Demoni

- procese care ruleaza in background si nu sunt asociate cu un terminal de control
- in general porniti de scripturile sistemului de operare la sfarsitul operatiei de boot
- ce se intampla insa cand un server e pornit de utilizator dintr-un shell (terminal)?
  - trebuie sa se dezasocieze de terminal pentru a evita interactiuni nedorite cu controlul proceselor executat de shell, gestiunea sesiunii de terminal sau pt a nu tipari mesajele server pe terminal (avand in vedere ca ruleaza in background)
- exemple demoni
  - serviciile sistem din */etc/rc* pornite cu privilegii de superuser (*inetd/xinetd*, *Web*, *sendmail/postfix*, *syslogd* etc)
  - job-uri *cron/at* invocate de demonul de *cron*
  - servere/programe pornite din terminal

Output pt demoni

- pentru ca demonii nu au terminal de control au nevoie de metode specifice pt a afisa mesajele atunci cand au nevoie (ex mesaje de eroare, urgenta, logging, etc)
- metoda standard foloseste functia *syslog* care trimite mesaje catre demonul *syslogd (rsyslogd* pe sisteme Linux*)*
  - in general exista un fisier de configurare *syslog.conf* care specifica ce trebuie facut cu fiecare mesaj primit de demon:
    - se adauga la sfarsitul unui fisier
    - se logheaza pe consola *(/dev/console*)
    - se forwardeaza catre demonul *syslogd* al altei masini
  - serviciul functioneaza pe portul 514 UDP (v. */etc/services*)
  - la primirea semnalului SIGHUP se reciteste fisierul de configurare (in sistemele moderne se foloseste comanda *service reload*)
    - ex: $ kill –HUP <pid syslogd>
- mesajele se pot trimite direct pe portul 514 UDP, dar uzual se foloseste functia *syslog*

Functia syslog

- in lipsa terminalului de control demonul nu poate folosi functii de genul *fprintf* la *stderr*
- logarea mesajelor se face cu functia *syslog*

```
#include <syslog.h>

void syslog(int priority, const char *message, …);
```

- primul argument este o combinatie intre *nivel* si *facilitate* (*level/facility*)
- mesajul (al doilea parametru) este un string de formatare ca la *printf*
  - *%m* poate fi folosit pt a loga mesajul de eroare corespunzator valorii curente a *errno*
- mesajele logate au un nivel intre 0 si 7
- daca utilizatorul nu specifica nici un nivel se foloseste implicit LOG_NOTICE (nivel 5)
- nivelul permite ca toate mesajele de un anumit nivel sa fie tratate la fel

Functia syslog (cont.)

- mesajele logate contin o *facilitate* care identifica procesul care trimite mesajul
- valoarea implicita pentru *facilitate* este LOG_USER
- ex: mesaj de eroare demon care nu reuseste sa redenumeasca un fisier

  syslog(LOG_INFO | LOG_LOCAL2, “rename (%s,%s): %m”, file1, file2);

- valoarea facilitatatii permite ca toate mesajele venite de la o anumita facilitate sa fie tratate la fel in syslog.conf
  - ex:

```
kern.*          /dev/console
local7.debug    /var/log/cisco.log
```

Functia syslog (cont.)

- la primul apel *syslog* se creeaza un socket Unix UDP care se conecteaza la portul 514 si care ramane conectat pana cand se termina procesul
- alternativ, se pot folosi functiile *openlog/closelog*

```
#include <syslog.h>

void openlog(const char *ident, int options, int facility);

void closelog((void);
```

- *ident* este un string care va prefixa toate mesajele logate prin *syslog*
- *options* este o valoare de OR logic intre mai multe constante (v. man page)
- daca se specifica *facilitatea* la *openlog*, ulterior cand se apeleaza *syslog* nu se mai paseaza ca prim parametru decat *nivelul*
- mesajele se pot loga si cu comanda *logger* (din linie de comanda)

Structura demoni

- (1) detasarea de terminal
  - se apeleaza *fork* si procesul parinte se termina imediat si copilul continua
  - cand parintele termina, shell-ul crede ca s-a terminat comanda, ca atare copilul ruleaza automat in background
  - copilul mosteneste GID-ul parinte dar are propriul PID, fapt ce face ca el sa nu poata fi liderul grupului de procese
  - copilul apeleaza *setsid* pentru a crea o noua sesiune si a deveni lider de grup in noua sesiune
    - copilul nu mai are terminal de control in acest moment
  - copilul se protejeaza de semnalul SIGHUP si apeleaza *fork* din nou
  - primul copil, acum parinte, termina si copilul sau (al doilea copil) continua sa ruleze
  - al doilea apel *fork* garanteaza ca demonul nu poate capata un terminal de control daca decide in viitor sa deschida un terminal
    - Cand un lider de sesiune fara terminal deschide un terminal, acel terminal devine terminalul de control al liderului sesiunii
    - al doilea apel fork garanteaza ca al doilea copil nu mai e lider de sesiune si nu mai poate obtine un terminal de control
  - SIGHUP trebuie ignorat pt ca atunci cand liderul de sesiune (primul copil) se termina, toate procesele din sesiune (al doilea copil) primesc semnalul SIGHUP

Structura demoni (cont.)

- (2) schimbarea directorului de lucru curent
  - demonul schimba directorul de lucru curent in / (sau alt director la alegere)
  - daca demonul genereaza un fisier core, acesata se salveaza in directorul de lucru current
  - daca directorul de lucru al demonului era pe un sistem de fisiere si demonul continua sa lucreze acolo, sistemul de fisiere respectiv nu va putea fi demontat
- (3) inchiderea tuturor descriptorilor de fisiere deschisi
  - se inchid toti descriptorii de fisiere mosteniti de la procesul care a creat demonul (in mod normal, shell-ul)
- (4) redirectarea stdin, stdout si stderr la /dev/null
  - garanteaza ca descriptorii de fisiere corespunzatori sunt deschisi, dar citirea intoarce EOF iar scrierea se pierde
  - necesar pt ca functiile de biblioteca care presupun folosirea acestor descriptori sa nu esueze
- (5) utilizarea syslogd pentru logarea erorilor
  - apel *openlog* in mod normal cu numele procesului (argv[0])