

# Sistemi Operativi[1]

Mattia Monga

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

a.a. 2014/15







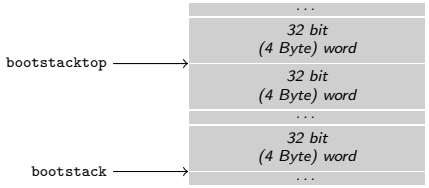


## Lezione XX: Gestione della memoria in JOS





## 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`