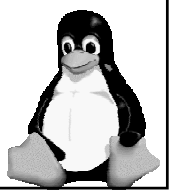
LINUX

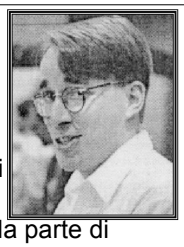


Guarino G. , Valiante G. – 2004/2005

Indice Argomenti

- **Introduzione**
- **Linux : Storia e Obiettivi**
- **Sistema Linux , Nucleo e Distribuzioni**
- **Principi di Progettazione**
- **Moduli**
- **Gestione Processi**
- **Scheduling**
- **Gestione della Memoria**
- **File System**
- **I/O**
- **Comunicazione tra Processi**
- **Strutture di Rete**
- **Sicurezza**

- **Introduzione**



...un pò di definizioni di rito:

Linux:

- è un sistema Operativo ispirato a UNIX da cui prende molti principi di progettazione, che ha avuto e continua ad avere consensi sia da parte di utenti che di aziende.
- Fin dagli inizi dello sviluppo il codice sorgente è stato gratuitamente disponibile... ..in seguito parleremo di licenze.

Linux : Storia e Obiettivi

Storia: I primi passi...

• '91 Linus Torvalds scrive un nucleo autosufficiente come alternativa al sistema operativo Minix.

•Reso disponibile tramite internet , grazie al lavoro di diversi utenti il nucleo è cresciuto fino ad includere numerosi servizi dello UNIX.

E gli Obiettivi ?

..progetto universitario come alternativa a Minix.

..oppure dare all'utente una possibilità di scelta....

http://www.linux.org/docs/index.html

Storia: Dietro le quinte

Per comprendere in realtà la natura di linux non si possono evitare le motivazioni o movimenti che hanno influenzato o sono alla base di tale sistema:

- **"Software Libero" e "Open Source" : tra filosofia e tecnica.**
- **Free Software Foundation.**
- **What's GNU? GNU's not Unix .**

http://www.linux.it

"Software Libero" e "Open Source" tra filosofia e tecnica

"L'espressione software libero si riferisce alla libertà dell'utente di eseguire , copiare , distribuire , studiare , cambiare e migliorare il software".

<< Free speech, not free Beer >>

Non è una tecnica è una questione di libertà:

- **Libertà di eseguire il programma , per qualsiasi scopo.**
- **Libertà di studiare come funziona il programma e adattarlo alle proprie necessità (l'accesso al codice sorgente ne è un prerequisito)**
- **Libertà di ridistribuire copie**
- **Libertà di migliorare il programma e distribuirne pubblicamente i miglioramenti,in modo tale che tutta la comunità ne tragga beneficio.**

Come fare legalmente?..."permesso d'autore": Copyleft vs Copyright

http://www.linux.it

Licenze

•Il Progetto GNU si è affidato al "Permesso d'autore" per proteggere le libertà del "software Libero"

•Il Kernel di Linux è regolamentato in accordo alla GNU General Public License (GPL) stabilita dalla FSF.

•Principi: Il sistema Linux non è di pubblico dominio, questo implica la rinuncia ai diritti d'autore, tuttovia il sistema Linux è *libero* quindi è possibile copiarlo , modificarlo e usarlo in qualunque modo (..e distribuirlo).

•..Quindi chiunque usi un sistema coperto da licenza GPL non può reclamare diritti di proprietà su un prodotto da esso derivato.

•..Inoltre i programmi sotto licenza GPL non si possono fornire solamente in formato binario ma si deve rendere disponibile il codice sorgente.

- **"Software Libero"e"Open Source" : filosofia e tecnica**

Mentre il "Software Libero" si propone come movimento di carattere sociale, l' "Open Source" si avvicina ad una metodologia di sviluppo

Anche se la differenze tra le due sembra netta non bisogna confondere la definizione di "Open Source" con:"puoi guardare il codice sorgente",bensì si avvicina (la definizione ufficiale di Software Open Source) alla definizione di Software Libero.

- **Free Software Foundation**

La FSF si occupa di eliminare le restrizioni sulla copia, sulla redistribuzione , sulla comprensione e sulla modifica di programmi per computer.Operano promuovendo l'uso di Software libero e contribuendo allo sviluppo del sistema operativo GNU.La FSF è finanziata dalla distribuzione di copie di software GNU e manuali e accetta donazioni per sostenere lo sviluppo di GNU.

- **What's GNU? GNU's not Unix .**

R.Stallman: "GNU è il nome del sistema software completo e UNIX-compatibile" che rientra nella definizione di "Software Libero", nato dall'esigenza di non essere legati(come era negli anni '80) al software proprietario.

Sviluppato con l'aiuto di aziende (denaro e macchine) e privati (programmi e lavoro).

•...e LINUX??

Entro gli anni 90 il progetto GNU era completo, tutto era stato scritto o trovato tranne un componente: il Kernel.

Quindi combinare Linux con il sistema GNU quasi completo ebbe come risultato un sistema Operativo completo, che è quello in uso

Quindi riferendosi al sistema operativo chiamandolo "Linux" in realtà è un errore.Una definizione più completa sarebbe GNU/LINUX.

Storia: ...i passi successivi

• il numero di utenti Linux (alle prime versioni di grado ALPHA) erano qualche dozzina..

•...uno sviluppo non controllato e non centralizzato ha portato ad avere un sistema poco manegevole e di difficile manutenzione...

•...La conseguenza di questo "disagio" diede origine all'idea di DISTRIBUZIONE

Strorie di..

- **Nucleo Linux**
- **Sistema Linux**
- **Distribuzioni**

Nucleo del Sistema Linux

- **La prima versione del nucleo (v0.01 del 14/05/1991):**
  1. Funzionava solo con CPU Intel 80386-compatibili.
  2. Nussun Servizio di rete.
  3. Gestione driver limitata.
  4. Filesystem Mimix

- **Kernel 1.0 (14/03/1994) ...dopo 3 anni di sviluppo.**
  1. Incorporava Protocolli Standard dello UNIX
  2. Gestione dei Servizi di rete.
  3. TCP/IP
  4. Interfaccia a Socket per la programmazione su rete
  5. Nuovo Filesystem
  6. Paginazione con file d'avvicendamento (swapping)
  7. Accesso a numerosi dispositivi(CD-rom,Schede audio, etc)
  8. Simulazione di operazioni in virgola mobile
  9. Comunicazione tra Processi (IPC- *interprocess communication*)

Convenzione di Numerazione: nuclei con numero dispari sono "Nuclei di Sviluppo", nuclei con numero dispari sono "Nuclei di Produzione".

Nucleo del Sistema Linux (2)

- **Kernel 1.1**
  apporta numerose correzioni di errori della versione 1.0

- **Kernel 1.2 (03/1995) ...non rivoluzionario come l'1.0 ma:**
  1. Gestiva un'ampia gamma di dispositivi (compreso bus PCI)
  2. Permetteva di simulare il DOS per PC
  3. Protocollo rete Aggiornato (Permetteva l'uso di IPX)
  4. Include Servizi Contabilizzazione risorse (Accounting)
  5. Servizi di Sicurezza (Firewall)
- **Kernel 2.0 (06/1996)**
  1. Gestiva Architetture diverse e più unità di elaborazione
  2. Miglioramenti nella gestione della memoria
  3. Miglioramenti nella gestione dei protocolli TCP/IP
  4. Gestione dei Moduli (con caricamento Automatico) e thread
  5. Configurazione dimanica del nucleo
- **Kernel 2.2 (01/1999) Affinamento servizi e dispositivi**

Convenzione di Numerazione:il Cambiamento dalla versione 1 alla 2 era giustificato dall'introduzione di caratteristiche importanti

Nucleo del Sistema Linux (3)

- **Kernel 2.4**

1. Aumento limiti (N.di CPU, memoria RAM max, grandezza file, ecc.)
2. Supporto per processori a 64 bit.
3. Miglioramenti generali (NFS, TCP/IP ecc.).
4. Supporto nuovi FS.
5. Supporto nuove schede video, sonore, rete, ecc.
6. Supporto unità USB.
7. Adattamenti per nuove versioni del compilatore

- **Kernel 2.6**

1. introduzione di un kernel preemtive ("concedere" il controllo della CPU a un normale processo in modalità utente)
2. Migliorare diversi supporti
3. Sistema di gestione per il risparmio energetico.
4. Migliorare supporto per architetture per i dispositivi embedded e quelle server
5. Nuove librerie per il multithreading
6. possibilità di creare oltre 4 miliardi di utenti

Sistema Linux

- Il Nucleo è la parte centrale del sistema ma è affiancato da numerosi componenti che rendono completo il Sistema Operativo.
- Tali componenti possono essere specifici per Linux , ma molti sono componenti concepiti per sistemi UNIX-like e quindi li ritroviamo in altri sistemi come: BSD Berkley (e FreeBSD), X Window del MIT e progetto GNU della FSF
- Problema della Comunità Linux: mantenere l'integrità del sistema sulla rete.
  FSHS (File System Hierarchy Standard): curato dalla comunità Linux al fine di preservare una compatibilità fra le varie parti del sistema.

  Specifica la struttura di un file system Linux stabilendo sotto quali nomi di directory i file di configurazione dovrebbero essere archiviati

Distribuzioni

- **In Principio fu la SLS..**
  la prima raccolta di pacchetti linux che possiamo definire Distribuzione

- Alla Pagina di "Linux Weekly News Distribuition" Possiamo trovare un elenco di 425(al momento della stesura) tra distribuzioni e varianti Di Linux.
- Problema: "SO Troppo Aggiornabile" e questo scoraggia gli sviluppatori di Software. Progetto LSB (Linux Standard Base).

- **Debian GNU/Linux** http://www.debian.org/
- **Fedora** http://fedora.redhat.com/
- **Fedora Legacy** http://www.fedoralegacy.org/
- **Gentoo Linux** http://www.gentoo.org/
- **Mandrakelinux** http://www.mandrakelinux.com/
- **Red Hat Enterprise** http://www.redhat.com/
- **Slackware Linux** http://www.slackware.com/
- **SuSE Linux** http://www.suse.com/ Unofficial SuSE FAQ: http://susefaq.sourceforge.net/
- **Turbolinux** http://www.turbolinux.com/

Elenco Distribuzioni: **www.lwn.net** *– Progetto LSB:* ***www.linuxbase.org***

Indice Argomenti

Linux – Principi di Porgettazione

Sistema molto simile a UNIX:
• Sistema Multiutente a più processi (con strumenti compatibili UNIX)

Principi Principali di Progettazione:
• Velocità
• Efficienza
• Standardizzazione

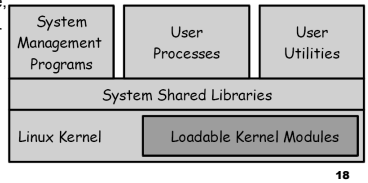
Rapporti con UNIX
• Segue Specifiche POSIX
• API basata su Unix SVR4
• Molti tool e librerie basati su BSD

Linux è un sistema in continua evoluzione che gestisce una vastissima gamma di architetture... Ma basterebbero meno di 4MB di RAM per utilizzarlo

Componenti Del Sistema

Composto principalmente da tre blocchi (seguendo la tradizione UNIX).

- **Nucleo :** Parte Centrale del sistema, responsabile delle astrazioni del SO, fornisce servizi necessari per eseguire e gestire processi e risorse.
- **Librerie di Sistema:** Definiscono uno standard di funzioni con le quali le applicazioni interagiscono col nucleo
- **Utilità di Sistema:** Programmi che eseguono compiti specializzati di gestione alcuni di configurazione, altri come i demoni per la gestione di richieste Che possono rimanere permanentemente in esecuzione.



Componenti Del Sistema (2)

Modalità di accesso alla CPU

- Modo Utente
- Modo Nucleo

Modo nucleo eseguito esclusivamente in forma privilegiata (fino al kernel 2.6)

Basato sul modello storico di Unix dove il nucleo è un unico blocco monolitico di codice binario, per migliorare le perstazioni.

Spazio di indirizzi unico per:

- Codice Nucleo
- Driver dispositivi
- File System
- Codice relativo alla gestione della rete

Tutto Insieme?...e la modularità?

Il nucleo può caricare e scaricare dinamicamente moduli durante l'esecuzione

Componenti Del Sistema (3)

Funzionamento comunicazione nucleo – applicazioni:

Le applicazioni non interagiscono direttamente con il sistema operativo ma con un interfaccia:

1. Applicazioni interagiscono con le librerie di sistema
2. Le librerie si chiamano dei servizi offerti dal nucleo
3. Una chiamata di sistema provoca il cambio di modalità da utente a nucleo.
4. Oppure chiamano funzioni più complesse delle chiamate di sistema codificate nelle librerie di sistema

...E poi ci sono i "Programmi D'utilità di Sistema o di utente" :

- Programmi inizializzazione sistema
- Configurazione della rete
- Server

Inizializzazione del Sistema

Linux può utilizzare l'inizializzazione del sistema in stile System V(SysV) oppure BSD.

❑ BSD

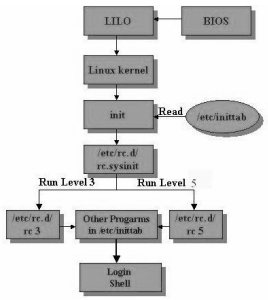
utilizza alcuni grossi script che gestiscono tutta l'inizializzazione del sistema (utilizzato soltanto dalle distribuzioni Slackware).

❑ SysV

utilizza livelli di esecuzione e un gruppo di script di inizializzazione che vengono eseguiti per avviare e arrestare i daemon (processi in background) a seconda del livello di esecuzione.Per ogni Daemon viene mantenuto uno script in una directory centralizzata.

Init: tutto inizia da Qui

- All'accensione del sistema, all'avvio del boot , il kernel si avvia ,viene caricato nella memoria ed Eegue un programma:Init.
- Questo programma è responsabile di tutto il resto, identificato come l'origine di tutti i processi.
- Init legge la configurazione del sistema da un file: *inittab*(tabella di inizializzazione).
- Il file inittab descrive in che modo init imposta il sistema.



Livelli di Esecuzione di System V

- I Livelli di esecuzione di SysV vanno da 0 a 6.in realtà ci sono anche 7 e 9.
- Ciascun livello di esecuzione corrisponde ad una modalità operativa.
- Alcuni livelli molto spesso non vengono utilizzati, (utilizzati quelli da 2 a 5)
- Garantisce una flessibilità maggiore rispetto all'inizializzazione di tipo BSD.

Livelli di Esecuzione (Solitamente abbiamo in inittab id:5:initdefault:)

- ✓ Liv 0 – Halt (raccomandato non utilizazzare questo livello come initdefault)
- ✓ Liv 1 – Single User Mode
- ✓ Liv 2 – Multiuser (senza nfs)
- ✓ Liv 3 – Fully Multiuser Mode
- ✓ Liv 4 – Unused
- ✓ Liv 5 – X11
- ✓ Liv 6 - reboot (raccomandato non utilizazzare questo livello come initdefault)

Moduli

Un modulo è un file oggetto, cioè un codice eseguibile che fa riferimento a funzioni e variabili esterne, oltre a dichiarare funzioni e variabili che lui stesso definisce.

Funzionamento:

Normalmente un file oggeto viene salvato sul disco con estensione ".o". Durante la compilazione di applicazioni i file oggetto generati da ciascun file sorgente vengono linkati in un unico file eseguibile....

...In realtà qundo si compilano moduli per il kernel i file oggetto non vengono linkati perchè i riferimenti estreni del modulo si riferiscono a simboli (funzioni, variabili) che fanno parte del kernel e non possono essere risolti nello spazio utente.

Cosa rappresenta un modulo?

Driver di dispositivo , file system , protocollo di comunicazione.

Come abbiamo visto il kernel quindi può essere visto come un unico blocco monolitico,un file eseguibile generato dal collegamento dei file oggetto.

Moduli (2)

Nucleo e Moduli: I moduli possono essere:
- Compilati
- Caricati/Scaricati

in modo indipendente dal resto del Kernel

Tre componenti formano il supporto di Linux per la Gestione dei Moduli
1. Gestione dei Moduli
2. Registrazione dei Driver
3. Risoluzione dei Conflitti

Problemi della Modularizzazione:

- Potrebbe causare problemi di sicurezza [Soluzione? Disattiva *CONFIG_MODULES*]
- "Race Condition" per l'eliminazione di un modulo kernel [*kernel/module.c*]si può non permettere di eliminare un modulo una volta caricato [*CONFIG_MODULE_UNLOAD*]
- Compatibilita/Incompatibilità tra le versioni differenti del Kernel

Moduli (3)

Vantaggi della Modularizzazione
•Moduli Modificabili Instantaneamente: Rimuovere, Ricompilare e Ricaricare
•Risparmiare Memoria Caricando/Scaricando "*a tempo debito*" senza ricompilare

"Riavviare un grande sistema multiutente potrebbe essere molto costoso, comportando l'estromissione temporanea di tutti gli utenti"

Ricompilare o non ricompilare...questo è il problema

Ricompilare il kernel è una soluzione accettabile quando viene sostituito o aggiunto dell'hardware (o altro),cioè quando vengono effettuati cambiamenti "permanenti".

La compilazione del kernel è richiesta di rado grazie ai moduli caricabili

I moduli vengono memorizzati sotto la directory "*/lib/modules*"

Gestione dei Moduli

Il sistema verifica che i riferimenti a simboli e i punti d'ingresso del nucleo (devono puntare a locazioni all'interno dello spazio d'indirizzi del nucleo)
- Gestione di sezioni di codice del modulo nella memoria del nucleo
- Gestione dei simboli ai quali il modulo può fare riferimento

Scenario:

Linux mantiene una tabella interna dei simboli nel nucleo (solo quelli esplicitamente esportati dal nucelo)

L'insieme dei simboli rappresenta l'interfaccia con la quale il modulo può interagire con il nucleo

Quando un modulo viene caricato un programma d'utilità di sistema lo esamina per rilevare riferimenti irrisolti.(cercandoli nella tabella simboli del nucleo)

Processo per il caricamento di un modulo:

1. Caricatore: richiede al nucleo lo spazio di memoria per allocare il modulo
2. Passaggio al nucleo del modulo e della tabella dei simboli esterni da esportare

Meccanismo di Richiesta dei Moduli:
Interfaccia utilizzata dal gestore dei moduli e dal nucleo attraverso la quale il nucleo richiede al gestore l'utilizzo di un Driver(o altro) i cui moduli non sono stati caricati

...e infine

Il Programma di gestione interroga ad intervalli regolari il nucleo per scoprire quali moduli caricati dinamicamente non sono più in uso e quindi scaricarli.

I moduli vengono memorizzati sotto la directory "*/lib/modules*"

Caricamento Moduli...un pò di pratica

Caricamento Manuale

Vengono solitamente impiegate 4 Utility per la gestione manuale dei moduli

• **modprobe** – Carica o inserisce moduli nel kernel dopo aver controllato le dipendenze in "*/lib/modules/<kernel version>/modules.dep*".(usa **insmod**)

• **insmod** – Carica o inserisce moduli nel kernel senza controllare le dipendenze.

• **rmmod** – Rimuove moduli attualmente caricati.

• **lsmod** – Elenca i moduli attualmente caricati.

Caricamento Automatico

Il caricamento viene gestito in modo leggermente diverso dalle diverse distribuzioni viene utilizzato *"/etc/modules.conf"*

Esempio: Metodi di OpenLinux di Caldera

•Moduli vengono caricati mediante righe di comando aggiunte al file *"/etc/rc.d/rc.local"*

•Il metodo **kmod** carica i moduli su richiesta (ma non li scarica automaticamente).

•I moduli vengono caricati da un elenco in *"/etc/modules"*mediante script (*rc.modules*)

Registrazione dei Driver

Il nucleo mantiene tabelle dinamiche di tutti i driver noti e fornisce un insieme di procedure per aggiungere/rimuovere un driver da queste tabelle. Per ogni nuovo modulo il nucleo avvia la procedura d'inizializzazione.

Un modulo può contenere diversi tipi di Driver secondo diversi tipi presenti nelle *"Tabelle di Registrazione":*
- **Driver dei Dispositivi** – dispositivi a caratteri(stampanti, terminali e mouse), dispositivi a blocchi(unità disco) e dispositivi di interfacciamento alla rete.
- **File System** – entità che comprende le procedure per un file system virtuale
- **Protocolli di Rete** – un modulo può essere una concreta realizzazione di un protocollo di rete.
- **Formato Binario** – Specifica un modo per riconoscere e caricare un nuovo tipo di file eseguibile.

Risoluzione dei Conflitti

Il problema deriva dalla vasta gamma di dispositivi possibili e dalla gestione modulare

Linux fornisce un meccanismo centrale di risoluzione dei conflitti che aiuta a regolare l'accesso a certe risorse fisiche.

Gestione dei conflitti:

• Impedire che moduli diversi entrino in conflitto per l'accesso alle risorse fisiche.

• Impedire che una procedura di autoverifica interferisca con driver gia presenti.

• Risolvere i conflitti che si creano tra diversi driver che tentano di accedere alla stessa risorsa fisica.

Soluzione:

Il nucleo mantiene una lista delle risorse fisiche assegnate, un driver che vuole accedere ad una risorsa deve passare per il nucleo (per prenotare la risorsa) che gestisce i possibili conflitti.

Indice Argomenti

Gestione dei Processi

- Identificatore del Processo
- Credenziali
- Personalità
- Modello fork-exec
- Ambiente di un Processo
- Contesto di un Processo
- Processi e Thread

Gestione dei Processi

Identificatore del Processo

Come abbiamo gia visto: "Un processo è un istanza in esecuzione di un programma". Linux adotta (..anzi DEVE adottare) un modello di processi simile allo Unix...
...ma aggiunge alcune caratteristiche proprie.

• Ogni processo sotto linux è identificato tramite un identificatore unico (*pid* 16-bit)

✓ Linux assegna il *pid* sequenzialmente e automaticamente alla creazione.

La gestione dei Pid è un albero dove ogni processo ha un padre a cui fa riferimento con il *ppid (Parent Process ID)*.

✓ *pid = 1* Assegnato a Init "il Padre di Tutti i Processi". (non termina mai)

In Quasi tutti i sistemi GNU/Linux è presente(/bin/sh) la bash (**B**ourne **A**gain **SH**ell)

• Vedere un Processo attivo: ps

```
% ps
  PID TTY          TIME CMD
21693 pts/8    00:00:00 bash
21694 pts/8    00:00:00 ps
```

• Uccidere un Processo : *kill <pid>*

Credenziali – Identità dei Processi

- Ogni File ha un identificatore utente (UID) e uno o più identificatori di gruppo (GID)
  Comando Shell "id" : per avere informazioni su UID e GID.
- Visualizzare informazioni sui Permessi di un file:

```
% ls -l hello
-rwxr-x---    1 samuel   csl         11734 Jan 22 16:29 hello
```

... e i Processi??... Ogni Processo ha:

- Real User ID: di solito chi esegue il processo (il superuser ID = 0 può cambiarlo)
- Real Group ID
- Effective User ID: Normalmente uguale a Real UID (non se si usa set-user-ID)
- Effective Group ID Normalmente uguale a Real GID (non se si usa set-user-ID)
- Saved set-user-ID
- Saved set-group-ID

Solitamente User ID processo = owner ID file e Group ID processo = Group ID file

Caso Particolare..Sticky bits

- I Permessi di base sono Lettura, Scrittura ed Esecuzione, ma per garantire sicurezza c'è bisogno di gestire in alcuni casi anche la cancellazione : *Sticky bit*.
- Questo bit è applicato solo alle Directory: Una Directory con lo Sticky bit settato permette la cancellazione solo al proprietario del file, non solo avendo il diritto in scrittura. (/tmp)

Personalità

Non fa parte dello Unix, ogni processo ha un identificatore di personalità che può modificare lievemente la semantica di certe chiamate del sistema.

Modello fork-exec

I sistemi DOS e Windows API hanno una famiglia di funzioni *spawn* queste funzioni prendono in input il nome del programma da eseguire e ne creano un istanza.

Linux affronta il problema in modo diverso: offre due famiglie di funzioni fork e exec.

• *fork* crea un processo figlio identico al padre con aree dati separate(copia), il padre e il figlio continuano. L'esecuzione in modo concorrente. La funzione torna 2 volte:

- ✓ Restituisce 0 al rpocesso figlio.
- ✓ Restituisce il pid del figlio al padre

Un processo può avere più figli e non c'è alcuna funzione che può dare al padre il pid dei suoi figli

Il figlio eredita: Uid,Gid,EUid,EGid,SUid,SGid,cwd...e altro

Come si fa a sapere chi finisce prima tra padre e figlio?

Famiglia di SysCall *wait* permettono ad un processo di "aspettare" che un processo figlio termini.(funzione wait aspetta "un figlio", waitpid ne aspetta uno in particolare)

37

Esito della fork

1. Il figlio termina prima del padre ed il padre ottiene il risultato (status) del figlio tramite la wait
2. Il Padre termina prima del figlio, il processo Init diventa il nuovo padre
3. Il figlio termina prima del padre ma il padre non ha chiamato la wait, allora il figlio diventa un processo *Zombie*.

- Famiglia SysCall *exec*

Esegue un programma memorizzato su disco. Differentemente dalla fork il nuovo processo rimpiazza completamente il processo che lo ha chiamato quindi cambiano:

- Codice da eseguire
- Dati
- L'Heap
- Lo Stack

38

La Famiglia delle Funzioni Exec

execl(..),execv(..), execle(..), execve(..),execlp(..),execvp(..);

Differenze:

- L = lista char* arg0,char* arg1, ... , char* argN, char* 0;
- V = vettore char* arg[]
- P = path cerca il file specificato nella variabile d'ambiente *path*
- E = environment setta le variabili d'ambiente, senza la "*e*" il nuovo processo usa le variabili d'ambiente correnti

Ambiente?..ma cos'è?...

E un programma che vuole eseguire un altro programma ma senza essere sostituito? :

- Prog1 esegue una fork
- Il nuovo figlio di Prog1 esegue l'exec

39

Ambiente di un Processo

- L'ambiente di un processo è un insieme di variabili.
- Composto da due vettori: Vettore degli argomenti (arg[ ]) e Vettore d'ambiente.
- L'Ambiente di un processo è ereditato dal genitore.
- Ogni programma ha a disposizione un puntatore che permette di accedere ad una struttura dati con le variabili d'ambiente.(extern char** environ)
- Creando un nuovo processo bisogna creare un nuovo ambiente.
- I vettori d'ambiente e degli argomenti vengono ignorati dal nucleo.

Esempio:

HOME=/home/gaetano

PATH=/bin

SHELL=/bin/sh

USER=gaetano

LOGNAME=gaetano

40

Contesto di un Processo

L'ID di un processo e le proprietà d'ambiente di solito sono stabilite al momento della creazione di un processo e non mutano fino alla terminazione. (un processo può decidere di cambiarle).

Il *CONTESTO di un processo* invece è lo stato del programma in esecuzione in ogni istante (quindi cambia in continuazione).

Tipi di Contesto:

- Contesto di Scheduling
- Contabilizzazione delle risorse
- Tabella dei file
- Contesto del file system
- Tabella dei gestori dei Segnali
- Contesto della Memoria Virtuale

41

Contesto di Scheduling

Sono le informazioni necessarie allo scheduler per sospendere e riavviare il processo,comprende :

- le copie di tutti i registri del processo tranne quelli in virgola mobile(ripristinati solo quando è necessario).
- Informazioni sulla priorità del processo e sui segnali attesi dal processo
- **Pila di nucleo** del processo è una area di memoria nello spazio di indirizzi del nucleo riservata esclusivamente all'uso di codice eseguito nel modo nucleo

Contabilizzazione delle risorse

Il nucleo mantiene informazioni sulle risorse correntemente impiegate da ciascun processo, e anche il conto totale delle risorse utilizzate da un processo.

Tabella dei File

Un vettore di puntatori alle strutture dati del nucleo relative ai file

42

Contesto del File System

Il contesto del File-System riguarda le richieste di apertura di nuovi file (Tabella dei file tiene traccia solo dei file gia aperti).Memorizza il path delle directory da cui iniziare la ricerca.

Tabella dei gestori dei segnali

Questa tabella specifica quali procedure nello spazio di indirizzi del processo si devono invocare quando si riceve un certo segnale.

Contesto della Memoria Virtuale

Descrive i contenuti dello spazio di indirizzi di un processo (vedremo in seguito).

Processi e Thread

Cosa intendiamo per Thread?...

Come abbiamo visto un processo rappresenta l'esecuzione di un unico programma, mentre i thread rappresentano contesti di esecuzione distinti e concorrenti all'interno di un singolo processo.

Due processi hanno due spazi di indirizzi distinti e indipendanti.

I thread condividono il *medesimo* spazio di indirizzi.

...e Linux cosa intende per Thread?...il trucco c'è ma non si vede.

L'implementazione dei thread POSIX nei sistemi GNU/Linux differisce dall'implementazione nei sistemi Unix-like:

I thread sono implementati come processi: si possono usare librerie PThread con:
pthread_create Linux crea un nuovo processo.

Quindi è una fork? NO! La fork crea una copia del contesto per il figlio.

Thread(2)

Il contesto non è contenuto interamente nelle strutture dati interne ad un processo ma mantiene puntatori a strutture esterne...questo garantisce la possibilità a un altro processo di puntare queste strutture.

CLONE VS. FORK

- Si usa la funzione *clone* specificando quali sottocontesti puntare e quali copiare al momento della creazione di un nuovo processo.
- Si può notare: la *fork* è solo una generalizzazione della *clone*.
- La *clone* richiede di specificare la locazione di memoria per lo stack di esecuzione che il nuovo processo utilizzerà.
- Clone non è molto utilizzata. Per creare un nuovo thread si utilizza *pthread_create()*

Quindi...

- Usa fork per creare un nuovo processo
- Usa pthread_create per creare un nuovo thread

Esempio

```
#include <pthread.h>
#include <stdio.h>
#include <unistd.h>

void* thread_function (void* arg)
{
  fprintf (stderr, "child thread pid is %d\n", (int) getpid ());
  /* Spin forever.  */
  while (1);
  return NULL;
}

int main ()
{
  pthread_t thread;
  fprintf (stderr, "main thread pid is %d\n", (int) getpid ());
  pthread_create (&thread, NULL, &thread_function, NULL);
  /* Spin forever.  */
  while (1);
  return 0;
}
```


Sorprese in Esecuzione

```
% ./thread-pid &
[1] 14608
main thread pid is 14608
child thread pid is 14610
% ps x
  PID TTY      STAT   TIME COMMAND
14042 pts/9    S      0:00 bash
14608 pts/9    R      0:01 ./thread-pid
14609 pts/9    S      0:00 ./thread-pid
14610 pts/9    R      0:01 ./thread-pid
14611 pts/9    R      0:00 ps x
% kill 14608
[1]+  Terminated              ./thread-pid
```

E il 14609?

È il "Manager thread" che fa parte dell'implementazione interna dei Thread GNU/Linux, è creato la prima volta che un programma chiama la *pthread_create* per creare un nuovo thread

Lo Standard POSIX

Il documento Posix.1c definisce un interfaccia di programmazione standard per eseguire Thread multipli:

Le librerie Linux realizzano quest'interfaccia in due modi, un applicazione può scegliere di utilizzare:

- Pacchetto realizzazione thread basati su modo utente
  - ✓ Evita ritardi dovuti allo scheduling.
  - ✓ Però tutti i thread sono eseguiti all'interno di un unico processo.
- Pacchetto realizzazione thread basati su modo nucleo
  - ✓ Usa la *clone.*
  - ✓ In questo modo si creano contesti di scheduling multipli.
  - ✓ Permette alle applicazioni di eseguire più Thread contemporaneamente in sistemi con più unità di elaborazione.
  - ✓ I Thread multipli possono eseguire più chiamate del sistema contemporaneamente.

Indice Argomenti

- **Introduzione**
- **Linux : Storia e Obiettivi**
- **Sistema Linux , Nucleo e Distribuzioni**
- **Principi di Progettazione**
- **Moduli**
- **Gestione Processi**
- **Scheduling**
- **Gestione della Memoria**
- **File System**
- **I/O**
- **Comunicazione tra Processi**
- **Strutture di Rete**
- **Sicurezza**

Scheduling

Lo Scheduling nei sistemi Linux non consiste nella semplice esecuzione e sospensione dei processi:

Gestione dei Task del Nucleo

Lo scheduling dei processi delle operazioni del nucleo è diverso dallo scheduling dei processi.

Richiesta di esecuzione codice nel Modo Nucleo:

- Programma in esecuzione richiede un servizio del sistema operativo.
- Un Driver di un dispositivo genera un segnale d'interruzione che esegue il gestore di tale interruzione definito dal nucleo.

Problemi:

Sezioni Critiche

Sincronizzazione del Nucleo

Basato su un meccanismo che impedisce l'interruzione di una sezione critica da parte di un altra sezione critica:

1. Impedire la sospensione del codice di nucleo.
   Prima Tecnica:Ricevuto un segnale di interruzione dal temporizzatore per una procedura di servizio del nucleo,operazione di scheduling non avviene immediatamente, viene impostato il flag *need_resched* del nucleo in modo che lo scheduler sia eseguito al termine della chiamata di sistema.

2. Con del codice nucleo in esecuzione: Garantire che nessun altra parte del codice nucleo sia eseguita finchè non si verifica:
   - ✓ Interruzione
   - ✓ Eccezione di Pagina Mancante
   - ✓ Chiamata allo scheduler da parte dello stesso codice nucleo

Sincronizzazione del Nucleo (2)

1. I Segnali di interruzione sono un problema se contengono sezioni critiche:

   Interruzioni dal temporizzatore: non provocano mai direttamente l'operazione di scheduling ma richiedono che lo scheduling sia eseguito appena possibile.

2. Pagina Mancante:se una procedura del nucleo tenta di accedere alla memoria utente potrebbe generare un'eccezione di pagina mancante.
   Soluzione: Assicurarsi che le sezioni critiche non contengano riferimenti alla memoria utente o attese per il completamento di operazioni di I/O.

Seconda Tecnica

Architettura di controllo delle interruzioni della CPU disabilitando le interruzioni durante l'esecuzione di una sezione critica, così il nucleo si assicura di poter procedere senza rischiare accessi concorrenti a strutture dati condivise

Svantaggio: costoso Abilitare e disabilitare le interruzioni (provoca un calo di prestazioni)

NON è sempre necessario: Linux assicura una sincronizzazione che permette l'esecuzione di lunghe sezioni critiche senza disabilitare le interruzioni

Sincronizzazione e Sezioni critiche

Linux divide le procedure di gestione delle interruzioni in due parti:

Parte Superiore

è un ordinaria procedura di gestione delle interruzioni, ed è eseguita solo dopo la disabilitazione condizionale delle interruzioni:

- ✓ Segnali con priorità più alta possono interrompere la procedura
- ✓ Segnali con priorità uguale o inferiore non possono interrompere la procedura

Parte Inferiore

Eseguita con tutte le interruzioni abilitate, da un minischeduler in grado di assicurare che tutte le parti inferiori non si interrompano mai.Una parte inferiore non può interrompere un altra parte inferiore ma può essere interrotta da una parte superiore.

Meccanismo di disabilitazione selettiva delle parti inferiori

Silberschats cap 20

Scheduling dei Processi

Linux adotta due distinti algoritmi di scheduling:

- Algoritmo a partizione di Tempo.
- Algoritmo per elaborazioni in tempo reale.

Quale utilizzare per un processo è determinato nella classe di scheduling.

Tali classi sono definite nell'estensioni dello standard POSIX all'elaborazione in tempo reale (Posix.1b).

Tre Classi di Scheduling:

1. Timesharing
2. Real-Time FIFO
3. Real-Time Round Robin

Priorità Statica VS Priorità Dinamica

Politiche di Scheduling di Linux

- Time-sharing con quanti di tempo dinamici
- Basato su thread
- Priorità dinamica
  - ✓ Per evitare starvation
  - ✓ Incremento ai processi che hanno eseguito recentemente
  - ✓ Decremento per i processi che hanno eseguito spesso

55

Algoritmo a Partizioni di Tempo (*Timesharing*)

Si basa su un algoritmo a priorità basato sui "crediti":

- Ogni Processo ha un numero di crediti di scheduling.
- La CPU viene assegnata al processo che ha il massimo numero di crediti.
- Ogni volta che arriva un segnale di interruzione il processo perde un credito.
- Quando i crediti del processo scendono a zero il processo viene sospeso.

Se tutti i processi hanno un numero di crediti uguale a zero Linux effettua un'operazione di riassegnamento dei crediti, aggiungendo crediti a TUTTI i processi del sistema (non solo a quelli eseguibili).

Utilizzando la Regola: *crediti = crediti/2 + priorità (Priorità DInamica)*

I Crediti dei processi meno utilizzati salgono in fretta tenendo sempre conto però della priorità

56

Scheduling In Tempo Reale

Come definisce lo standard Posix.1b questo tipo di scheduling è diviso in due classi

- FCFS
- RR

In entrambe i casi si assegna una priorità ai processi. Nello shceduling a partizione di tempo c'è una competizione tra processi, invece nello scheduling in tempo reale lo scheduler esegue sempre il processo con priorità più alta:

FCFS continua l'esecuzione fino alla terminazione o in caso di blocco (esempio I/O).

RR un processo può essere sospeso allo scadere del suo *quanto* di tempo ed essere spostato alla fine della coda di scheduling.

Questo tipo di Scheduling è in tempo reale *debole*(e non stretto) cioè:

Lo scheduler offre rigide garanzie sulle priorità relative ai processi in tempo reale, ma il nucleo non fornisce alcuna garanzia sulla rapidità con la quale un processo in stato di pronto entri effettivamente in stato di esecuzione.

57

Deteminare la lunghezza del "Quanto" di Tempo per RR

- Se Troppo corto, cresce L'overhead
- Se troppo lungo, i processi non sembrano più essere eseguiti in concorrenza
- Sotto Linux, quanti di tempo lunghi non degradano necessariamente il tempo di risposta per i processi interattivi

Un Quanto di Linux può arrivare a 20 "Clock tick":200 millisecondi

58

Problematiche dello Scheduling di Linux (fino kernel 2.4)

- Non è perfettamente Scalabile
  - ✓ Con la crescita del numero dei processi è necessario ricalcolare le priorità dinamiche di tutti i processi
  - ✓ Per questo motivo lo scheduling viene diviso il "epoche" e non ad ogni context switch.
- Un gran numero di processi runnable può rallentare il tempo di risposta
- Il Quanto predefinito è troppo lungo per carichi elevati
- Il supporto per i processi in real-time è debole

59

Scheduling 2.6

Con La Versione del Kernel 2.6, viene introdotto un nuovo scheduler:

- Altamente Scalabile - O(1) utilizza code a doppio puntatore separate per ogni livello di priorità.
- Anche i kernel Thread possono essere ora preempted.
- Miglior supporto per Muliprocessing: strutture dati separate per ogni processore

60

Multielaborazione Simmetrica (SMP)

La Multielaborazione Simmetrica è la capacità di eseguire processi o Thread in parallelo su unità di elaborazione distinte.

Problema: Sincronizzazione del nucleo.

L'SMP di Linux stabilisce che un unica unità di elaborazione possa eseguire codice nel modo nucleo.

Come?

SMP adotta per il nucleo un unico semaforo ad attesa attiva (*spinlock*) questo potrebbe causare un sensibile calo di prestazioni nel caso di processi che richiedono grandi quantità di codice del nucleo. Questo fino al Kernel 2.0

Dal Kernel 2.2 : SMP viene reso più flessibile dividendo il singolo semaforo ad attesa attiva del nucleo in bloccaggi multipli in modo che ognuno di essi protegga un sottoinsieme delle strutture dati del nucleo.

61