004801  str x1, [x0,
  #144]
  f8: b9401fe0  ldr w0, [sp,
  #28]
  fc: 94000000  bl  0 <
  isatty>  fc:
  R_AARCH64_CALL26  isatty
 100: 7100001f  cmp w0, #0x0
 104: 1a9f07e0  cset  w0,
 ne  // ne = any
```

34) Візьмемо реалізацію функції виконання команд ОС system у бібліотеці klibc

```
┌──(kali㉿kali)-[~/lab1-reverse]
└─$ cat system.c
/*
 * system.c
 *
 * The system() function.  If this turns out to actually be *used*,
 * we may want to try to detect the very simple cases (no shell magic)
 * and handle them internally, instead of requiring that /bin/sh be
 * present.
 */

#include <stdlib.h>
#include <unistd.h>
#include <signal.h>
#include <sys/wait.h>

int system(const char *string)
{
        pid_t pid;
        struct sigaction ignore, old_int, old_quit;
        sigset_t masked, oldmask;
        static const char *argv[] = { "/bin/sh", "-c", NULL, NULL };
        int status;
        extern char** environ;

        /* Block SIGCHLD and ignore SIGINT and SIGQUIT */
        /* Do this before the fork() to avoid races */

        ignore.sa_handler = SIG_IGN;
        sigemptyset(&ignore.sa_mask);
        ignore.sa_flags = 0;
        sigaction(SIGINT, &ignore, &old_int);
        sigaction(SIGQUIT, &ignore, &old_quit);

        sigemptyset(&masked);
        sigaddset(&masked, SIGCHLD);
        sigprocmask(SIG_BLOCK, &masked, &oldmask);

        pid = fork();

        if (pid < 0)
                return -1;
        else if (pid == 0) {
                sigaction(SIGINT, &old_int, NULL);
                sigaction(SIGQUIT, &old_quit, NULL);
                sigprocmask(SIG_SETMASK, &oldmask, NULL);

                argv[2] = string;

                execve(argv[0], (char *const *)argv, (char *const *)environ);
                _exit(127);
```

35) Компілюємо для Linux i686 (gcc)

```
┌──(kali㉿kali)-[~/lab1-reverse]
└─$ i686-linux-gnu-gcc system.c -c -o system.i686

┌──(kali㉿kali)-[~/lab1-reverse]
└─$ i686-linux-gnu-objdump -drwC -Mintel -S system.i686 > system_gcc_i686.lst
```

36) Компілюємо для Linux amd64 (gcc)

```
┌──(kali㉿kali)-[~/lab1-reverse]
└─$ gcc system.c -c -o system.amd64

┌──(kali㉿kali)-[~/lab1-reverse]
└─$ objdump -drwC -Mintel -S system.amd64 > system_gcc_amd64.lst
```

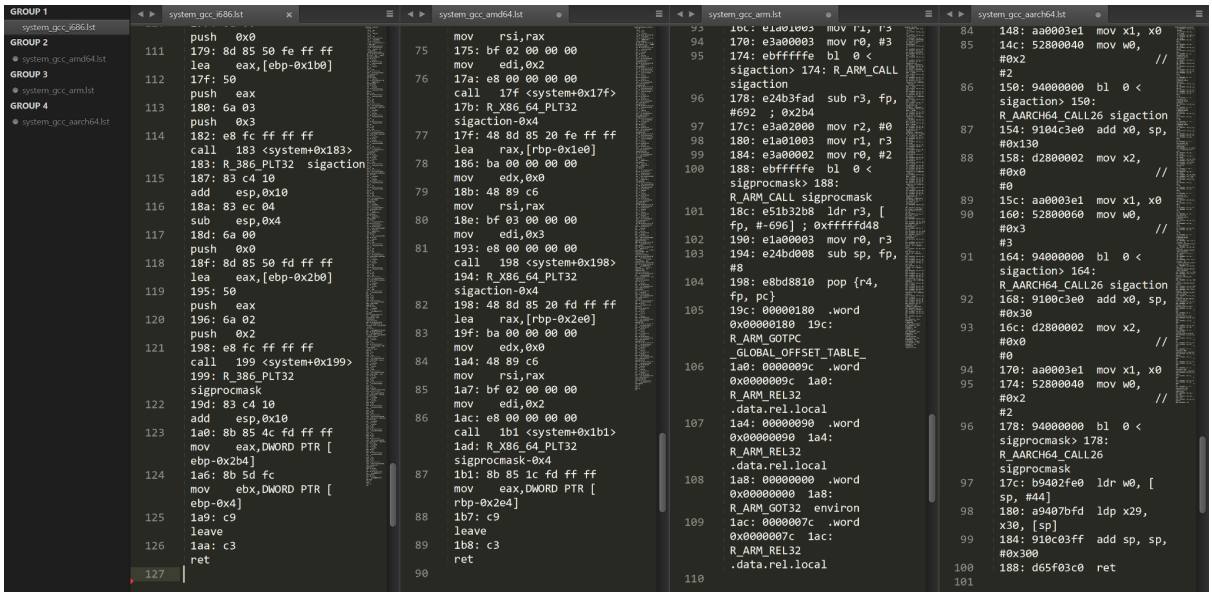
37) Компілюємо для Linux arm (gcc)

```
┌──(kali㉿kali)-[~/lab1-reverse]
└─$ arm-linux-gnueabi-gcc system.c -c -o system.arm

┌──(kali㉿kali)-[~/lab1-reverse]
└─$ arm-linux-gnueabi-objdump -drwC -S system.arm > system_gcc_arm.lst
```

38) Компілюємо для Linux aarch64 (gcc)

```
┌──(kali㉿kali)-[~/lab1-reverse]
└─$ aarch64-linux-gnu-gcc system.c -c -o system.aarch64

┌──(kali㉿kali)-[~/lab1-reverse]
└─$ aarch64-linux-gnu-objdump -drwC -S system.aarch64 > system_gcc_aarch64.lst
```

39) Всі файли кладемо в архів і порівнюємо


GROUP 1
system_gcc_i686.lst
GROUP 2
system_gcc_amd64.lst
GROUP 3
system_gcc_arm.lst
GROUP 4
system_gcc_aarch64.lst
system_gcc_i686.lst
push 0x0
179: 8d 85 50 fe ff ff
lea eax,[ebp-0x1b0]
17f: 50
push eax
180: 6a 03
push 0x3
182: e8 fc ff ff ff
call 183 <system+0x183>
183: R_386_PLT32 sigaction
187: 83 c4 10
add esp,0x10
18a: 83 ec 04
sub esp,0x4
18d: 6a 00
push 0x0
18f: 8d 85 50 fd ff ff
lea eax,[ebp-0x2b0]
195: 50
push eax
196: 6a 02
push 0x2
198: e8 fc ff ff ff
call 199 <system+0x199>
199: R_386_PLT32
sigprocmask
19d: 83 c4 10
add esp,0x10
1a0: 8b 85 4c fd ff ff
mov eax,DWORD PTR [
ebp-0x2b4]
1a6: 8b 5d fc
mov ebx,DWORD PTR [
ebp-0x4]
1a9: c9
leave
1aa: c3
ret
system_gcc_amd64.lst
mov rsi,rax
175: bf 02 00 00 00
mov edi,0x2
17a: e8 00 00 00 00
call 17f <system+0x17f>
17b: R_X86_64_PLT32
sigaction-0x4
17f: 48 8d 85 20 fe ff ff
lea rax,[rbp-0x1e0]
186: ba 00 00 00 00
mov edx,0x0
18b: 48 89 c6
mov rsi,rax
18e: bf 03 00 00 00
mov edi,0x3
193: e8 00 00 00 00
call 198 <system+0x198>
194: R_X86_64_PLT32
sigaction-0x4
198: 48 8d 85 20 fd ff ff
lea rax,[rbp-0x2e0]
19f: ba 00 00 00 00
mov edx,0x0
1a4: 48 89 c6
mov rsi,rax
1a7: bf 02 00 00 00
mov edi,0x2
1ac: e8 00 00 00 00
call 1b1 <system+0x1b1>
1ad: R_X86_64_PLT32
sigprocmask-0x4
1b1: 8b 85 1c fd ff ff
mov eax,DWORD PTR [
rbp-0x2e4]
1b7: c9
leave
1b8: c3
ret
system_gcc_arm.lst
16c: e1a01003 mov r1, r3
170: e3a00003 mov r0, #3
174: ebffffe bl 0 <
sigaction> 174: R_ARM_CALL
sigaction
178: e24b3fad sub r3, fp,
#692 ; 0x2b4
17c: e3a02000 mov r2, #0
180: e1a01003 mov r1, r3
184: e3a00002 mov r0, #2
188: ebffffe bl 0 <
sigprocmask> 188:
R_ARM_CALL sigprocmask
18c: e51b32b8 ldr r3, [
fp, #-696] ; 0xfffffd48
190: e1a00003 mov r0, r3
194: e24bd008 sub sp, fp,
#8
198: e8bd8810 pop {r4,
fp, pc}
19c: 00000180 .word
0x00000180 19c:
R_ARM_GOTPC
_GLOBAL_OFFSET_TABLE_
1a0: 0000009c .word
0x0000009c 1a0:
R_ARM_REL32
.data.rel.local
1a4: 00000090 .word
0x00000090 1a4:
R_ARM_REL32
.data.rel.local
1a8: 00000000 .word
0x00000000 1a8:
R_ARM_GOT32 environ
1ac: 0000007c .word
0x0000007c 1ac:
R_ARM_REL32
.data.rel.local
system_gcc_aarch64.lst
148: aa0003e1 mov x1, x0
14c: 52800040 mov w0,
#0x2 //
#2
150: 94000000 bl 0 <
sigaction> 150:
R_AARCH64_CALL26 sigaction
154: 9104c3e0 add x0, sp,
#0x130
158: d2800002 mov x2,
#0x0 //
#0
15c: aa0003e1 mov x1, x0
160: 52800060 mov w0,
#0x3 //
#3
164: 94000000 bl 0 <
sigaction> 164:
R_AARCH64_CALL26 sigaction
168: 9100c3e0 add x0, sp,
#0x30
16c: d2800002 mov x2,
#0x0 //
#0
170: aa0003e1 mov x1, x0
174: 52800040 mov w0,
#0x2 //
#2
178: 94000000 bl 0 <
sigprocmask> 178:
R_AARCH64_CALL26
sigprocmask
17c: b9402fe0 ldr w0, [
sp, #44]
180: a9407bfd ldp x29,
x30, [sp]
184: 910c03ff add sp, sp,
#0x300
188: d65f03c0 ret