# SpiderpigOS

Davide Gessa

04-03-2010

# Indice

# Elenco delle tabelle

# Capitolo 1

# Introduzione

SpiderpigOS nasce con l'idea di creare un progetto che permettesse di ampliare le mie conoscenze informatiche riguardo i processori (ed in generale l'hardware) ed i sistemi operativi, e per produrre poi un progetto che riguardasse il programma svolto nel triennio dell'indirizzo informatica abacus, da poter poi eventualmente presentare all'esame. Il progetto proseguira' anche successivamente, sara' possibile vederne lo sviluppo sul sito spiderpig.osdev.it

## 1.1 Licenza

Il proggetto e' rilasciato interamente sotto licenza GPLv3, e' riportato qui di seguito l'header presente in ogni file sorgente del progetto:

```
SpiderpigOS Copyright (C) 2009 - 2010 Davide Gessa
   This program is free software:  you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation, either version 3 of the License, or (at
your option) any later version.
   This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License
for more details.
   You should have received a copy of the GNU General Public License
along with this program.  If not, see <http://www.gnu.org/licenses/>.
```

## 1.2 Strumenti software

Ecco una lista dei software principali utilizzati per realizzare il progetto (compilatori, ide, emulatori). Da sottolineare che tutti sono free software e opensource, e che lo sviluppo e' avvenuto coi sistemi gnu/linux gentoo, gnu/freebsd gentoo e haiku-os, anch'essi free e open.

**gcc** compilatore per il linguaggio C

**make** sistema per facilitare la compilazione

**nasm** compilatore per il linguaggio assembler

**grub** bootloader

**qemu** emulatore di computer per varie architetture

**bochs** emulatore di computer x86

**geany** C ide

**winefish** latex ide

**subversion** controllo di revisione

**latex** compilatore per il linguaggio LaTeX

## 1.3 LaTeX

Per scrivere la documentazione del sistema e' stato utilizzato il linguaggio di markup LaTeX, che permette di preparare dei testi basati su TeX, un linguaggio di composizione tipografica. Utilizzare LaTeX, permette di risparmiare un tempo notevole per quanto riguarda la formattazione delle pagine, la creazione degli indici, la visualizzazione di formule matematiche e molto altro, e per questo motivo e' utilizzato da gran parte di accademici, scienziati, matematici e ingegneri. LaTeXe' distribuito come software libero ed e' disponibile su molte piattaforme.

# Capitolo 2

# Bootloader

Il bootloader e' il software che ha il compito di caricare il kernel del sistema da un unita' di archiviazione (come cd, harddisk, floppy, etc) alla memoria centrale della macchina, e succesivamente di mandarlo in esecuzione. Non essendo il bootloader lo scopo del progetto, ho deciso di utilizzarne uno gia' esistente, e la mia scelta e' ricaduta su grub.

## 2.1 Grub

Grub e' uno dei bootloader piu' utilizzati nell'ambito Linux e *BSD. Grazie alla sua flessibilita', e' possibile utilizzarlo su molteplici dispositivi (floppy, cd, usb), e con molteplici opzioni; e' possibile configurarlo semplicemente modificando un file di configurazione.

#### Vesa Bios Extension

Con l'utilizzo di una patch chiamata vbegrub, grub introduce delle funzioni che permettono di impostare una risoluzione grafica del vbe e successivamente caricare il sistema. Questa patch e' stata utile in SpiderpigOS in quanto ha permesso di implementare una semplice gui, senza dover scrivere un driver per la grafica vesa, o senza dover creare un processo per l'emulazione della modalita' reale, che quindi permetteva di utilizzare gli interrupt e cambiare modalita' grafica.

# Capitolo 3

# Kernel

Il kernel e' la parte fondamentale del sistema operativo: esso infatti si occupa in genere, di gestire la memoria, i task, i file system, il networking, e tanto altro; le funzioni del kernel pero' variano considerevolmente a seconda del modello di riferimento.

## 3.1 Modello di kernel

Uno dei primi problemi all'avvio della realizzazione di SpiderpigOS, e' stata la scelta del modello di kernel da utilizzare; i modelli piu' usati sono il micro kernel, il kernel monolitico ed il kernel ibrido:

#### Kernel Monolitico

Kernel contenente al suo interno la maggior parte delle features implementate nel sistema. L'inconveniente e' che se e' presente un errore al suo interno, compromette il funzionamento dell'intero sistema.

#### Microkernel

Kernel con funzionalita' molto ridotte, in genere offre solo un interfaccia base per l'hardware, e le funzionalita' come filesystem, audio, network sono offerte da dei server avviati a livello utente. Il lato positivo di questo modello di kernel e' che se una parte del sistema crasha (filesystem, audio o altro) puo' essere riavviata senza compromettere il funzionamento dell'intero sistema.

#### Kernel Ibrido

E' una via di mezzo tra microkernel e kernel monolitco: in genere il kernel offre un interfaccia per l'hardware e il livello utente, gestione di task, filesystem, etc, mentre a livello utente i server offrono ai programmi le varie funzionalita' piu' ad alto livello del sistema.

Questa descrizione e' un po' imprecisa, dato che da sistema a sistema, gli sviluppatori fanno scelte diverse, pur avendo come riferimento uno dei modelli base.

La scelta per questo sistema e' ricaduta sul kernel di tipo ibrido: al momento pero' tutte le funzionalita' del sistema sono implementate all'interno del kernel in quanto non e' ancora pienamente supportato il caricamento di programmi esterni, ma appena il supporto sara' soddisfacente, la maggior parte delle funzionalita' ora presenti a livello kernel saranno spostate a livello utente sotto forma di applicazioni server.

## 3.2 Compilazione

Il sistema di compilazione del sistema, ha un interfaccia in linguaggio python che permette di scegliere i componenti da compilare all'interno del kernel; i componenti disponibili sono registrati nello stesso file python in una struttura contenente le dipendenze, i define da dichiarare, i file oggetto da creare, i flags da passare al compilatore, le architetture supportate, etc; il programma inoltre permette di scegliere l'architettura da compilare e la lingua per l'output dei messaggi del kernel. E' anche possibile utilizzare dei preset di configurazione presalvati o gia' disponibili con i sorgenti nella sottocartella cartella config. Il programma e' avviabile tramite il commando seguente, eseguito nella cartella src/kernel:

```
python kselect.py
```

Dopo che lo script avra' creato i file di configurazione e creato i link simbolici necessari, e' possibile utilizzare make per compilare l'intero kernel. Make produrra' un file binario chiamato kernel, che puo' essere eseguito dal bootloader.

## 3.3 Astrazione hardware

Nel kernel e' presente un sistema che permette di implementare diverse architetture senza apportare grandi modifiche. Ogni architettura dovra' offrire l'implementazione delle funzioni standard stabilite nell'header arch.h, dei driver specifici per far funzionare il sistema, uno script per il linker, ed un implementazione in assembly dell'avvio della funzione principale del kernel.

## 3.4 Gestore della memoria

SpiderpigOS divide la memoria del computer in pagine da 4kb (e' possibile modificare questo valore), e tiene traccia delle pagine occupate, tramite un sistema costituito da una mappa di bit; per registrare la dimensione delle allocazioni effettuate, e' presente una lista che memorizza indirizzo base e numero di byte allocati, per permettere in futuro di liberare la memoria. Successivamente sara' implementata la paginazione hardware per le architetture che la supportano, per migliorare la protezione della memoria tra i vari processi, ed un sistema per lo swapping delle pagine su memoria di archiviazione.

## 3.5 Tasking

SpiderpigOS e' un sistema preemtive multitasking; ad ogni task durante la creazione, viene allocata una struttura contenente le informazioni necessarie,

come indirizzo di memoria, terminale, etc. Durante l'inizializzazione del tasking e' possibile scegliere la funzione che eseguira' il task switching, per ora e' implementato solo il metodo round robin.

## 3.6 Device

## 3.7 Driver

L'implementazione dei driver[1] e' presente una struttura di driver generico, che contiene al suo interno una variabile che ne indica il tipo; a seconda del tipo viene allocata la struttura specifica che contiene i metodi generici di quel tipo. Tramite questo metodo e' possibile quindi implementare filesystem, device driver, e altri contenuti del sistema.

## 3.8 Virtual File System

Durante l'inizializzazione del vfs resenta una struttura a livelli per quanto riguarda le operazioni su file. Il livello piu' alto e' il VFS[2], l'unico livello accessibile all'utente che offre tutte le funzioni tipiche della gestione file e filesystem (apertura, lettura, chiusura, scrittura, mounting, umounting, etc) indipendentemente dal filesystem e dall'hardware che viene utilizzato.

| VFS |
|---|
| File System |
| Device driver |
| Device |

Tabella 3.1: Livelli per le operazioni sui fs

Il VFS e' una struttura di nodi ad albero.

| Attributo | Spiegazione |
|---|---|
| Nome | Nome del nodo (univoco in directory) |
| Numero | Numero del nodo (univoco nell'albero) |
| Tipo | Tipo di nodo (device, link, dir, file) |
| Ow | Indica se il nodo e' temporaneo |
| Next, Prev | Puntatori a prossimo e precedente nodo |
| Parent, Childs | Puntatori a padre e figli del nodo |

Tabella 3.2: Struttura di un nodo

Quando viene richiesto un determinato path, il VFS controlla se il path esiste, poi guarda la strada che deve effettuare per arrivare al path, alloca dei nodi temporanei per custodire le informazioni necessarie e interroga i filesystem interessati.

[1] interfaccia tra hardware e software
[2] Virtual File System

### 3.8.1 DevFS

E' fondamentalmente un filesystem virtuale che permette ai device a cui e' assocciato un driver, di essere trattati come dei file. Le informazioni dei device e dei driver risiedono in dei nodi allocati nella struttura principale del vfs, che possono essere eliminati solamente nel caso il device non sia piu' disponibile; i programmi di interfacciamento utilizzeranno questi file per utilizzare l'hardware a livello utente, anche se sono presenti anche delle syscall per ottenere il controllo di un device.

### 3.8.2 TmpFS

Utilizza la ram come device di archiviazione.

### 3.8.3 Initramfs

Sfrutta la funzionalita' del bootloader che permette di caricare in ram un file.

## 3.9 Networking

Per quanto riguarda le funzioni di rete e' stato implementato uno stack di protocolli seguendo il modello gerarchico tcp-ip.

| Livello | Protocolli implementati |
|---|---|
| Trasporto | Tcp, Udp |
| Rete | Icmp, Ip (Ipv4) |
| Link | PPP, Eth, Arp |

Tabella 3.3: Stack dei protocolli

Ogni interfaccia di rete possiede una sua struttura, dove viene indicato il nome, il device, il relativo driver e la struttura con le funzioni e le informazioni di gestione del livello di datalink.

Un device che riceve un pacchetto, lo passa al corrispettivo livello di datalink (per esempio, se e' un modem lo passa al protocollo ppp, se e' una scheda ethernet lo passa al protocollo eth). Questo livello si occupera' di togliere il suo header e di mandare il pacchetto al gestore del livello di rete a cui il pacchetto e' destinato (ad esempio, se e' un pacchetto icmp, al protocollo icmp); cosi' via, anche il protocollo a livello di rete togliera' il suo header ricavando le informazioni necesarie per indirizzare il pacchetto al protocollo destinato del livello superiore.

# Capitolo 4

# Applicazioni e toolchain

Il sistema e' corredato di qualche applicazione, del toolchain di compilazione e delle librerie che si e' deciso di implementare. Fondamentalmente il sistema a livello utente mira a essere compatibile allo standard posix, consentendo quindi di compilare senza apportare modifiche applicazioni posix.

## 4.1 Toolchain

Il toolchain e' un insieme di strumenti che ci permettono di compilare applicazioni per spiderpig da un altra piattaforma.

## 4.2 Applicazioni native

C'e' pero' un set di applicazioni native del sistema:

### 4.2.1 Screenpig

Screenpig e' un gestore di finestre; fondamentalmente mira a diventare il server per l'interfaccia grafica standard del sistema. Per ora e' possibile provare una bozza compilando il programma all'interno del kernel.

## 4.3 Porting

Sarebbe un po' inutile pero', scrivere tutto il software interamente da zero. Come abbiamo gia' detto la compatibilita' con lo standard posix semplifica il porting di applicazioni gia' esistenti per sistemi operativi di tipo *nix. Tra i migliori porting in lista d'attesa abbiamo:

**bash** shell

**make** sistema per facilitare la compilazione

**nasm** compilatore per il linguaggio assembler

**nano** editor di testo da riga di commando

**libncurses** libreria per la grafica ncurses

**subversion** controllo di revisione

**links** browser testuale che utilizza ncurses

# Capitolo 5

# Prestazioni

In questo capitolo trattero' le prestazioni del sistema in esecuzione, effettuando dei confronti con altri sistemi operativi. Prima di tutto propongo una tabella coi limiti del sistema:

| Caratteristica | Quantita' |
|---|---|
| File aperti | 32.000 |
| FS montati in contemporanea | 0 |
| Task in esecuzione | 0 |
| Numero massimo allocazioni | 4096 |
| Driver caricati | 0 |

Tabella 5.1: Prestazioni generiche

# Bibliografia


[1] Intel Manuals

[2] http://www.osdev.it

[3] http://wiki.unix-below.net/doku.php?id=osdev

[4] http://www.osdev.org