3rd session

/ /

* ال emulator كمان من ضمن الحاجات اللى بيستخدموها فيه
انى لو بيعمل processor جديد، بروح اكمل run instruction
على ال emulator بتاعه اللى انا بعمله، حتتكده من السيشن
اللى فاتت 😉

* ال bash بت Parse ال > وال < بتوع ال I/O redirection
الاول قبل ما ~~بت Parse~~ بتشغل ال Command بتاعك

Elf Format Cont.:-

* ال debug section موجود عشان يـ map بين ال instruction
اللى شغال وال code اللى انت شايفه

gcc 1.c 2.c -o exe --save-temps

دة هيطلعلك ال executable بتاعك وكمان ال files
اللى بتطلع فى النص اللى هما ال s. وال i. وال o.

reading elf:-

* لو فتحت ال elf.h هتلاقى ال data structures اللى مستخدمة
وشكل ال elf وحاجات حلوة كده

* بس فى command اسمه readelf، ده بتديله ال elf
بتاعك، و option لازم، option، و بتقرأ بيه
ال elf

/ /

Ex: readelf -a elfName:
display all the informations he can get

Ex: readelf -h elfName:
display elf header

Ex: readelf -S elfName:
display elf sections

* فلو جينا عملنا readelf -h elfName نشوف ال header مثلا، هتلاقى منه Magic Number فى الاول، و نوعه ELF64 وكلام حاجة كده، ولو جيت فتحت ال elf عادى، هتلاقى منه characters غريبة لانه مش ASCII، بس لو انت عامل مثلا printf لـ string هتلاقيه، وهتلاقى برده فى الاول كلمة elf

Note:

*ال .bss فى ال elf او ال data memory اختصار block starting symbol، فى الحقيقة ال .bss مبيتزلش فى ال elf، اللى بيحصل ان هو كده كده فيه ال Variables اللى محطوطها uninit او init بس بـ zero، انا بقول هو بادئ منين واخره فين وال startup code يحجز المكان ده وفى ناس بتسميه better save space برده

* البرنامج بتاعك اما يجي يـ start بـ start مش ال start_، ولو عملت readelf -a elfName وعملت Search على start_ هتلاقى طلعلك address، يعنى كان 1100 مثلا ممكن يتغير عندك عادى

/ /

* ولو جيت عملت readelf -h elfName هتلاقي ان الـ Entry point address كانت الـ 1100 اللي هي نفس الـ _start ، و بردو لو عملت readelf -S elfName هتلاقي ان الـ .text section بردو ليه address 1100 ، معنى كده ان الـ _start هي بداية الـ ~~main~~ program مش الـ main

Note that:-

* في جوا الـ header عندك بتاع الـ elf حاجة اسمها الـ Type طب ايه دي ؟

* روح على الـ elf.h اللي موجود في الـ path دي /usr/include/elf.h و اعمل vim و search على DYN هتلاقي الـ types ظهرتلك

* فانت عندك zero دي ملهاش نوع ، one دي relocatable ، 2 دي executable ، 3 دي shared object يعني DYN ، 4 دي File ، 5 دي corefile ، و دي عدد الـ types خالص ، فانت بتروح تكتب رقم الـ type في الـ elf و كده كده معمولة #define بنوعه و خلاص

* فانت لو بتـ debug برنامج عندك ، وانت عايز تعمل debugging اما يحصل مشكلة بس مش دايمًا ، فبتخلي الـ Kernel تديلك حاجة اسمها core file يعني ايه ؟ corefile دي الـ Kernel بتجيب snapshot من ~~Process~~ وضع الـ program وقت المشكلة ، يعني الـ state وقت حدوث المشكلة

/ /

* الـ snapshot دى بيكون فيها اكتر من حاجة زى مثلا :-

A- bss section
B- data section

How to get assembly From excutable:-

بنستخدم objdump ، بتديله الـ excutable يديك الـ assembly

```
objdump -S elffile
```

بس بيديك الـ addresses الحقيقة ، يعنى مش relocatable ،

Virtual memory

* بص انت عشان تـ excute اى program ، او process ، انت بتروح تـ excute فى الـ bash ، والـ bash تتعامل هى مع الـ Kernel ، الـ Kerenl بتروح تـ init الـ process address space ، وخلى بالك انا شغالين GPOS مش MC ، يعنى مش هنشغل rom كله فى الـ RAM ، و يروح ياخد الـ heap والـ bss والـ stack والـ data و يعملهم allocation فى المكان بتاعهم

* انت الـ addresses اللى موجودة دى كلها مش addresses حقيقة (physical address) لا دى Virtual

* الـ addresses لو بصيت عليها هتلاقيها بدأت بـ zero ، والـ kernel بتروح تـ map الـ addresses دى فى الـ memory بطريقة ما هنقولها بعدين.
طب ليه الحوار ده؟

1- protection:
ان لو عندك اكتر من process او program شغال محدش يبوظ التانى

2- Multiplixing data memory:-
ان لو الـ memory عندك كانت 4 GByte مثلا ، وعندك اكتر من process شغال ، كل واحد منهم فاكر انه معاه الـ 4 GByte كلهم مع ان اللى حاصل العكس وانا مقسمهم عليهم

4- Portability:-
ان الـ addresses دى محطوطة ورا بعض وانا بروح ازود عليها offset ، بالتالى اقدر اكملها compile على اكتر من جهاز عادى بدون خوف ان لو الـ addresses الـ Fixed اللى محطوطة دى فاضية ولا لا

How does Virtualization occurs: → بعت عالاسلايد زمع الشرح

* المسؤول عن الـ Virtualization عمتا هو الـ Memory managment unit «MMU»

* بتقسم الـ process بتاعك كـ pages لـ RAM ، وتقسم الـ RAM بتاعتك لـ pages بردو ، والـ Kernel بتمسك لكل process جدول بتـ map بيه كل page في الـ process موجودة فين في الـ main Memory ، اسمه page table

* لو الـ Kernel طلبت virtual address مش موجود بيحصل page Fault

* ممكن يكون عندك اكتر من process ليهم نفس الـ Page في الـ main memory اسمها Shared page وده لو هما بيـ Communicate ـوا مع بعض

* لو جيت تحط page جديدة في الـ RAM و ملقتش مكان ساعتها بيحصل Virtual RAM ، وفي page بتتبعت على الـ secondary storage (SSD-HDD) ، والـ page اللي بتتبعت دي بتكون الاقل استخداما ، او اقدم page موجودة ، او محدش لها block على حاجة على حسب بقى الـ algorithm بتاعك

* الحوار طويل وفيه تفاصيل كتير بس حلو عليك كده اوي

System calls:-

* دى الحاجات اللى انت كـ application بتستخدمها عشان تـ interface مع الـ Kernel

* اعمل man syscalls واقرأ منه شويت (:

* احنا اتكلمنا الـ session دى مع open و read و write و close، فاعمل man و اديله session2 و اسم الـ system call واقرأ منهم شويت الحوار سهل

* الـ system calls اللى فوق دول بستخدمهم عشان اتكامل مع files، الـ linux كمثلا بيعمل كده مع كل حاجت كـ file وده اللى مخلى عدد الـ syscalls قليل بتاع 448

* تقدر تجيب على الـ syscalls كلهم فى الـ file ده /usr/include/x86_64-linux-gnu/asm/unistd_64.h، وهتلاقى ارقام لكل syscall، ده لان الـ linux اول الـ Kernel بيتعامل معاهم بارقامهم مش اساميهم

* الـ file descriptor ده الـ ID بتاع الـ file جوا الـ process، بالتالى ممكن يتكرر بين الـ processes

لو عايز ابعت حاجت للـ main، بستخدم argc و argv من الـ cmd