

南開大學

## 汇编语言与逆向技术课程实验报告

### 实验一：HelloWorld



学 院 网络空间安全学院  
专 业 信息安全  
学 号 2213924  
姓 名 申宗尚  
班 级 信安班

## 一、实验目的

- 1、熟悉 Win32 汇编 MASM32 的编译环境;
- 2、命令行输出 “HelloWorld”
- 3、窗口输出 “HelloWorld”

## 二、实验原理

### 1、MASM32

MASM32 是国外的 MASM 爱好者自行整理和编写的一个软件包，最高版本为 11.0 版，MASM32 并不是微软官方发布的软件，微软官方发布的软件 MASM 最新版本也只到 6.15 版，微软发布的 MASM 系列版本从 6.11 版才开始支持 windows 编程，6.11 版以前的版本都不支持 windows 编程，只能用来写 DOS 程序。

MASM32 汇编编译器是 MASM6.0 以上版本中的 ml.exe，资源编译器是 Microsoft Visual Studio 中的 rc.exe，32 位链接器是 Microsoft Visual Studio 中的 Link.exe，同时包含有其他的一些如 lib.exe 和 DumpPe.exe 等工具。

MASM 的 windows 编程的教学书籍有《windows 环境下 32 位汇编语言程序设计第二版》。

### 2、实验代码及解析如下：

#### A) hello\_console.asm

```
.386
.model flat, stdcall
option casemap :none
include \masm32\include\windows.inc
include \masm32\include\kernel32.inc
include \masm32\include\masm32.inc
includelib \masm32\lib\kernel32.lib
includelib \masm32\lib\masm32.lib

.data
    str_hello BYTE "Hello World!", 0

.code
start:
    invoke StdOut, addr str_hello
```

```

        invoke ExitProcess, 0
END start
(1).386
指明指令集。
(2).model flat, stdcall
定义了程序的内存模型和调用规则。model flat 表示使用平坦内存模型,stdcall
表示使用标准调用规则。
(3)option casemap :none
这行代码设置不区分大小写。
(4)include \masm32\include\windows.inc
include \masm32\include\kernel32.inc
include \masm32\include\masm32.inc
includelib \masm32\lib\kernel32.lib
includelib \masm32\lib\masm32.lib
这五行代码用于包含所需的头文件和链接库。其中, windows.inc 包含了
Windows API 的定义, kernel32.inc 包含了 Windows 内核 API 的定义,
masm32.inc 包含了 MASM32 库的定义。includelib 语句告诉编译器链接相应
的库文件。
(5).data str_hello BYTE "Hello World!", 0
在.data 部分,程序定义了一个字符串变量 str_hello,内容为 "Hello World!",
并在末尾加上了一个空字符(字符串结束标志)。
(6).code start:
在.code 部分,程序的主函数开始,起始标签是 start。
(7)invoke StdOut, addr str_hello
这行代码调用了函数 StdOut,该函数的作用是将指定的字符串输出到标准
输出(通常是控制台)。invoke 是 MASM 提供的宏,用于简化函数调用。addr
str_hello 获取 str_hello 字符串的内存地址,并作为参数传递给 StdOut 函数。
(8)invoke ExitProcess, 0
这行代码调用了 ExitProcess 函数,该函数用于退出当前进程。传递的参数是
退出码,通常 0 表示正常退出。
(9)END start
END 语句标志着程序的结束。start 是程序的入口点,也就是程序开始执行的地
方。

```

```

B) hello_window.asm
.386
.model flat, stdcall
option casemap :none

```

```

include \masm32\include\windows.inc
include \masm32\include\kernel32.inc
include \masm32\include\user32.inc
includelib \masm32\lib\kernel32.lib
includelib \masm32\lib\user32.lib
.data
    str_hello BYTE "Hello World!", 0
.code

start:
    invoke MessageBox, NULL, addr str_hello, addr str_hello, MB_OK
    invoke ExitProcess, 0
END start

```

#### (1).386

指定汇编器生成的代码为 80386(或以上)处理器的指令。

#### (2).model flat, stdcall

定义内存模型为平坦模型(flat)，调用规则为 stdcall，即参数从右向左入栈，被调用者负责清理堆栈。

#### (3)option casemap :none

设置标识符不区分大小写。

```

(4)include \masm32\include\windows.inc
include \masm32\include\kernel32.inc
include \masm32\include\user32.inc

```

包含 Windows API 的头文件，windows.inc 包含了基本的 Windows API 定义，kernel32.inc 包含了 Windows 内核 API 的定义，user32.inc 包含了用户界面 API 的定义。

```

(5)includelib \masm32\lib\kernel32.lib
includelib \masm32\lib\user32.lib

```

使用链接库 kernel32.lib 和 user32.lib，这些库包含了对应的 API 函数的实现。

```

(6).data str_hello BYTE "Hello World!", 0

```

在 .data 部分，定义了一个以 NULL 结尾的字符串，内容为 "Hello World!"。

**(7).code start**

在 .code 部分，程序的主函数开始，起始标签是 start。

**(8)invoke MessageBox, NULL, addr str\_hello, addr str\_hello, MB\_OK**

调用 MessageBox 函数，该函数用于显示一个消息框。第一个参数是消息框的父窗口句柄(这里为 NULL，表示没有父窗口)，第二个和第三个参数是消息框中显示的文本(都指向 str\_hello)，第四个参数指定消息框的样式(这里为 MB\_OK，表示消息框只有一个“确定”按钮)。

**(9)invoke ExitProcess, 0**

调用 ExitProcess 函数，该函数用于终止当前进程。传递的参数是退出码(这里为 0，表示正常退出)。

**(10)END start**

END 语句标志着程序的结束。start 是程序的入口点，也就是程序开始执行的地方。

3、实验命令行代码及解析如下：(以 hello\_console.asm 文件命令为例)

**"\masm32\bin\ml /c /Zd /coff  
C:\Users\Administrator\Desktop\hello\_console.asm"**

(1)"\masm32\bin\ml"代表打开盘中 masm32 中 bin 文件夹的 ml 应用程序。

(2)/c 代表仅进行编译，不自动进行链接。

(3)/coff 代表产生的 obj.文件格式为 COFF 格式。

(4)/Zd 代表行号调试信息。

(5)C:\Users\Administrator\Desktop\hello\_console.asm 操作对象文件地址。

**"\masm32\bin\link /SUBSYSTEM:CONSOLE  
C:\Users\Administrator\hello\_console.obj"**

(1) "\masm32\bin\link" 代表打开盘中 masm32 中 bin 文件夹的 link 应用程序

(2)SUBSYSTEM:CONSOLE:具体设置哪个程序入口点由连接器的 "/subsystem:" 选项参数确定，它告诉操作系统如何运行编译生成的.exe 文件。可以指定四种方式："CONSOLE|WINDOWS|NATIVE|POSIX"

(3) C:\Users\Administrator\hello\_console.obj 操作对象文件名

### 三、实验过程

(A) 命令行输出 “HelloWorld”

1. 编辑：用编辑软件 (Notepad) 形成源程序 (.asm), 如: hello\_console.asm。

将编辑好的代码放到 masm32 软件的界面当中，而后将代码保存。



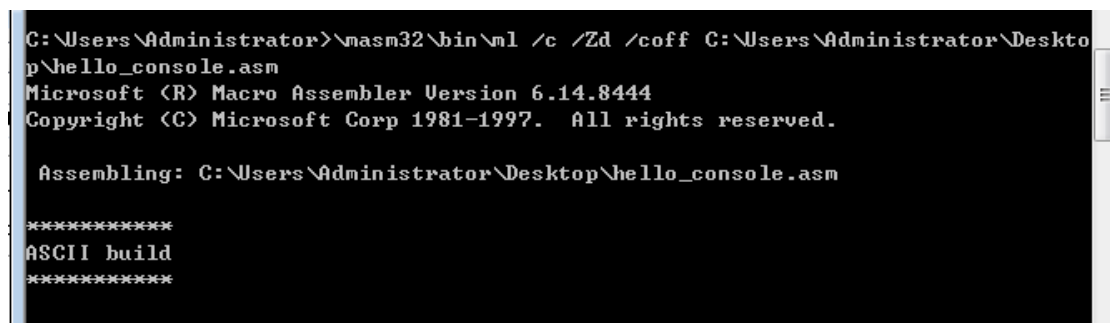
```
hello_console.asm - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

.386
.model flat, stdcall
option casemap :none
include \masm32\include\windows.inc
include \masm32\include\kernel32.inc
include \masm32\include\masm32.inc
includelib \masm32\lib\kernel32.lib
includelib \masm32\lib\masm32.lib

.data
    str_hello BYTE "Hello World!!",0

.code
start:
    invoke StdOut, addr str_hello
    invoke ExitProcess,0
END start
```

2. 编译：用汇编程序 (\masm32\bin\ml.exe) 对源程序进行汇编，形成目标文件 (.obj)，格式如下：“\masm32\bin\ml /c /Zd /coff hello\_console.asm”。



```
C:\Users\Administrator>\masm32\bin\ml /c /Zd /coff C:\Users\Administrator\Desktop\hello_console.asm
Microsoft (R) Macro Assembler Version 6.14.8444
Copyright (C) Microsoft Corp 1981-1997. All rights reserved.

Assembling: C:\Users\Administrator\Desktop\hello_console.asm

*****
ASCII build
*****
```

3. 连接：用连接程序 (\masm32\bin\link.exe) 对目标程序进行连接，形成可执行文件 (.exe)，格式如下：“\masm32\bin\Link /SUBSYSTEM:CONSOLE hello\_console.obj”。

```
C:\Users\Administrator>\masm32\bin\Link /SUBSYSTEM:CONSOLE C:\Users\Administrator\hello_console.obj
Microsoft (R) Incremental Linker Version 5.12.8078
Copyright (C) Microsoft Corp 1992-1998. All rights reserved.

C:\Users\Administrator>\masm32\bin\Link /SUBSYSTEM:CONSOLE C:\Users\Administrator\hello_window.obj
Microsoft (R) Incremental Linker Version 5.12.8078
Copyright (C) Microsoft Corp 1992-1998. All rights reserved.

C:\Users\Administrator>
```

4. 执行：找到可执行文件(hello\_console.exe), 输入到 cmd 中执行可执行文件。程序成功输出 helloworld。

```
C:\Windows\system32\cmd.exe
Microsoft Windows [版本 6.1.7601]
版权所有 (c) 2009 Microsoft Corporation。保留所有权利。

C:\Users\Administrator>C:\Users\Administrator\hello_console.exe
Hello World!!

C:\Users\Administrator>
```

(B) 窗口输出 “HelloWorld”

1. 编辑：用编辑软件(Notepad)形成源程序(.asm)，hello\_window.asm.

```
hello_window.asm - 记事本
文件(F)  编辑(E)  查看(V)  帮助(H)

.386
.model flat, stdcall
option casemap :none
include \masm32\include\windows.inc
include \masm32\include\kernel32.inc
include \masm32\include\user32.inc
includelib \masm32\lib\kernel32.lib
includelib \masm32\lib\user32.lib
.data
    str_hello BYTE "Hello World!",0
.code

start:
    invoke MessageBox, NULL, addr str_hello, addr
    str_hello, MB_OK
    invoke ExitProcess, 0
END start
```

2. 编译：用汇编程序(\masm32\bin\ml.exe)对源程序进行汇编，形成目标文件(.obj)，格式如下：“\masm32\bin\ml /c /Zd /coff hello\_window.asm”。

```
C:\Users\Administrator>\masm32\bin\ml /c /Zd /coff C:\Users\Administrator\Desktop\hello_window.asm
Microsoft (R) Macro Assembler Version 6.14.8444
Copyright (C) Microsoft Corp 1981-1997. All rights reserved.

Assembling: C:\Users\Administrator\Desktop\hello_window.asm

*****
ASCII build
*****
```

3. 连接：用连接程序(\masm32\bin\link.exe)对目标程序进行连接，形成可执行文件(.exe)，格式如下：“\masm32\bin\Link /SUBSYSTEM:WINDOWS hello\_window.obj”。

```
C:\Users\Administrator>\masm32\bin\Link /SUBSYSTEM:CONSOLE C:\Users\Administrator\Desktop\hello_console.obj
Microsoft (R) Incremental Linker Version 5.12.8078
Copyright (C) Microsoft Corp 1992-1998. All rights reserved.

C:\Users\Administrator>\masm32\bin\Link /SUBSYSTEM:CONSOLE C:\Users\Administrator\Desktop\hello_window.obj
Microsoft (R) Incremental Linker Version 5.12.8078
Copyright (C) Microsoft Corp 1992-1998. All rights reserved.

C:\Users\Administrator>
```

4. 执行：找到可执行文件(hello\_window.exe),输入到 cmd 中执行可执行文件。程序成功在窗口输出 helloworld。



#### 四、实验结论及心得体会



实验结论：

在这次实验中，我学习了如何在 Win32 汇编环境下使用 MASM32 来编写程序，并实现了两个不同的输出方式：命令行输出和窗口输出。

首先，我学会了使用 MASM32 编写一个控制台应用程序，通过命令行输出了 "HelloWorld"。也学会如何使用 MASM32 的汇编语言来调用 Windows API 函数来完成控制台输出。接下来，我学会了编写一个窗口应用程序，同样输出了 "HelloWorld"，但这次是通过一个弹出窗口实现的。这个实验展示了如何使用 MASM32 编写一个 Windows GUI 程序，并使用 Windows API 函数来创建一个简单的消息框。

心得体会：

MASM32 提供了一套工具和库，使得 Win32 汇编编程变得相对容易。熟悉这些工具和库可以帮助我们更高效地编写汇编程序。深入了解 MASM 汇编语言是必要的。在这次实验中，我学习了如何定义数据段、代码段以及如何使用汇编语言指令。

同时在这次实验中，我调用了一些 Windows API 函数，如 `ExitProcess` 和 `MessageBox`，以实现程序的功能。理解这些函数的参数和调用方式对于 Win32 汇编编程至关重要。

同时我发现，窗口应用程序和控制台应用程序之间有很大的区别。在本实验中，我们实现了两种不同的输出方式，这两种方式分别适用于不同的应用场景。掌握这两种方式可以让我们更好地适应不同的需求。