



LAB 1

INTRODUZIONE A OS161

PROGRAMMAZIONE DI SISTEMA (JZ-ZZ)

2023/24

ANDREA PORTALURI

OBIETTIVI DEL LABORATORIO

- Introduzione all'ambiente OS161
- Navigare nelle directories di OS161
- Prime modifiche al kernel
- Usare il debugger GDB

OS161

- Sistema operativo didattico UNIX scritto in linguaggio C
- È eseguito su macchina virtuale (sys161, simula architettura MIPS)
- La versione 2.x (quella da noi utilizzata) supporta i sistemi a multiprocessore
- Macrokernel, predilige semplicità di comprensione (a discapito delle performances)

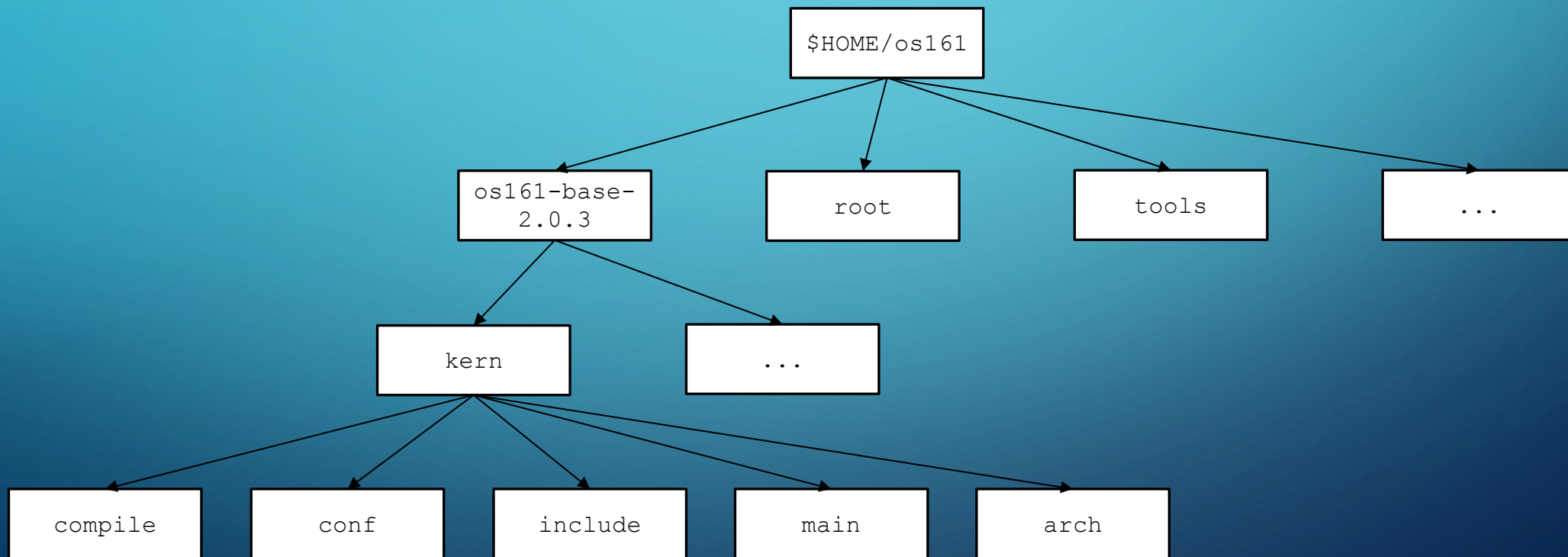
IL FRAMEWORK

OS161 include:

- I files sorgente del kernel per:
 - modificare e aggiungere features
 - eseguire e debuggare
- Una toolchain per:
 - cross-compiling
 - eseguire il kernel sulla macchina virtuale

OS161 DIRECTORY TREE

Supponendo l'installazione della directory \$HOME:



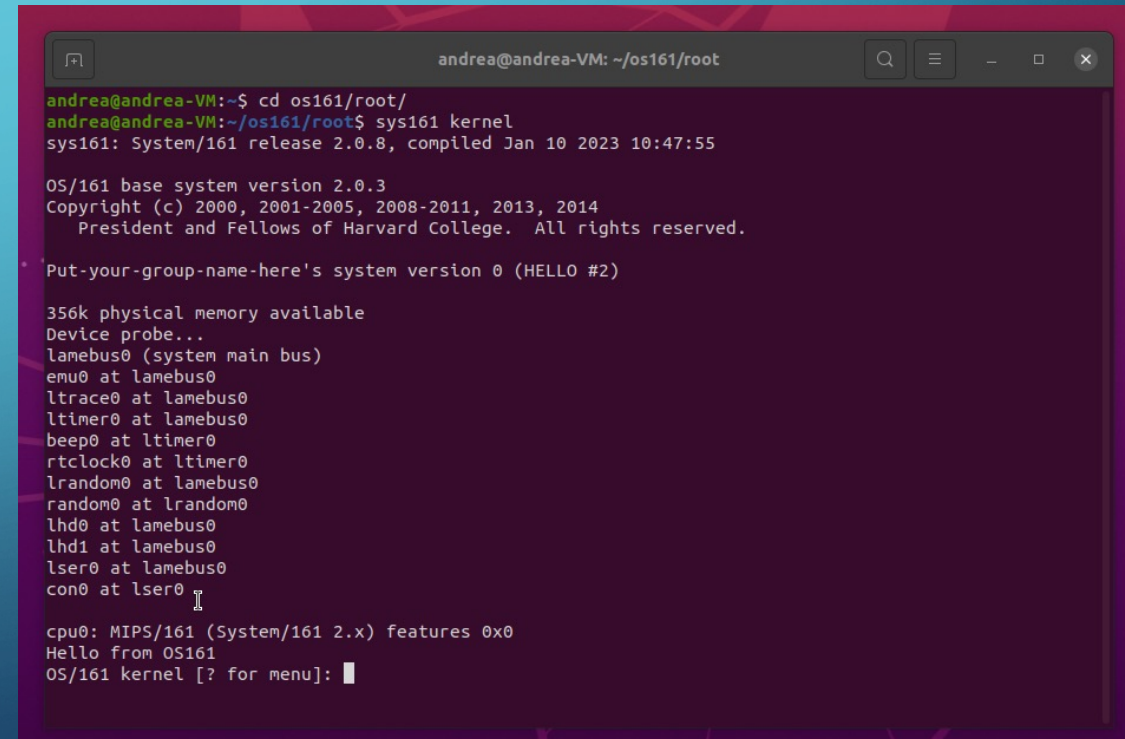
LAVORARE CON OS161

Eeguire il bootstrap del kernel:

- in `$HOME/os161/root` lanciare il comando:

```
>> sys161 kernel
```

- Appare un'interfaccia utente base
- `sys161` è un eseguibile che simula una macchina MIPS. Bisogna interagire con il kernel, NON con `sys161`.



```
andrea@andrea-VM: ~/os161/root
andrea@andrea-VM:~$ cd os161/root/
andrea@andrea-VM:~/os161/root$ sys161 kernel
sys161: System/161 release 2.0.8, compiled Jan 10 2023 10:47:55

OS/161 base system version 2.0.3
Copyright (c) 2000, 2001-2005, 2008-2011, 2013, 2014
  President and Fellows of Harvard College.  All rights reserved.

Put-your-group-name-here's system version 0 (HELLO #2)

356k physical memory available
Device probe...
lamebus0 (system main bus)
emu0 at lamebus0
ltrance0 at lamebus0
ltimer0 at lamebus0
beep0 at ltimer0
rtclock0 at ltimer0
lrando0 at lamebus0
random0 at lrando0
lhd0 at lamebus0
lhd1 at lamebus0
lser0 at lamebus0
con0 at lser0

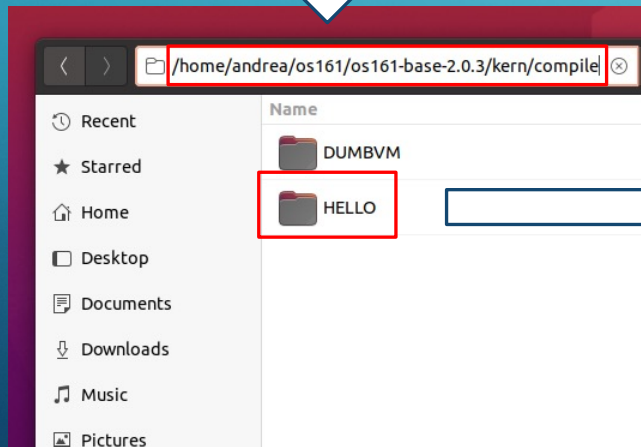
cpu0: MIPS/161 (System/161 2.x) features 0x0
Hello from OS/161
OS/161 kernel [? for menu]:
```


BUILDARE IL KERNEL

In `$HOME/os161/os161-base-2.0.3/kern/conf` creare una nuova versione “HELLO” (copiando la precedente “DUMBVM”) e configurarla:

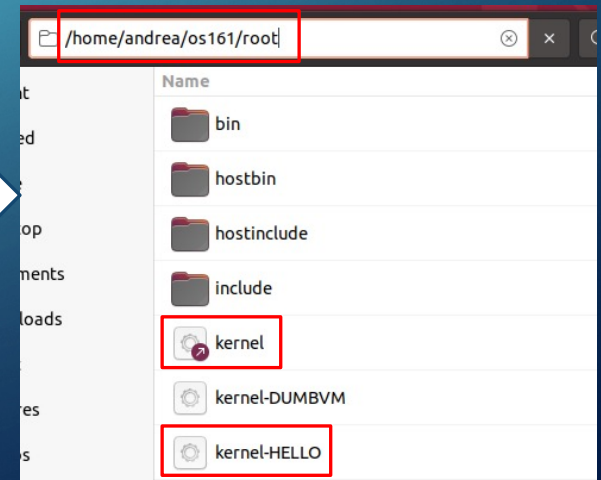
```
>> cp DUMBVM HELLO  
>> ./config HELLO
```

Una nuova cartella (“HELLO”) verrà creata in `./kern/compile`



In `$HOME/os161-base-2.0.3/kern/compile/HELLO`:

```
>> bmake depend  
>> bmake  
>> bmake install
```



MODIFICARE IL KERNEL

Vogliamo creare una funzione *hello* che stampi a schermo un messaggio di benvenuto.

- In `/kern/main`, creiamo il file *hello.c* (copiando gli include dal file *main.c*) dove definiremo la funzione come:

```
void hello (void) {  
    kprintf("Hello from OS161\n");  
}
```

Le funzioni
necessitano `void` in
assenza di parametri

- In `/kern/include`, creiamo un header *hello.h* con il prototipo di funzione. Includiamo “*hello.h*” in *hello.c*
- In `/kern/main/main.c`, aggiungiamo la funzione a *kmain* (tra *boot()* e *menu()*) e includiamo “*hello.h*”
- Modifichiamo `/kern/conf/HELLO`, aggiungendo `options hello`
- Infine, in `/kern/conf/conf.kern`, definiamo l’opzione sopra descritta come:

```
defoption hello                # defoption "nome_dell_opzione"  
  
optfile hello main/hello.c    # optfile "nome_dell_opzione" "file da includere"]
```

ATTENZIONE: ogni modifica dei files sorgente necessita una nuova compilazione (`./config`, `bmake depend`, ...) per essere attiva.

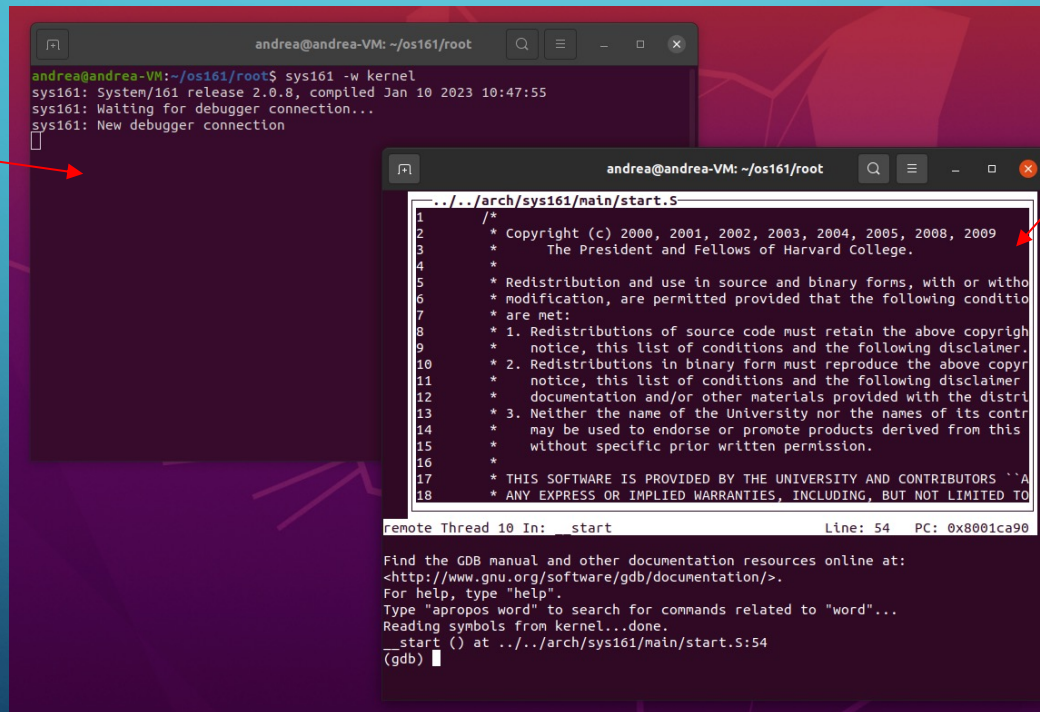
DEBUGGARE OS161

Occorre mettere in ascolto sys161 su un socket. In `$HOME/os161/root`, eseguire in due finestre di terminale:

```
>> sys161 -w kernel
```

```
>> mips-harvard-os161-gdb -tui kernel
```

Versione del debugger con
una interfaccia grafica



In `.gdbinit` (file nascosto) è
definita una funzione che
permette di collegare il
debugger all'avvio

DEBUGGARE OS161

Comandi utili per gdb:

- `b <func>` : crea un breakpoint
- `c` : continua l'esecuzione fino al primo breakpoint
- `s` : esegue uno step dentro la funzione
- `n` : continua l'esecuzione di una linea di codice
- `up` : risale lo stack di un livello
- `p <var>` : stampa il valore di `var`
- `dis <#cp>` : disabilita il checkpoint

Altri debugger più “user-friendly” sono:

- `>> ddd --debugger mips-harvard-os161-gdb kernel`

- `>> emacs &`

Selezionare dal menu “Tools > Debugger” e in basso inserire la linea:

```
mips-harvard-os161-gdb -i=mi kernel
```

**Choose
your
fighter**

DEBUGGARE OS161

Come esempio di navigazione di codice con gdb è utile provare a debuggare un built-in test di thread.

Tramite il vostro debugger preferito, osservare le funzioni:

- `kmain`
- `thread_create`
- `thread_yield`
- `thread_fork`
- `thread_destroy`

mentre il sistema esegue il test *tt1* dal menu principale.