Sistemi Operativi[1]

Mattia Monga

Dip. di Informatica
Università degli Studi di Milano, Italia
mattia.monga@unimi.it

a.a. 2013/14

Lezione XVIII: JOS

Iniziare con JOS

Servono 512MB di ram e `persistence-jos.qcow` in modo da salvare il proprio lavoro.

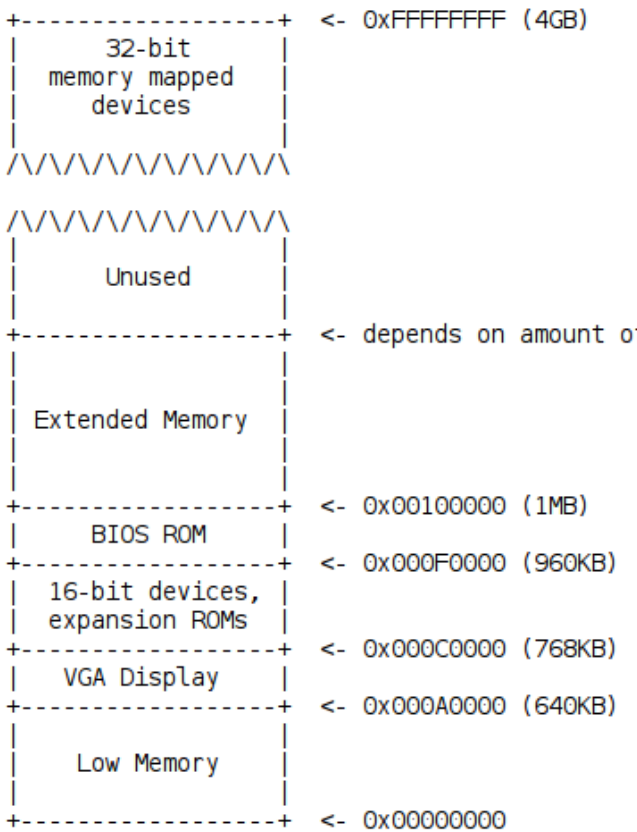
```
1  $ cd joslab
2  $ make
3  $ make qemu-nox
4
5  K> kerninfo
6  Special kernel symbols:
7     _start 0010000c (phys)
8     entry f010000c (virt) 0010000c (phys)
9     etext f0101a6d (virt) 00101a6d (phys)
10    edata f0112300 (virt) 00112300 (phys)
11    end f0112944 (virt) 00112944 (phys)
12 Kernel executable memory footprint: 75KB
```

Per uscire `Ctrl-a+x`

Struttura dell'esercitazione

Seguiremo
http://pdos.csail.mit.edu/6.828/2012/labs/lab1/
(spesso semplificando per motivi di tempo: non è vietato cercare di seguire tutti gli spunti del corso MIT! Tenete conto che gli studenti MIT hanno circa 2 settimane per realizzare gli obiettivi di ogni lab)

Layout della memoria



La mappa della memoria è definita dal costruttore. Generalmente accessibile via firmware o con tecniche di probing (GRUB2 fornisce un comando lsmmap)

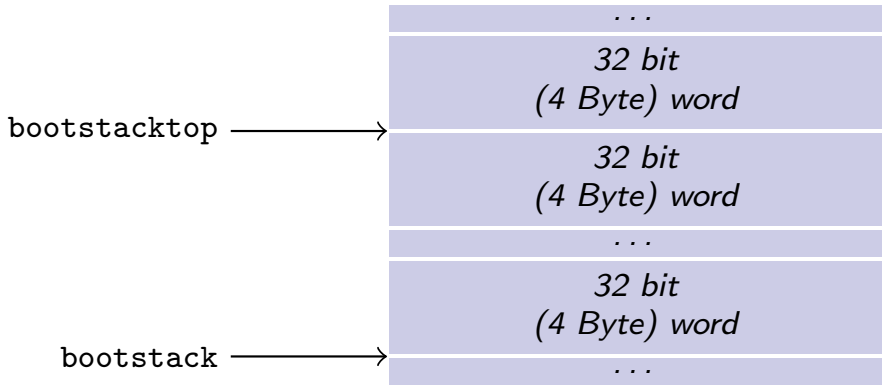
Layout della memoria

Start

```
1 [f000:fff0] 0xffff0: ljmp $0xf000,$0xe05b
```

L'indirizzo fisico è calcolato secondo il Real-Mode addressing (a 16 bit)

Lo stack



- ESP == bootstacktop
- bootstacktop == bootstack + KSTKSIZE
- Una push *sottrae* 4 Byte all'indirizzo ESP, una pop li *aggiunge*. (ESP è sempre divisibile per 4)
- Una call gestisce automaticamente il salvataggio dell'indirizzo di ritorno sullo stack, mentre EBP deve essere gestito a mano (salvandovi il vecchio ESP in modo da poter identificare facilmente il *record di attivazione* o *stack frame*)

Indirizzi

Nei manuali x86 si parla di 3 tipologie di indirizzi

virtuali quando sono relativi ad un segmento: un puntatore C è un *offset*

lineare selettore di segmento + offset permette di calcolare un indirizzo nello spazio di indirizzamento (virtuale) lineare 0–4GB

fisico l'indirizzo lineare è "mappato" su un indirizzo fisico dalla MMU (che non può essere saltata!)

Come manipolare gli indirizzi

Segmentazione e MMU non possono essere saltati: il programmatore "vede" esclusivamente indirizzi virtuali.

- JOS configura tutti i segmenti (in `boot/boot.S` tramite la prima GDT) in `0-0xffffffff` (0–4GB), quindi il segmento può essere ignorato
- Quando serve manipolare indirizzi fisici (che non possono essere dereferenziati) devono essere usati *numeri* che sarà utile contrassegnare con il tipo physaddr_t
- Un numero che può essere dereferenziato (perché si tratta di un indirizzo virtuale) verrà contrassegnato con uintptr_t e per derefenziarlo come T va interpretato come T$*$.

Il mapping iniziale

I kernel sono generalmente caricati a un indirizzo (lineare) alto, p.es. `0xf0100000` (3,75GB), che potrebbe perfino non esistere nello spazio fisico.

- il programmatore del kernel (e il programma!) usa `0xf0100000` (virtuale)
- il boot loader carica il kernel all'indirizzo `0x00100000`
- il boot loader istruisce la MMU perché mappi `0xf0100000` $\rightarrow$ `0x00100000`

le prime page table

- La page table 'zeresima' in `boot/boot.S` configura il mapping *identità*, quindi indirizzi lineari uguali a fisici.
- La prima vera page table è in `kern/entrypgdir.c`

lineare	fisico
0xf0000000 (KERNBASE)	0x00000000
...	...
0xf0400000	0x00400000 (4MB)
0x00000000	0x00000000
...	...
0x00400000	0x00400000 (4MB)
*	eccezione

Macro che sostituiscono la MMU

0xf0000000 == KERNBASE $\rightarrow$ 0x00000000
0xf0100000 == KERNBASE + 1MB
0xf0400000 == KERNBASE + 4MB $\rightarrow$ 0x00400000

Alla fine del lab2 verranno mappati 256MB. Si noti che esiste una relazione semplice fra fisico e lineare: quando serve il programmatore può calcolare l'indirizzo lineare aggiungendo KERNBASE al fisico. Per farlo meglio usare KADDR (e PADDR per l'inverso) che controllano che il numero cui si applica sia sensato.