

Sviluppo di software di rete con l'utilizzo di sistemi UNIX

Processi demone

Claudio Vicari

Processi demone: introduzione

- ➔ un processo demone (daemon) è un processo eseguito in background, privo di terminale di controllo
- ➔ possiamo volere che un processo sia demone:
 - perché continui a funzionare anche senza controllo dell'utente
 - perché non occupi la risorsa del terminale

Processi demone: introduzione

- ➔ `ps axj`:
 - l'opzione `-x` mostra anche i processi privi di terminale di controllo
 - l'opzione `-j` mostra informazioni sui PID che coinvolgono il processo
- ➔ i processi demone hanno `init` come genitore
- ➔ nella maggior parte dei casi, il nome dei processi demone finisce con una `d`

Processi demone: possibilità di avvio

- ➔ i processi demone possono essere avviati:
 - durante l'avvio del sistema, tramite gli script che normalmente sono in /etc/rc (dipende dal sistema)
 - dal processo cron (periodico), o da altri processi simili
 - da altri processi demone: è quello che avviene con il processo inetd
 - oppure anche da terminale, per esempio: `postfix start` per far partire un server SMTP

Lanciare un processo come demone/1

Passi base per lanciare un demone:

- 1 dal processo invocato, chiamare `fork` e quindi fare `exit` dal padre
 - una eventuale shell pensa che il comando sia terminato
 - il figlio ottiene un nuovo *process id*, ma mantiene lo stesso *process group id*, quindi non è il leader di un *process group*. Questo è necessario per la fase 2
- 2 chiamare `setsid` per creare una nuova sessione
 - il processo diviene session leader della nuova sessione (il sistema non lo considera privo di parent)
 - diviene process group leader
 - non ha un terminale di controllo

Lanciare un processo come demone/2

- 3 fare `chdir /`, in modo che la directory corrente sia la radice.
 - Il parent, infatti, poteva essere stato lanciato su un filesystem esterno, che altrimenti non potrebbe più essere smontato
- 4 reimpostare la *file creation mode mask* a 0
 - per evitare problemi riguardo a come fosse impostata in precedenza
- 5 chiude tutti i file descriptor non necessari
 - così non ci sono problemi con file descriptor ereditati dal parent
- 6 aprire il log di sistema

Funzione daemon_init

Visualizza

da notare:

- ➔ facciamo una fork, per non avere parent
- ➔ la seconda fork, per garantire anche che il processo non sia session leader e non possa riacquisire un terminale
- ➔ chiudiamo tutti i file descriptor, semplicemente provandoli!
- ➔ le tre chiamate ad open associano i primi tre file descriptor a /dev/null

Logging

- ➔ il processo deve comunque avere una opportunità di scrivere messaggi in output
- ➔ questo si può fare
 - scrivendo in un file apposito
 - scrivendo direttamente nel log di sistema

Logging: scrivere nel log di sistema

- ➔ si possono generare messaggi nel log:
 - usando le funzioni del kernel
 - usare la funzione syslog
 - inviare pacchetti UDP alla porta 514 (se abilitati)
- ➔ il demone `syslogd` si occupa di raccogliere i dati da queste fonti e inviarli al log
- ➔ la configurazione di `syslogd` è normalmente in `/etc`, possiamo vedere quali opzioni sono abilitate o disabilitate

Funzioni per manipolare il log

```
#include <syslog.h>
```

```
void openlog(const char *ident,  
             int option, int facility);  
void syslog(int priority,  
            const char *format, ...);  
void closelog(void);
```

Funzioni per manipolare il log

- ➔ chiamare `openlog` e `closelog` è opzionale
 - se non facciamo `openlog`, comunque verrà aperto al momento di chiamare `syslog`
 - se non facciamo `closelog`, il log verrà comunque chiuso al termine del programma
- ➔ con `openlog`, però, possiamo specificare una stringa `ident`, che viene quindi scritta insieme ad ogni singolo messaggio di log

Funzioni per manipolare il log

- ➔ syslog utilizza la formattazione come in printf, in più, ogni volta che si incontra la sequenza %m, questa viene sostituita con la stringa ottenuta chiamando `strerror(errno)`
- ➔ possiamo specificare:
 - una facility (il tipo di oggetto che fa log)
 - una priority (or logico fra facility e level)

Esempio:

```
#include "daemon_init.c"
```

```
int main() {  
    daemon_init( "daemon_example", LOG_USER );  
    syslog( LOG_ERR, "processo demone avviato"  
);  
  
    for( ; ; ) {  
        sleep(1);  
        syslog( LOG_ERR, "passato un  
secondo" );  
    }  
}
```

Esercizi

- ➔ scrivere un programma demone che:
 - alla ricezione di SIGUSR1, stampi un messaggio qualsiasi sul log
 - alla ricezione di SIGUSR2, lanci un processo figlio. Questo processo si limiterà a scrivere qualcosa nel log, per segnalare la propria creazione, e quindi terminerà dopo una sleep di 5 secondi
 - liberare le risorse del processo figlio! Gestendo il segnale
 - alla ricezione di SIGTERM, terminerà stampando qualcosa sul log per segnalare la propria terminazione