1. Function call และ stack frame

โดยทั่วไป ฟังชันมีวิธีจัดเก็บตัวแปรจำพวกที่อยู่ชั่วคราวอยู่ โดยบริเวณที่เก็บจะถูกเรียกว่า stack frame ซึ่งถูกจัดเก็บอยู่ภายใน stack ของ process

Frame pointer: Intel x86 architectures ใช้รีจิสเตอร์ ebp ในการจัดเก็บค่า frame pointer ซึ่งเป็น base address ของฟังชัน function frame

ภายในฟังชัน โค้ดที่ใช้เข้าถึงตัวแปร local ต่างๆจะถูกจัดอยู่ในรูป offset จาก frame pointer

Stack pointer: Intel x86 architectures ใช้รีจิสเตอร์ esp ในการจัดเก็บค่าของ stack pointer ซึ่งมีการเปลี่ยนแปลงค่าในช่วงที่ฟังชันทำงาน ซึ่งมีการ push และ pop ค่าใน stack

Frame pointer จะไม่มีการเปลี่ยนแปลงค่าในช่วงการทำงานของฟังชัน

1. สิ่งที่เกิดขึ้นในช่วงการทำงานของฟังชัน (อาจแตกต่างออกไปตามแต่สถาปัตการออกแบบ)

1.ค่าใน frame point ถูก Push ลงใน stack เพื่อให้สามารถนำกลับมาใช้ภายหลังได้

2.move ค่าที่อยู่ใน stack pointer ไปใส่ใน frame pointer เพื่อเป็นการจุดเริ่มต้นของ frame

3.ลบค่าออกจาก stack pointer ตามขนาดที่ฟังชันต้องการใช้ในการเก็บข้อมูล ซึ่งช่วยป้องกันพื้นที่จากการ push ข้อมูลไหม่ลงใน stack

4.คำสั่ง call จะ push ตำแหน่งของ return address ลงใน stack ก่อนเรียกใช้งานฟังชัน

5.โค้ดของฟังชันเริ่มทำงาน โดยการอ้างอิงค่าไปยังตัวแปร local จะเป็นการใช้ negative offset ออกจาก frame pointer

6.เมื่อสิ้นสุดการใช้ฟังชัน จะมีการ copy ค่าใน frame pointer ไปลงใน stack pointer เพื่อเคลียพื้นที่ที่ใช้ไปใน stack frame แล้วหลังจากนั้นจึง pop ค่าเก่าของ frame pointer ด้วยการใช้คำสั่ง leave

7.กลับจากการเรียกใช้ฟังชันด้วยคำสั่ง ret ซึ่งจะทำการ pop ค่า return address ออกจาก stack แล้วจึงย้ายการทำงานกลับไปยังจุดที่เรียกใช้ฟังชัน

3. การสร้าง static library

Gcc –c fun.c //สร้าง object

Ar cr libfunc.a func.o

4.การสร้าง shared (dynamic) Library

Windows

Gcc – shared –o libfunc.dll –Wl, --out –implib,libfunc.dll.a \* .o

MacOsX

Gcc -shared –o libfunc.dylib \*.o

Linux

Gcc –shared –o libfunc.so \*.o

4.การเรียกใช้ shared library ของภาษา c จาก python

From ctypes import cdll

C\_lib = cdll.LoadLibrary(“libfunc.dll”)

C\_lib.c\_function(123)