Questa è la Dedica*:*
ognuno può scrivere quello che vuole,
anche nulla . . .

Introduzione

Nell'ambito delle reti negli ultimi anni con l'avvento di internet si è assistito ad una crescita esponenziale delle apparecchiature connesse a questa rete. Fino ad oggi questa immensa rete è stata gestita per lo più attraverso indirizzamento delle interfacce di rete di cui i computer sono dotati.
Al momento però ci si trova a fare i conti con la carenza degli indirizzi e pian piano si procede verso una migrazione basata sul nuovo (ma non troppo) protocollo IPv6 che prevede uno spazio di indirizzamento molto più ampio e lascia spazio a nuove idee impraticabili con protocollo IPv4.
In seguito sentiremo parlare di IoT (Internet of Things) ed IoTh (Internet of Threads), due paradigmi che possiamo affermare essere il risultato di un'evoluzione sempre crescente dei sistemi informativi.
L'elaborato sarà strutturato in capitoli.
Il primo di questi si premura di introdurre il lettore allo stato attuale dei protolli di rete e del paradigma IoTh; nel secondo capitolo verrà trattato il progetto VsnLib, ovvero la libreria oggetto della tesi; in fine il terzo capito si occuperà di descrivere alcuni esempi di utilizzo della stessa con immagini dimostrative ed istruzioni.

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Internet of Threads

1.1 Paradigma

IoT (Internet of Things), ovvero l'internet delle cose.
Questo concetto vuole rapprensentare la diffusione dei sistemi embedded come veri e propri nodi di rete, possiamo definirlo il precursore del sistema che stiamo per descrivere ed è il sistema che ad oggi ha permesso di incontrare il nostro condizionatore o la nostra caldaia, ma addirittura il nostro tostapane, in rete.
Ad un certo punto si è sentita l'esigenza di elevare questa astrazione, nasce il concetto di IoTh.
L'idea diventa quella di avere processi come nodi di rete e non oggetti fisici, l'analogia è molto simile alla differenza tra telefoni fissi e telefoni cellulari, ovvero in passato era necessario pensare al luogo in cui una persona potesse trovarsi, mentre attraverso l'assegnamento di un numero personale oggi possiamo comunicare direttamente con la persona desiderata[**?**, **?**].
In termini di internet questo si traduce nella possibilità di migrare servizi da un capo all'altro del mondo con la semplicità di un kill and start.
Come possiamo notare infatti, con questa tecnologia lo stack è direttamente integrato nel programma.
Una nota va anche fatta in termini di sicurezza, ogni servizio può essere eseguito con il proprio stack di rete e questo impedirebbe a software di port mapping di carpire infor-

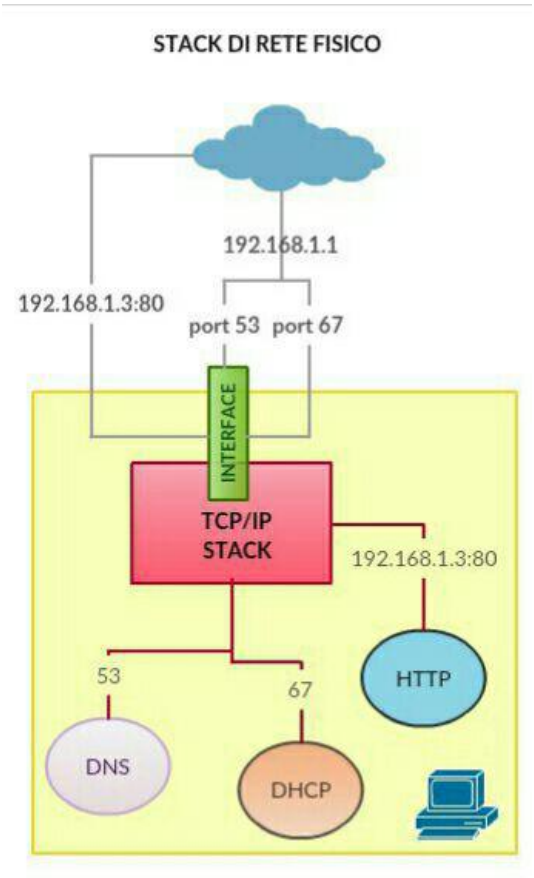

Figura 1.1: stack associato all'interfaccia fisica

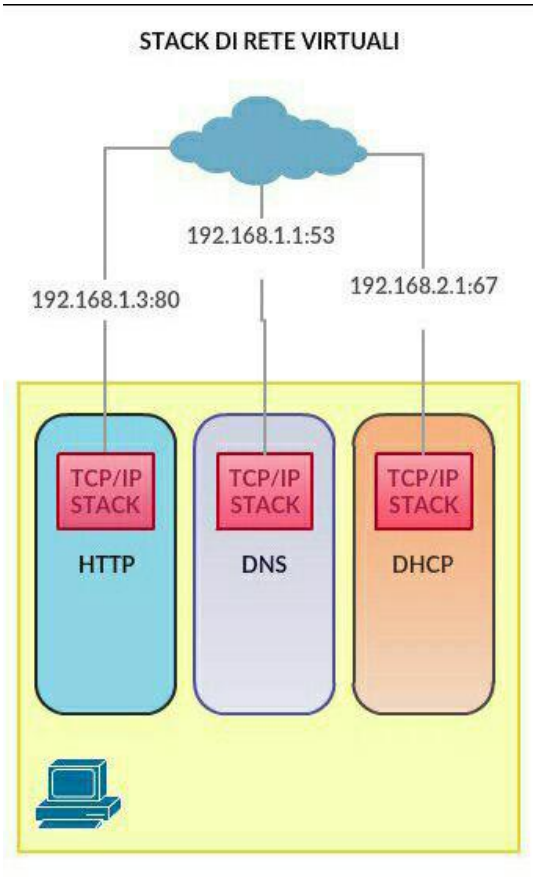

Figura 1.2: stack associatoai processi

mazioni sulla macchina che ospita detto servizio. Inoltre possedendo il proprio stack di rete possiamo eseguire i processi con un utente non privilegiato e pertanto anche un bug del demone non comprometterebbe l'intero sistema.

1.2 Stack Implementati

Diversi sono i progetti che si occupano di offrire ai processi il proprio stack di rete; tra questi ne verranno presi in esame tre, considerati più indicati per uno studio in quanto open source, Ognuno di essi offre la propria implementazione di stack di rete.
Uno stack di rete legato al processo permette di connettere lo stesso ad una rete reale,

tramite le interfacce fornite dal sistema, o virtuale (ad esempio VDE) come se fosse una macchina fisica a se stante, a questo punto ogni processo può staccarsi dal modo in cui la macchina che lo ospita gestisce la rete ed avere le proprie regole.

1.2.1 PicoTCP

Supportato da Tass Belgium (Altran); è lo stack più conosciuto e diffuso per sistemi embedded ed esistono anche dei progetti basati su IP di reti mesh realizzati con picoTCP[1].
Purtroppo per motivi commerciali alcune parti del codice non sono state rese pubbliche anche se rappresentano una piccola percentuale dell'intero progetto.

1.2.2 LWIP

Nasce dal progetto di Adam Dunkels pensato per sistemi embedded ed inizialmente non forniva supporto ad IPv6.
Light-weight IP vanta un core molto piccolo e la possibilità di eseguire anche senza sistema oprativo (single-threaded) e con il minor consumo di RAM possibile, inoltre è modulare ed offre la possibilità di aggiungere protocolli come DHCP e DNS solo in caso di necessità.
Supporta sia little che big endian e gira su processori a 8 e 32 bit, quindi funziona anche su un semplice ATMEGA.
Unica pecca è il protoccolo IPv6 che è ancora in fase sperimentale, infatti il supporto IPv6 può essere aggiunto ed è scaricabile da git.

1.2.3 LWIPv6

Precedemente abbiamo fatto notare che LWIP non aveva supporto IPv6 e che tutt'ora è ancora in fase sperimentale, ed è questo uno dei motivi della nascita di LWIPv6.
Quando al team di virtual square è sopravvenuta la necessità di avere uno stack per la rete vde non esisteva ancora nulla di maturo o comunque plug and play, da questa

[1] http://www.picotcp.com/mesh-design-guide

esigenza il team ha pensato di realizzare uno stack versatile in versione libreria tale per cui ogni processo potesse avere il proprio stack di rete.
Caratteristica principale di LWIPv6 è il suo motore ibrido che, di fatto funziona solo con IPv6 e mappa in questo protocollo le comunicazioni IPv4 utilizzando i primi 80 bit dell'indirizzo come flag di riconoscimento con tutti e 80 impostati a 1, ad esempio se volessimo rappresentare la maschera 255.255.255.0, che in esadecimale è equivalente a 0xffffff00 e la sua rappresentazione in IPv6 secondo LWIPv6 sarà la seguente 0xffffffff.ffffffff.ffffffff.ffffFFFF.ffffff00.

1.3 Netlink

1.3.1 Inter Process Comunication

Molti processi necessitano di scambiare informazioni per i motivi più disparati, dalla comunicazion e di rete a quella interna ad una sola macchina.
Viene da se pensare che costruire programmi che non interagiscono con il mondo esterno o con altri programmi sarebbe una risorsa limitata.
Con IPC intendiamo quindi l'insieme delle tecnologie adottate per permettere ai processi di comunicare tra essi, che siano ospitati sulla stessa macchina o distribuiti sulla rete, tra queste tecnologie esiste appunto quella utilizzata all'interno della libreria oggetto di questo elaborato.

1.3.2 Il Sistema IPC Netlink

Netlink è un sistema IPC (Inter Process Comunication) usato perchè in grado di mettere in comunicazione diversi task, solitamente serve per far comunicare task in user-space con task in kernel-space ma può far interagire anche processi entrambi in user-space.
Studiato per essere il successore di ioctl per le configurazioni ed il monitoraggio, si propone di essere più flessibile. Ma come avviene esattamente la comunicazione attraverso netlink?
Netlink utilizza un sistema di socket indicizzato in base ai processi ogni processo può definire socket diversi di tipo netlink (AF_NETLINK, NETLINK_GENERIC, NETLINK_XFRM)

per inviare e ricevere messaggi con e da altri processi, implementa un sistema di porte basato sul process ID dei processi; ad esempio per la comunicazione con il kernel viene usata la porta 0.
L'immagine successiva rende perfettamente il concetto, ed è una rielaborazione (per puri motivi stilistici) dell'immagine della documtazione di libnl (reperibile qui[2]).

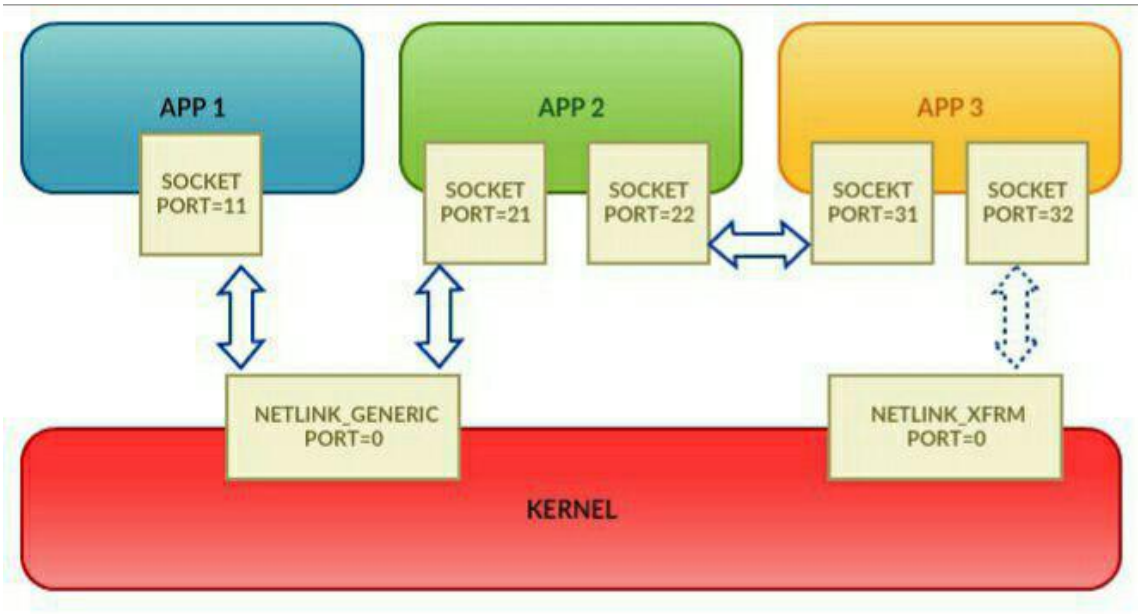

Figura 1.3: scambio di messaggi attraverso socket netlink

L'interfaccia di comunicazione è abbastanza standardizzata ma personalizzabile a livello di payload, ognuno può definire una struttura personalizzata per poi usarla per comunicare attraverso i socket netlink con altri processi.

1.3.3 Libnl

Netlink Protocol Library Suite (libnl), è una raccolta di librerie ed utility che forniscono API di comunicazione netlink basate su quelle del kernel linux e comprende [**?**]:

[2]https://www.infradead.org/ tgr/libnl/doc/images//addressing.png

libnl Core della libreria, offre uno strato di unificazione delle interfacce sulle quali poi si basano le altre librerie che pertanto di pendo da questa;

libnl-route Questa libreria si occupa di fornire API per la configurazione degli elemnti della famiglia NETLINK_ROUTE;

libnl-genl genl significa generic netlink e questa libreria offre una versione estesa del protocollo netlink;

libnl-nf API per configurazioni netlink basate su netfilter.

Capitolo 2

VsnLib

Strato di compatibilità tra pacchetti netlink e stack di rete virtuali.

2.1 Il Progetto

Gli stack analizzati in precedenza offrono a grandi linee la stessa tipologia di servizio seppur ognuna con le proprie caratteristice.
Nessuno dei progetti ha però tenuto in considerazione l'idea di utilizzare un'interfaccia di configurazione che fosse standard e pertanto è necessario usare le funzioni specifiche per ognuno di questi progetti, costringendo il programmatore a cambiare modalità di interazione da stack a stack.
Il progetto di creare questa libreria nasce proprio da questa esigenza, ovvero cercare di uniformare le interfacce di comunicazione in modo tale che l'utente non sia costretto ad adattarsi ogni qual volta decida di cambiare stack.

2.2 VnsLib

2.2.1 Preambolo

Per configurare uno stack di rete programmi come ip generano un pacchetto netlink con le informazioni necessarie lo inviano al kernel che lo elabora esegue le operazioni

e genera un messaggio di errore nel quale esiste un flag contenente il numero di errore generato (0 in caso di successo), a questo messaggio è associato un payload di risposta che può essere usato per generare un feedback da mostrare all'utente che sta interagendo con il programma.
VsnLib è un progetto in fase di sviluppo scritto in C e completamente opensource, liberamente scaricabile attraverso la piattaforma github al seguente link `https://github.com/simonepreite/vsnlib`.

2.2.2 Ambiente

La necessità primaria era quella di catturare queste comunicazioni netlink e per farlo la strada era quella di intercettare la system call di invio della richiesta contenente appunto il payload, questo momento è quello in cui viene eseguita la sendto.
In questo punto interviene la nostra libreria, che si interpone e di fatto fa le veci del kernel, al quale il pacchetto netlink non arriverà mai realmente.
Il progetto inizialmente utilizzava come motore di cattura delle system call la libreria purelibc[1], in seguito però si è giunti alla conclusione che costruire un modulo ad hoc all'interno del sistema vuos fosse un'alternativa più versatile, immediata e pulita, inoltre vuos ha un sistema di debug built in che rende più semplice l'individuazione dei problemi legati allo sviluppo.
Vuos è un progetto nato all'interno del team di sviluppo di virtual square, ha come obiettivo quello di virtualizzare parti del sistema operativo lasciando all'utilizzatore la scelta di cosa virtualizzare e cosa no. È completamente open source e viene rilasciato sotto licenza GPL, è scaricabile attraverso il repo su github `https://github.com/virtualsquare/vuos` nella versione pura che non comprende il modulo sperimentale unrealvsn, questa versione invece è reperibile al seguente link `https://github.com/simonepreite/vuos`.
Il modulo in questione si chiama unrealvsnlib per analogia alla libreria ma ognuno potrebbe costruire un suo modulo a seconda di quello che intende controllare.

[1] http://wiki.virtualsquare.org/wiki/index.php/PureLibc

2.2.3 Core

La libreria costituisce uno strato intermedio di compatibilità tra netlink e le specifiche configurazioni degli stack.
Vediamo come è composta:

VnsLib: Il primo stato si occupa di inizializzare la libreria in base allo stack che si intende utilizzare, in questo modo possiamo caricare dinamicamente solo il modulo che ci interessa.

Modules: Sono la parte specifica della libreria, essi contengono l'intestazione delle funzioni generiche che si occupano di chiamare quelle particolari per ogni stack.

2.2.4 VnsLib

Espone le interfacce di comunicazione della libreria, che di fatto sono due, una per inizializzare l'ambiente ed una per inviare il pacchetto da gestire.
L'utente non ha accesso ad altre funzioni perchè è la libreria stessa ad analizzare il pacchetto, riempire i campi della struttura generica ed inviarlo al modulo relatvo allo stack.
Di ritorno si riceverà una struttura che permetterà la costruzione del paccehetto netlink di risposta alla recv del programma, questo è l'ultimo compito della nostra libreria.

2.2.5 Moduli

La potenza dei moduli risiede nel fatto che chiunque abbia intenzione di costruire uno stack personalizzato, o di utilizzarne uno per il quale non esiste un modulo, può facilmente realizzare il proprio e la libreria si occuperà di farne il caricamento qualora fosse richiesto. Ogni modulo, infatti, ha la stessa struttura e contiene una funzioni generiche comuni in tutti gli stack. I prototipi di queste funzioni hanno in comune la desinenza che serve a specificare l'azione da compiere, mentre a distinguerle è la radice che prende il nome dello stack, pertanto ogni modulo è composto dallo stesso numero di funzioni che si occupano mediare la comunicazione tra la libreria e il vero stack. Questa mediazione non è particolarmente complessa da comprendere in quanto si tratta di una

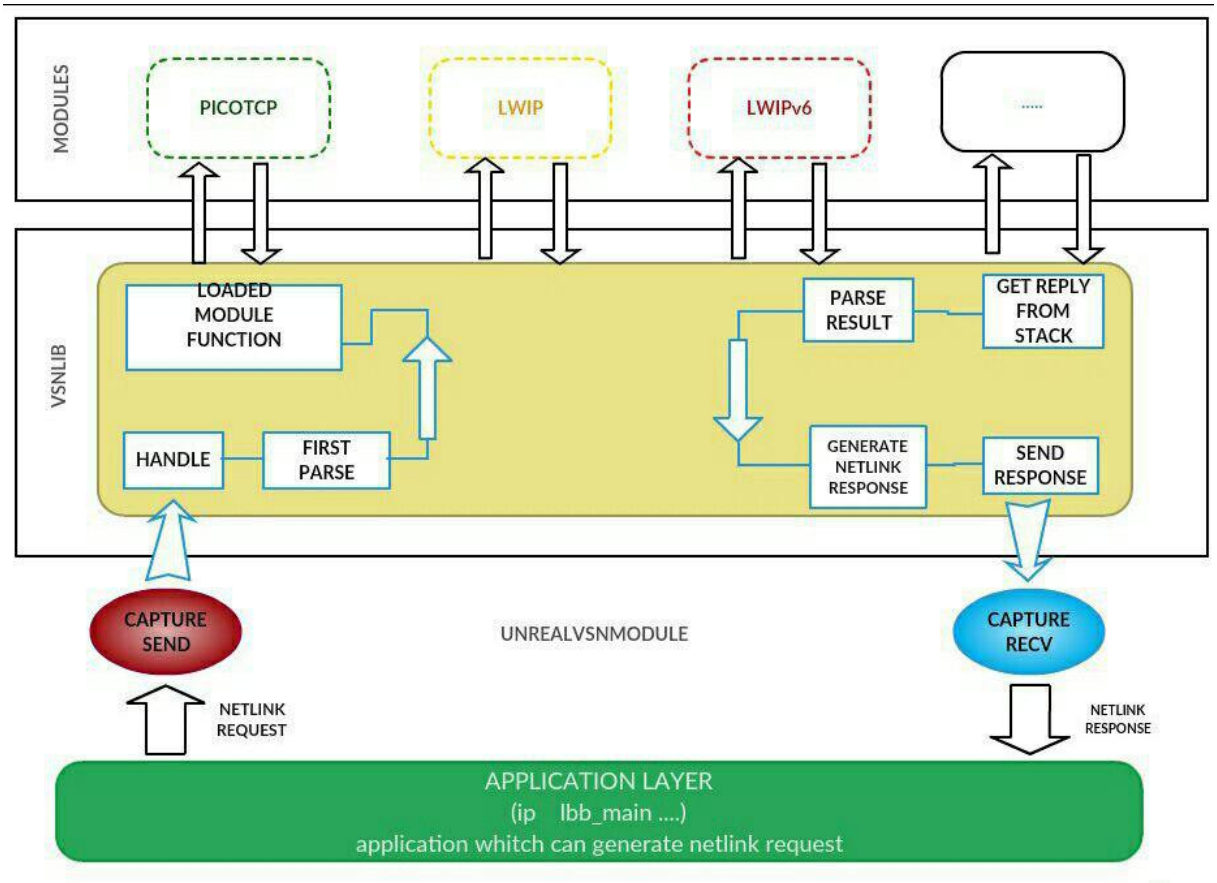

Figura 2.1: mappa concettuale libreria

semplice sequenza di chiamate alle funzioni specifiche dello stack per realizzare l'azione della richiesta.

Per generalizzare la libreria si è usata una tecnica di puntatori a funzione, in questo modo possiamo evitare di specificare il nome completo delle funzioni che, come già accennato si differenziano per nome. Allora l'idea è stata quella di inizializzare, in fase di caricamento della libreria (che avviene con dlopen), proprio un array di puntatori a funzione (attraverso dlsym) in modo tale da poter effettuare le chiamate attraverso di esso. Questo sistema ci permette di caricare dinamicamente solo il modulo di cui si necessita. Ogni funzione riceve lo stesso tipo di struttura e dopo aver raccolto i risultati delle funzioni specifiche dello stack riempie i campi utili a formare la struttura di risposta che poi

sarà ritornata alla libreria per elaborare il pacchetto netlink di risposta.

2.3 Sviluppi Futuri

In fase di progettazione non sono stati posti limiti alla crescita del progetto e la sua modularità è legata principalmente a questo aspetto.
L'augurio è che questo progetto continui a crescere arricchendosi di funzionalità.
In questa fase ci si è occupati delle configurazioni e del routing che deve essere completata e testata prima di poter affermarne la stabilità.
Tra le varie ipotesi di proseguimento c'è quella di gestione del filtering ovvero occuparsi di tutta la parte riguardante le iptables.

Capitolo 3

Casi d'uso

Di seguito alcuni esempi di utilizzo della libreria. Le dimostrazioni seguenti sono state effettuate su una macchina con architettura amd64 e con installato il sistema operativo debian in versione sid (quindi unstable) ma sono state testate e riprodotteanche su altre configurazioni e distribuzioni GNU/Linux.

3.1 Environment

Come precedentemente accennato possiamo integrare la nostra libreria in vuos attraverso il modulo unrealvsnlib o creandone uno alternativo che dipenda da questa libreria.

Creazione moduli: La creazione di un modulo è molto semplice e basta serguire le linee guida di quelli preesisteni, inoltre grazie ad una routine in cmake non è necessario aggiungere alcuna riga manualmente al makefile e sarà, appunto, cmake ad occuparsi della compilazione.

Start Vuos: Vuos si avvia attaverso il comando umvu ed è in grado di interpretare comandi successivi come argomenti, come mostrato nell'esempio infatti umvu è affiancato dal comando per attivare il programma xterm e pertanto verrà lanciata una nuova istanza di terminale con ambiente vuos. Nel caso specifico, per motivi grafici è stato utilizzato il programma konsole.

Include dei Moduli vuos I moduli in vuos vengono inclusi attraverso la direttiva

```
vu_insmode <nome_modulo>
```

Un esempio di avvio e inclusione del modulo è quello della figura seguente.

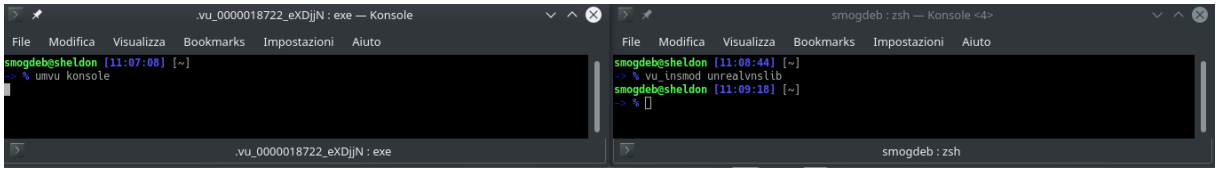

Figura 3.1: start and include vuos

3.2 Esempi

Conclusioni

L'utilizzo di queste tipologie di stack dipende strettamente dalla diffusione del protocollo IPv6 e dei sistemi embedded e sembra essere la direzione in cui questo settore sta muovendosi. La ricerca e lo sviluppo di questi meccanismi sono necessari ad ottenere un sistema performante e stabile quando IPv6 sarà l'ordinario, vanno poi considerati anche i vantaggi conseguenti nelle reti virtuali che già ora vengono utilizzate per la sperimentazione.

L'IoTh ne è un'astrazione più forte In queste conclusioni voglio fare un riferimento alla bibliografia: questo è il mio riferimento [**?**, **?**].

Appendice A

Prima Appendice

In questa Appendice non si è utilizzato il comando:
`\clearpage{\pagestyle{empty}\cleardoublepage}`, ed infatti l'ultima pagina 8 ha l'intestazione con il numero di pagina in alto.

Appendice B

Seconda Appendice

Bibliografia

[1] Renzo Davoli. Internet of Threads. http://www.cs.unibo.it/~renzo/papers/2013.iciw.pdf, 2013.

[2] Renzo Davoli. Internet of Threads: Processes as Internet Nodes. http://www.cs.unibo.it/~renzo/papers/2014.IntTechIoTh.pdf, 2014.

[3] Renzo Davoli. Internet of Threads. http://www.cs.unibo.it/~renzo/papers/ConfGARR11_SelectedPapers_Davoli.pdf.

[4] Altran. picoTCP. https://github.com/tass-belgium/picotcp.

[5] Virtual Square Team. vuos. https://github.com/virtualsquare/vuos.

[6] Virtual Square Team. LWIPv6. http://wiki.v2.cs.unibo.it/wiki/index.php/LWIPV6.

[7] Virtual Square Team. UMview. http://wiki.v2.cs.unibo.it/wiki/index.php/UMview.

[8] Virtual Square Team. purelibc. http://wiki.virtualsquare.org/wiki/index.php/PureLibc.

[9] NetLink associazione LUGMan (Linux Users Group Mantova). http://lugman.org/images/4/43/NetLink.pdf, 2009.

[10] Thomas Graf. https://www.infradead.org/~tgr/libnl/.

[11] Thomas Graf. https://github.com/tgraf/libnl.

[12] Thomas Haller. https://github.com/thom311/libnl.

[13] LWIP. https://savannah.nongnu.org/projects/lwip/.

Ringraziamenti

Qui possiamo ringraziare il mondo intero!!!!!!!!!!
Ovviamente solo se uno vuole, non è obbligatorio.