



Sistemi
Operativi

Bruschi
Monga

Software
factory

Make

JOS

Layout della memoria
Stack

Sistemi Operativi¹

Mattia Monga

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

a.a. 2019/20



Sistemi
Operativi

Bruschi
Monga

Software
factory

Make

JOS

Layout della memoria

Stack

Lezione XVII: Software factory



- UNIX nasce come sistema *per i programmatori* (l'unica tipologia di utente all'inizio degli anni '70...)
- progettato insieme ad un linguaggio di programmazione (C)
- la 'filosofia di UNIX' (piccoli programmi che fanno molto bene una sola cosa su file) si adatta perfettamente al paradigma di sviluppo **edit-compile-debug**
- tool all'avanguardia nell'elaborazione di *file di testo* (per lo più organizzati per "righe") e per la scrittura dei programmi di elaborazione stessi (lex, yacc,...)



- Editor: `ed`, `vi`, `emacs` manipolano arbitrariamente i byte di un file, generalmente interpretandoli come caratteri stampabili (testo)
- Compilatore: `cc` (`gcc`)
 - 1 `cc` sorgente (`.c`) \rightsquigarrow assembly (`.s`)
 - 2 `as` assembly \rightsquigarrow *oggetto* (`.o`)
 - 3 `ar` archivia diversi oggetti in una *libreria* (`.a`)
 - 4 `ld` *oggetti* e *librerie* \rightsquigarrow eseguibile (`a.out`) (il formato storico è COFF, oggi ELF)

Si noti che a sua volta anche la compilazione vera e propria è fatta da due passi (pre-processor `cpp` e compilazione `cc1`).



- Scrivere in assembly (nasm) una funzione `somma` che restituisce (in `eax` secondo la convenzione del C) la somma di due interi (passati sullo stack, secondo la convenzione del C)
- Scrivere un programma C che usa la funzione `somma`
- Collegare i due programmi in un unico eseguibile



Make

Stuart Feldman, 1977 at Bell Labs.

Permette di specificare **dipendenze** fra processi di generazione.

Dipendenze: se cambia (secondo la data dell'ultima modifica) un prerequisito, allora il processo di generazione deve essere ripetuto.

```
helloworld.o: helloworld.c
    cc -c -o helloworld helloworld.c
```

```
helloworld: helloworld.o
    cc -o $@ $<
```

```
.PHONY: clean
clean:
    rm helloworld.o helloworld
```

Sistemi
Operativi

Bruschi
Monga

Software
factory

Make

JOS

Layout della memoria
Stack



- Scrivere in assembly (nasm) una funzione `somma` che restituisce (in `eax` secondo la convenzione del C) la somma di due interi (passati sullo stack, secondo la convenzione del C)
- Scrivere un programma C che usa la funzione `somma`
- Collegare i due programmi in un unico eseguibile
- **Codificare il procedimento in un Makefile**



Sistemi
Operativi

Bruschi
Monga

Software
factory

Make

JOS

Layout della memoria
Stack

Lezione XX: Gestione della memoria in JOS



Iniziare con JOS

Servono almeno 512MB di ram (-m 512 in Qemu) e persistence-jos.qcow (-hda persistence-jos.qcow in Qemu) in modo da salvare il proprio lavoro.

```
$ cd /home/user/joslab
$ make
$ make qemu-nox
```

```
K> kerninfo
```

```
Special kernel symbols:
```

_start		0010000c	(phys)
entry	f010000c	(virt)	0010000c (phys)
etext	f0101a6d	(virt)	00101a6d (phys)
edata	f0112300	(virt)	00112300 (phys)
end	f0112944	(virt)	00112944 (phys)

```
Kernel executable memory footprint: 75KB
```

```
Per uscire Ctrl-a+x
```



Sistemi
Operativi

Bruschi
Monga

Software
factory

Make

JOS

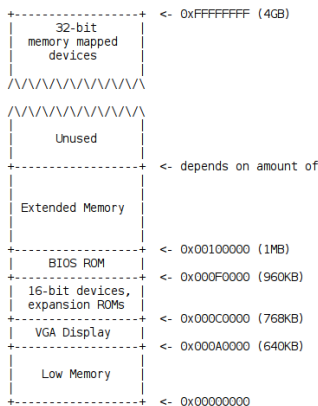
Layout della memoria

Stack

Seguiremo

<http://pdos.csail.mit.edu/6.828/2016/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)

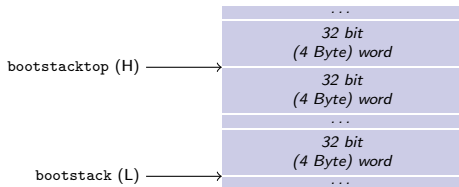


La mappa della memoria è definita dal costruttore. Generalmente accessibile via firmware o con tecniche di probing (GRUB2 fornisce un comando `lsmmmap`)



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

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



- $ESP == bootstacktop$
- $bootstacktop == bootstack + KSTKSIZE$
- 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*)