

PdS 2022 - Laboratorio di OS161 - 1

Il Sistema SIS161 – OS161 è installato in una macchina virtuale VBOX OSE, con S.O. Linux Ubuntu 20.04 (versione in inglese). La macchina non corrisponde esattamente al sistema usato nelle videolezioni, che fanno riferimento a Ubuntu 14. Non ci sono sostanziali differenze.

La macchina virtuale (file ubuntu20-os161-2022) è disponibile per download nella sezione materiale del portale della didattica (dropbox). Si consiglia di installare VirtualBox (<https://www.virtualbox.org>). Volendo la versione “vecchia” (Ubuntu 14), la si trova su linux-pds-ovf10.ova. Non ci sono sostanziali differenze.

Oltre alla macchina virtuale Ubuntu pre-installata, è possibile

- utilizzare un proprio sistema (virtualizzato o no) Linux (Ubuntu o altro) su cui installare direttamente os161 utilizzando il kit kit-per-installazione-su-ubuntu20
- in alternativa utilizzare Docker come riportato in kit-per-docker (<https://github.com/marcopalena/polito-os161-docker>)

Sistemi MAC con processore M1

Per i sistemi con processore M1 NON è possibile usare VBOX, né importare la macchina virtuale Ubuntu pre-installata. Si consiglia di generare un proprio sistema Ubuntu20 (con virtualizzazione Parallel) e seguire le istruzioni in kit-per-installazione-su-ubuntu20, oppure di usare la soluzione con Docker (kit-per-docker)

Se la macchina virtuale non parte

Verificare per prima cosa che la copia da rete non sia corrotta.

Eventualmente provare a modificare le opzioni di virtualizzazione vbox in settings/impostazioni – system

Provare eventualmente Vbox 6.0.

Se proprio non si riesce (su alcune piattaforme HW in effetti è meglio ripartire da 0) passare alle stesse soluzioni indicate per MAC-M1.

Per attivare la macchina virtuale, eseguire i passi sotto-elencati

- 1) Scaricare ed installare VirtualBox (<https://www.virtualbox.org>).
- 2) Scaricare la macchina virtuale di OS161.
- 3) Avviare VirtualBox.
- 4) Importare la macchina virtuale utilizzando il comando “*import appliance*” (“importa applicazione virtuale”).
- 5) Avviare la macchina virtuale.
- 6) E’ abilitato automaticamente il login con (nella versione ubuntu14 user: pds, pass: pdsuser):

- USER: os161user
- PASSWORD: os161user

7) Qualora si volesse “salvare” oppure portare la macchina virtuale (con modifiche personali) su un altro PC, la si può esportare, con una procedura duale: comando “export appliance” (esporta applicazione virtuale).

Ambiente OS161

Su Ubuntu 20

Il sistema operativo os161 è stato pre-installato nel direttorio **os161/os161-base-2.0.3**, mentre i pacchetti SW richiesti, binutils, compilatore gcc e debugger gdb, emulatore MIPS, sono installati nella cartella **os161/tools**. SI NOTI CHE TUTTI I DIRETTORI SONO INDICATI A PARTIRE DA \$HOME, che è /usr/os161.

Su Ubuntu 14

Il sistema operativo os161 è stato pre-installato nel direttorio **os161/os161-base-2.0.2**, mentre i pacchetti SW richiesti, binutils, compilatore gcc e debugger gdb, emulatore MIPS, sono installati nella cartella **os161/tools**. SI NOTI CHE TUTTI I DIRETTORI SONO INDICATI A PARTIRE DA \$HOME, che è /usr/pds.

Si ricorda che il sistema OS161 è attivato sull'emulatore di processore MIPS, SYS161. Al fine di ri-compilare OS161, nonché di fare debug e altre operazioni (ad esempio visualizzazione di file eseguibili), sono necessari programmi previsti per la piattaforma MIPS, che sono stati installati con il prefisso “**mips-harvard-os161-“**: **mips-harvard-os161-gcc**, **mips-harvard-os161-gdb** (eseguibili in **os161/tools/bin**).

Il sito web di riferimento è: <http://www.os161.org/>.

Un altro sito con informazioni interessanti è quello del corso cs350 dell'università di Waterloo (Canada): <http://www.student.cs.uwaterloo.ca/~cs350/common/OS161main.html>

Sul desktop della macchina virtuale è presente un link al codice sorgente “navigabile” da un web browser. Per eventuali installazioni sul proprio PC, occorre seguire le istruzioni presenti su uno dei siti sopra proposti. Si forniscono a parte degli script di installazione per un sistema Linux-Ubuntu.

Informazioni dettagliate su avvio ed esecuzione di OS161 sono raggiungibili, ad esempio, dal link: [Working with OS161](#)

La cartella iniziale di **lavoro** proposta è **os161/root** (su Ubuntu 20) oppure **pds-os161/root** (su Ubuntu 14).

ATTENZIONE: ci sono quindi i seguenti direttori

- *os161/os161-base-2.0.3: contiene sorgenti, file di configurazione, compilazione ed eseguibili, di os161 e dei tool utilizzati (è quindi l'area in cui si modifica e ri-compila os161)*
- *os161/tools: contiene i tool utilizzati per fare compilazione, debug e altro (adatti al processore MIPS virtualizzato)*
- *os161/root: si tratta del direttorio in cui eseguire il (fare boot del) sistema operativo, ed eventualmente attivare processi user (è l'area in cui si esegue e si testa il sistema os161).*

Per fare bootstrap di OS161 in ambiente emulato sys161 (da una shell, attivabile in una finestra “Terminal”), sono possibili due modalità:

- Esecuzione normale:

```
cd $HOME/pds-os161/root  
sys161 kernel
```

Compare una videata come la seguente

```
sys161: System/161 release 2.0.8, compiled Mar 30 2016 12:38:39
```

```
OS/161 base system version 2.0.2
```

```
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 (DUMBVM #1)
```

```
788k physical memory available
```

```
Device probe...
```

```
lamebus0 (system main bus)
```

```
emu0 at lamebus0
```

```
ltrace0 at lamebus0
```

```
ltimer0 at lamebus0
```

```
beep0 at ltimer0
```

```
rtclock0 at ltimer0
```

```
lrandom0 at lamebus0
```

```
random0 at lrandom0
```

```
lhd0 at lamebus0
```

```
lhd1 at lamebus0
```

```
lser0 at lamebus0
```

```
con0 at lser0
```

```
cpu0: MIPS/161 (System/161 2.x) features 0x0
```

```
OS/161 kernel [? for menu]:
```

Dalla quale sono attivabili comandi (menu con ?). Alcuni comandi (es. quelli selezionabili con ?o) non sono completamente disponibili, in quanto OS161 non è un sistema completo (richiede aggiunte da parte dello studente).

- Esecuzione con *debugger*.

ATTENZIONE: il programma eseguito sul sistema Ubuntu è *sys161*. *Sys161* è un eseguibile per la macchina host (con processore Intel o AMD, *sys161* viene già fornito e **NON** necessita debug). Il comando “*sys161 kernel*” manda in esecuzione “*sys161*”, un programma che funge da macchina virtuale con processore MIPS, per la quale “*kernel*” è un file “eseguibile” che viene caricato e mandato in esecuzione: obiettivo del laboratorio è interagire con “*kernel*”, eseguito su macchina MIPS (*sys161*), **NON** interagire con *sys161*.

Occorre evitare di fare il debug dell’emulatore SYS161. Il debugger (per la piattaforma MIPS) deve quindi essere eseguito una volta avviato SYS161. A tale scopo occorrono due processi, uno per eseguire *sys161* (avviato in una modalità predisposta per fare debug del kernel) e uno per *mips-harvard-os161-gdb* (un debugger predisposto per macchina MIPS), comunicanti mediante socket. Si consiglia di attivare due finestre terminale.

Sulla prima, dal direttorio *pso-os161/root* eseguire il comando:

```
sys161 -w kernel
```

sulla seconda, dallo stesso direttorio

```
mips-harvard-os161-gdb kernel  
(gdb) dir ../../os161/os161-base-2.0.2/kern/compile/DUMBVM  
(gdb) target remote unix:./sockets/gdb
```

attenzione ad utilizzare lo stesso kernel. DUMBVM rappresenta una particolare versione, eventualmente modificata, del sistema operativo *os161*. Le tre righe rappresentano, rispettivamente:

1. L’eseguibile (MIPS) di cui fare debug.
2. Il direttorio (di compilazione) da cui partire per localizzare i file sorgenti (solo se si è interessati, **fortemente consigliato**, a una sessione di debug in cui si visualizzi il programma sorgente C). In questo caso è sufficiente localizzare i file oggetto, che contengono i riferimenti ai sorgenti.
3. La connessione al socket per far colloquiare *sys161* e debugger.

La creazione di una nuova versione viene descritta nel seguito (“Modificare il Kernel”). Se, dopo aver ricompilato altri kernel (es. *kernel-GENERIC*, *kernel-ASST1*, *kernel-HELLO*, ...) si utilizzerà uno di questi, va usato lo stesso nelle due finestre.

I due comandi iniziali (*dir* e *target*) sono stati per comodità inseriti in un file di comandi di inizializzazione per gdb (*pds-os161/root/gdbinit*), che definisce un unico comando, “*dbos161*” (e lo chiama, rendendo inutile l’esecuzione esplicita). QUINDI, NELLA VERSIONE INSTALLATA, NON SONO NECESSARI I DUE COMANDI, SE SI USA DUMBVM. SE SI CAMBIA VERSIONE, SI CONSIGLIA DI MODIFICARE OPPORTUNAMENTE, *pds-os161/root/gdbinit*, in modo da prevedere altre versioni. Si può infatti modificare tale file aggiungendo altri comandi per eventuali altre versioni.

La versione di gdb appena descritta corrisponde alla versione “in linea” (piuttosto scomoda). SI SCONSIGLIA DECISAMENTE DI USARE QUESTA VERSIONE. MEGLIO LE ESECUZIONI CON INTERFACCIA A FINESTRE, DESCRITTE SOTTO:

Per eventuale esecuzione di gdb con interfaccia a finestre, sono disponibili:

- gdb in versione con finestra per il sorgente (è la scelta più elementare/semplice):

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

- ddd (comando `ddd --debugger mips-harvard-os161-gdb kernel`)

- l'editor *emacs*, da cui è possibile attivare una finestra di debugger: una volta entrati in emacs (comando "emacs"), il debugger si attiva con `tools->Debugger`, modificando il comando proposto (riga in basso) con: `mips-harvard-os161-gdb -i=mi kernel`

ATTENZIONE: nel caso in cui sys161 sia andato in crash, occorre farlo ripartire, e ri-connettere il debugger, facendolo ripartire, oppure ri-eseguendo semplicemente il comando "*dbos161*". In tal caso, NON E' NECESSARIO FAR RIPARTIRE *mips-harvard-os161-gdb*, MA SOLO sys161. Va comunque, dal debugger, ri-connesso il socket (ad esempio rieseguendo "*dbos161*" o altro comando equivalente, eventualmente aggiunto a *pds-os161/root/gdbinit*).

VSCode

per usare VSCode, fare riferimento al file "Configurare VS Code per OS161.pdf", che si trova in os161-kit/Doc_VSCode.zip. VSCode è anche disponibile nella versione basata su Docker, per usare la quale si faccia riferimento al kit di installazione.

Modificare il kernel

(I sorgenti del kernel sono nella cartella *os161/os161-base-2.0.2/kern* e relative sotto-cartelle: si omette in questa parte il prefisso e si indicano i direttori a partire da "*kern*").

Una nuova versione del kernel implica modifica e/o aggiunta di file sorgenti. Si veda la descrizione [Building OS/161](#) (sezioni: "Configure a kernel" e "Compile a kernel").

Ogni nuova versione del kernel corrisponde a un file di configurazione (scritto in maiuscolo) nella cartella: *os161/os161-base-2.0.2/kern/conf*. Nelle vecchie versioni di os161 le configurazioni avevano nomi ASSTx (x=0,1,2,3,4, ...). Ora si propone DUMBVM (o DUMBVM-OPT) per la versione con gestione della memoria "dumbvm", GENERIC (o GENERIC-OPT) per una nuova versione, ad esempio la prima su cui lavorare, altri nomi, in funzione del tipo di lavoro effettuato.

Si consiglia di iniziare su DUMBVM, per la prima esecuzione, senza modifiche, per passare poi a una nuova versione, chiamata HELLO. Il primo lavoro consiste nell'inserire un messaggio aggiuntivo su video al bootstrap. Per fare questo, si chiede di aggiungere un file *hello.c* nel direttorio *kern/main*, nel quale creare una funzione *hello()*, che scrive un messaggio su video utilizzando la funzione *kprintf()*.

Si riportano qui le istruzioni dettagliate.

Creare un file *kern/main/hello.c*

Scrivere nel file appena creato una funzione *hello* che utilizza *kprintf()* per scrivere un messaggio a video. Pur se non necessario, si consiglia di creare un file *kern/include/hello.h*, che contenga il prototipo della funzione *hello*.

ATTENZIONE: il compilatore C usato necessita parametro void nel caso di assenza di parametri. Ad esempio, il prototipo di *hello* potrebbe essere

```
void hello (void);
```

Modificare *kern/main/main.c* inserendo una chiamata a *hello()*. Per utilizzare correttamente *kprintf()*, è necessario includere *types.h* e *lib.h*.

L'eventuale inclusione di `hello.h` (`#include "hello.h"`) va fatta sia in `main.c` che in `hello.c`.

Modificare `kern/conf/conf.kern` inserendo il nuovo file `hello.c` nell'elenco dei file. Ad esempio

```
file      main/main.c

file      main/menu.c

defoption hello

optfile   hello   main/hello.c
```

Riassumendo, al fine di chiamare in `main.c` una funzione presente in `hello.c`, è necessario (in `main.c`) il prototipo di tale funzione. Questo può essere fatto in modo esplicito, oppure (soluzione migliore) includendo al file `kern/include/hello.h`.

Riconfigurare e ricompilare il sistema

CONFIGURAZIONE

In `kern/conf` generare il file HELLO (ad es. copiando DUMBVM: ATTENZIONE, **aggiungere a tale file una riga “options hello”!**) e dare il comando

```
./config HELLO
```

Per fare in modo che `hello.c` e la chiamate a `hello()` siano visibili/attive solo con l'opzione di configurazione “*hello*”, sono quindi necessari i passi seguenti:

- Usare l'opzione *hello*, definita in `conf.kern` e rendere *hello.c* file opzionale (abilitato da tale opzione). Come conseguenza sarà generato automaticamente un file *opt-hello.h*, contenente `#define OPT_HELLO 1` oppure `#define OPT_HELLO 0`.
- Rendere opzionali le istruzioni di altri file che utilizzino *hello*:

- o Il file `hello.h`

```
#ifndef _HELLO_H_
#define _HELLO_H_

void hello(void);

#endif
```

potrebbe rendere opzionale il suo contenuto come segue

```
#ifndef _HELLO_H_
#define _HELLO_H_

#include "opt-hello.h"
#if OPT_HELLO
void hello(void);
#endif

#endif
```

- o La chiamata a `hello()` nel main può esser resa opzionale come segue

```
#if OPT_HELLO
    hello();
#endif
```

COMPILAZIONE

In *kern/compile/HELLO* effettuare

```
bmake depend
```

```
bmake
```

```
bmake install
```

Nel caso di errori di compilazione, è sufficiente ripetere *bmake*.

Provare a eseguire *os161* per verificare che al bootstrap sia stampato il messaggio.

Programmazione concorrente con OS161

Realizzare questo esercizio con configurazione: THREADS, anziché DUMBVM o HELLO: Si consiglia di tentare una configurazione nuova solamente come esercizio, nonostante non ve ne sia necessità (non si modifica nessun sorgente).

Built-in thread tests

Quando si avvia *os161*, tra le opzioni disponibili dal menu, si possono avviare i test per thread. Si tratta di funzioni NON caricate come eseguibili separati, ma direttamente linkati nel kernel (in pratica, quindi, di parti del kernel).

I programmi di test dei thread usano sincronizzazione basata su semafori. Si può tentare di tracciarne l'esecuzione in GDB, per verificare come lavora lo scheduler, come sono creati i thread, e cosa succede al context-switch. Per far questo, si consiglia di tracciare funzioni quali *thread_create()*, *thread_fork()*, *thread_yield()*, ...

Il test “tt1” stampa i numeri da 0 a 7 ad ogni loop del thread, “tt2” stampa solo ad inizio e fine dei thread (serve a dimostrare che lo scheduler non genera starvation). I threads sono avviati e girano per un po’ di iterazioni. Il test “tt3” utilizza semafori, che nella versione base di OS161 non funzionano correttamente: il problema sarà affrontato in un futuro laboratorio.

Il sorgente di avvio dei test si trova in *menu.c*.

Suggerimento per fare debug di programmi con thread

Siccome la funzione *thread_yield()* viene chiamata a intervalli random, per generare (e farne il debug) sequenze di esecuzione ripetibili si consiglia di usare un seme per inizializzazione fissa del generatore di numeri casuali (direttiva “*random*” in *sys161.conf*). Passare ad “autoseed” solo quando tutto funziona.