Aurkibidea

PCB, blokeatuta-prest, memoria babestua 2

A3 azterketa ereduko lehen galdera: exekutagarria moduluekin lerro batean ala askotan sortu? 2

Liburutegi dinamikoak sortzeko -fPIC flag-a erabili..... 3

A3 azterketa ereduko 2. eta 3. galderetarak hobeto ulertzeko hainbat azalpen 3

gcc-en erabili -lorokor eta ez liborokor.so edo liborokor.a..... 4

SIGKILL seinalearen funtzionalitatea ezinda berdefinitu 5

C-ko azterketa ereduko lehen ariketa: funtzioa eta modulua 5

Pakete/zerbitzuen... konprobaketak eta bash -x 6

Seinale tratatu gabeekin zer? alarn eta pause beti batera? 7

Prozesuetarako Linux API-a atalerako laguntzatxoa 9

Prozesuetarako Linux API-a atalerako laguntzatxoa 10

cron eta anacron..... 11

PCB, blokeatuta-prest, memoria babestua

Kaixo,

PCB

Process Control Block-a (PCB), datu egitura bat da, non sistema eragileak prozesu bati dagokion informazio guztia gordeko duen. Beraz, prozesu berri bat sortzen denean berarekin batera sistema eragileak PCB berri bat sortuko du. Beraz, prozesu bakoitza bere PCB-an gordetzen dela esan daiteke.

Non gordetzen da prozesu bat blokeatuta dagoenean? eta prest dagoenean?

Prozesu bat memorian ala prest egon, printzipioz, memorian kargatuta, gordeta, egongo da (ikus 5_1_multiprogramazioa.pdf-ko 4. gardenkiko diagrama). Gauza da, prest bezala markatuta badago, schedulerrak exekutatzera pasatzeko hautagai bezala hartuko duela.

Memoria babestua

Erabiltzaileak sistema eragilearen APIa erabiltzen du, alegia, erabiltzaileak sistema dei hau erabili nahi dut parametro hauekin adieraziko dio. EDT-a atzitzea jada kernel-modua da, kernelak kontrolatuko du. Horregatik, erabiltzaileak sistemaren baliabideak sistema deien bitartez atzituko ditu, ez nahieran, eta honela lortzen da sistema erabiltzailearen "baldarkerietatik" babestea.

Kodean zuzenean edo zeharka sistema dei bati dei egitean, EDT taulara joango da, sistema dei hori zehazki memoriako zein helbidetan dagoen kargatuta jakiteko. Makina bakoitzean, memorian kargatutako sistema deiaren zatiak, egoiliar errutina izena hartuko du. Gauza da, EDT taulan sarrerak gutxitzeko sistema deiak multzokatu egiten direla, sakabanatze errutinatan. Beraz, EDT taulan sakabanatze errutinari dagokion helbidea lortuko da eta honen parametroen bidez jakingo da ondoren zehazki zein egoiliar errutinari hots egin. Ikus 4-SistemaDeiak.pdf-ko 31. gardenkiko irudia.

Ondo izan,

Iñigo Perona Balda

A3 azterketa ereduko lehen galdera: exekutagarria moduluekin lerro batean ala askotan sortu?

Kaixo,

A3 azterketa ereduko lehen galdera: exekutagarria moduluekin lerro batean ala askotan sortu?

Bi moduak ontzat hartuko dira. Modulu bakoitzaren .o konpilazio-emaitzak banan bana sortzeak duen abantaila, gero, hurrengo azpi-atalean makefile-a sortzeko behar diren komandoak jada sortuak eta probatuak dituzuela da. Nik gomendatu, lerro askotan egitea gomendatuko nuke, make atalerako lan erdia aurreratzen delako.

Ondo izan,

Liburutegi dinamikoak sortzeko -fPIC flag-a erabili

Kaixo,

Liburutegi dinamikoa sortzeko .so-n enpaketatuko diren .o konpilazio-emaitzak -fPIC flag bidez sortuak izan behar dute, estatikoan ez bezala. Honela, .o Position Independent Code (PIC) bezala sortuko da, liburutegia ez baita exekutagarrian txertatuko, estatikoan ez bezala eta beraz, liburutegiko elementuek posizio independentzia behar dute exekutagarriko posizioekiko.

Kontuan izan, despiste hori behin baino gehiagotan ikusi dizuet eta.

Ondo izan,

A3 azterketa ereduko 2. eta 3. galderetarak hobeto ulertzeko hainbat azalpen

Kaixo,

[A3] Azterketa ereduko 2. galderako kodea ulertzeko hainbat azalpen:

"man 2 lstat" eginez ikus daiteke, lstat-ek lehen argumentu bezala fitxategi izena hartzen duela eta bigarren argumentu bezala stat motako erregistrora pointerra.

"man 2 lstat" eginez RETURN VALUE aztertuz, 0 ondo badoa -1 arazoak egon badira. Beraz, -1 lortzen denean, fitxategia atzitzean arazoren bat egin den seinale.

"man 2 lstat"-etik baita ere, st_ino /* Inode number */ dela ikus daiteke. Inode zenbakia.

Beraz, programak, bi fitxategiek inode berdina duten ala ez esango du, beharrezko errore tratamenduank eginda.

Programa batek zenbat deskriptore zabalduko dituen galdetzen denean, kontuan izan defektuz beti prozesu orok 3 deskriptore irekita dituela: sarrera estandarra (stdin ,0), irteera estandarra (stout, 1) eta erroreentzako irteera (sterr, 2). Gogoratu berbideraketak egitean nola erabiltzen genituen 1>, 2> eta &>. Ondoren erabiltzen den fitxategi bakoitzeko honi deskriptore bat sortzen zaio, normalean honela: FILE *fd = open("fitx.txt", "rw"); Kasu honetan lstat-ek fitxategi izena jasotzen du parametro bezala ez deskriptorea. Bere bixkia den fstat-ek, aldiz, deskriptorea jasotzen du. Ikus "man 2 lstat". Hala ere, lstat-ek fitxategiak atzitu ahal izateko ezinbestean bere

barnean deskriptorea sortzen du. Beraz, programa honek 5 deskriptore irekitzen ditu. Hiru defektuzko eta beste bi argv[1] eta argv[2] fitxategiak atzitzeko. Erantzuna beraz bost da.

perror-i dagokionez, sistema deiek huts egiten dutenean, -1 balioa itzultzen dute, eta akatsaren zenbakia errno aldagaian gordetzen dute. perror() funtzioaren bitartez errno zenbakiari dagokion errore mezua pantailaratu dezakegu. Alegia, errno aldagaiak duen kodea, errore mezu irakurgarri batean pantailaratzen du perror-ek. Ikus man 3 perror.

[A3] Azterketa ereduko 3. galderako kodea ulertzeko hainbat azalpen:

for(;;) -> infinitorarte iteratzen geratzeko erabiltzen da. Gure kasuan iterazio infinitu hauek seinaleen bidez eteten saiatzen gara.

"kill -l" eginda ikus daiteke zein seinalerentzako berridazten den funtzionalitatea.

"man 7 signal" eginda, ikus daiteke seinale bakoitzaren besterik ezeko funtzioa:
SIGALRM 14 Term Timer signal from alarm(2)
SIGUSR1 30,10,16 Term User-defined signal 1
SIGUSR2 31,12,17 Term User-defined signal 2

Programan erabiltzen diren seinale guztiek besterik ezeko funtzionalitatea, prozesua akabatzea, terminatzea dute: Term. Baina 14a eta 10a birdefinitzen ditugu, beraz, ez dute besterik ezeko bere funtzioa exekutatuko, baizik eta guk birdefinitua. Alegia, ez dute Term exekutatuko.

Programaren jarraipena segundoz segundo ea hobeto ulertzen den, hala ere kodea kopiatu, konpilatu eta exekutatzea lagungarri izango zaizue:
- Exekuzioa hasi eta 3 segundora: umeak gurasoari 10 seinalea bidali
- Exekuzioa hasi eta 3 segundora: gurasoak 10 seinalea jaso trapper() exekutatu eta pause()-tik atera. Segidan umeari 12 seinalea bidali (Term).
- Exekuzioa hasi eta 3 segundora: umeak 12 jaso eta hilko da.
- Exekuzioa hasi eta 3+5 segundora: gurasoak 14 seinalea (SIGALRM) jasoko du, trapper exekutatu etta pause()-tik aterako da. Jada gurasoaren kill-ek ez du eraginik.
- Exekuzioa hasi eta 3+5+5 segundora: berdina, eta honela betirarte.

Ondo izan,

gcc-en erabili -lorokor eta ez liborokor.so edo liborokor.a

Kaixo,

gcc-ri liburutegiak adieraztean ez erabili liborokor.so edo liborokor.a formak, baizik eta erabili beti -lorokor forma. Liburutegiak adieraztean ez da liburutegiaren izen osoa erabiliko. Adibidez, liborokor.so beharrean -lorokor idatziko dugu. liborokor.so izenari hasierako lib eta bukaerako .so kenduko zaio. Ondoren liburutegia adierazteko -l (hau -l library-ri dagokio ez -I Include-ri) flag-ari itsatsita liburutegi izen murriztua ipiniko da.

Beti erabili modu hau, bai liburutegi dinamiko, bai estatikoetan, bestela liburutegiak uneko karpetan dagoenean bakarrik funtzionatzen du. **Laburbilduz, gcc-n liburutegiak adierazteko beti erabili behar den modua: -lorokor da.**

Ondo izan.

SIGKILL seinalearen funtzionalitatea ezinda berdefinitu

Kaixo,

Seinaleen atalean ikusi dugu, C kodea idaztean, prozesuak jasoko dituen seinaleen funtzionalitateak berridatzi ditzakegula. Seinaleari gure funtzzio bat esleitzeko honela egiten da:

signal(SIGINT, trapper);

Seinaleen posibleen zerrenda: kill -l

Baina badira bi seinale, SIGKILL eta SIGSTOP, zeintzuen funtzionalitateak ezin diren berridatzi. Logika ere badu, bestela programa bat egin genezake inongo seinalek ere gelditu ezingo lukeena. Kode gaizto batek esplotatu lezakeen ahulgunnea. SIGKILL aukera ukiezina izanda, erabiltzaileari programa bat geratzeko ahalmena bermatzen zaio.

Izan ere, batzuk SIGKILL-en funtzionalitatea berdefinitzen saiatu zarete eta ez zizuen ezer egiten. Arrazoia jakin dezazuen.

Ondo izan,

C-ko azterketa ereduko lehen ariketa: funtzioa eta modulua

Kaixo,

C-ko atalaren azterketa adibideko lehen ariketarako laguntzatxo bat:

- Enuntziatutik kodea kopiatzean kakotxekin kontuz, batzuetan konpilatzaileak karaktere arraro bezala detektatzen baititu.

- Lehenik probatu KateLuzapena.c, behar diren include-ak ipiniz (osotu gabe dagoela horregatik dio). Alegia, string.h-ko strlen funtzioa erabili eta probatu kodea:
#include <stdio.h>
#include <string.h>

- Warning-a kentzeko, gets-en ordez fgets erabiltzeko iradokitzen du, gets deprekatuta baitago. Beno, hau ez da beharrezkoa baina. "man 3 fgets" egin, nola funtzionatzen duen ikjusi eta, honela ordezka genezake:

fgets(kat, N, stdin);

- Norberak garatutako funtzioak ongi funtzionatzen duen ikusteko lehenik programa nagusia dagoen fitxategian probatu sortu duzuen kodea. Kasu honetan kendu #include <string.h>, ordezkatu srtlen(kat) mystrlen(kat)-gatik eta inplementatu mystrlen funtzioa bertan. Eta probatu kodea. Behin funtzionatzen duela, ekin modulua sortzeari.

- Eta funtzioa inplementatzen ez baduzue asmatzen, ez ataskatu puntu honetan, ipini funtzioaren gorputzean printf("String luzera kalkulatu\n") eta hurrengo pausuak egiten jarraitu.

- Behin funtzioak ondo egiten duela ziurtatuta sortu modulua: fkateak.h eta fkateak.c fitxategiak. Kontuan izan #ifndef definitzean fitxategiaren izena letra larriz jarri ohi dela:

vim fkateak.h
#ifndef _FKATEAK_H
#define _FKATEAK_H
int mystrlen(char kat[]);
#endif

- Gehitu moduluari dagokion goiburukoa programa nagusian, guk sortua denez kakotx bikoitzen artean:

#include "fkateak.h"

Lagungarri izango zaizuelakoan.

Pakete/zerbitzuen... konprobaketak eta bash -x

Zer konprobatu behar den eta zer ez

Irizpide orokor bezala eduki buruan, esandakoa automatizatu behar dela eta erabilterraza izango dela, erabiltzaile ez aditu batek exekutatzeko modukoa.

menu.sh-aren aukerak ordenean exekutatuko dira eta behin aukera bat exekutatu denean bera eta bere aurrekoak berriz exekutatzean arazorik ez luke eman behar.

Zerbait instalatzean edo sortzean, ea lehendik instalatua zegoen ala ez ikustea komeni da. Funtzioren batek pakete askoren instalazioa egiten badu, hauetatik nagusiena begiratzearekin nahiko, izan ere, gure script-ak, guztiak batera instalatu eta desinstalatuko ditu, blokean.

Zerbait testatzean, berau testatzeko behar den zerbitzua/zerbitzuak martxan dagoen/dauden begiratuko da.

netstat komandoa begiratu egingo da instalatuta dagoenentz, erabiltzaileari

automatikoki instalatzea erabilterrazago egiten baitu.

ssh komandoa ez da automatikoki instalatuko. Hau zerbitzu ezaguna da eta instalatuta dagoela suposatuko da, baita bertan autentifikazio oker eta zuzenak eginak dituela ere.

 # Erroreak topatzeko laguntzatxoa

Script-a egiten ari dena jarraitzeko eta erroreak topatzeko ondo etor dakizueke. Exekutatu scripta modu honetan:

bash -x menu.sh

man bash-etik:
-x: Print commands and their arguments as they are executed.

Seinale tratatu gabeekin zer? alarn eta pause beti batera?

Zer gertatuko litzateke prozesu batek tratatu gabeko seinaleren bat jasoko balu?

Linux-eko prozesu batek, jaso ditzakeen 64 seinaleek (kill -l) dute defektuzko funtzioren bat lotuta. Beraz, prozesu batek dena delako seinalea jasotzen badu, defektuzko, besterik ezeko funtzioa exekutatuko du. Defektuzko funtzio hauek "man 7 signal" egin eta bertako Signal-Value-Action-Comment tauletan ikus daitezke Action zutabean. Akzio hauek Term, Ign, Core, Ign, Stop eta Cont. "man 7 signal"-ek honela definitzen ditu:

The entries in the "Action" column of the tables below specify the default disposition for each signal, as follows:
- Term: Default action is to terminate the process.
- Ign: Default action is to ignore the signal.
- Core: Default action is to terminate the process and dump core (see core(5)).
- Stop: Default action is to stop the process.
- Cont: Default action is to continue the process if it is currently stopped.

Beraz guk kodean seinale bat tratatzean, defktuzko akzioa gainidatziko dugu eta seinale hau jasotzean gure funtzioa exekutatuko da defektuzkoa beharrean.

alarm deiaren ondoren beti dator pause?

alarm ez doa beti pause batekin. Pause-k seinaleren bat jaso artean (SIGALRN izan daiteke edo beste edozein) prozesua pausan uzten du. Probak egiteko alarm-pause bikotea egoki datorkigu baina ez dute zertan batera joan. Alegia, alarm bat ezarri dezakegu eta gero begizta luze batean sartu adibidez. Kasu honetan, SIGALRM seinalearen bidez begizta horretan zein puntutan aurkitzen garen informatzeko programatu dezakegu. Adibidez, ondoko programak zenbaki lehenak aurkitzen ditu, bat aurkitutakoan segundu bateko sleep-a egin eta aurrera jarraitzen du, horrela lehen MAX_TAM lehenak aurkitu arte. SIGALRM seinalea begiztan zein puntutan gauden

inprimatzeko erabiliko da, alegia, aldagai globalen informazioa inprimatuko du.

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

#define MAX_TAM 10000

int g_i = 0;
int g_lehena = 2;

void nirealarm(int sig){
   printf("ALARM: %d zenbaki lehenean, %d-garrenean doa.\n", g_lehena, g_i);
   alarm(3);
}

int main(){
   int j, lehena, kont;
   signal(SIGALRM, nirealarm);
   alarm(3);
   while(g_i<MAX_TAM){
      kont = 0;
      for(j=2; j<g_lehena; j++){
         if(g_lehena%j==0){
            kont++;
            break;
         }
      }
      if(kont==0){
         g_i++;
         sleep(1);
      }
      g_lehena++;
   }
}
```

Beraz, alarm baten ondoren pause bat ez da derrigorra, adibide hau kasu.

Ea hobeto ulertzen laguntzen dizuen.

Zaindu,

Prozesuetarako Linux API-a atalerako laguntzatxoa

Kaixo,

8) Beheko programak emanda, azaldu “./padre 60 hijo” deiak egiten duena:

padre-k reloj eta erabiltzaileak pasatzen dion programaren (kasu honetan hijo) arteko lasterketa moduko batean epaile lanak egiten ditu, non lehenaren, reloj-en exekuzio denbora erabiltzaileak finkatzen baitu (kasu honetan 60 segundu). Beraz, joku moduko honetan, erabiltzaileak lehen parametroan emaniko segundo kopurua bigarrenean emaniko programarena baino luzeagoa izatea lortu beharko da. Kasu horretan 0 itzuliko du, dena ondo joan dela adieraziz. Kontuan izan UNIX-eko komandoek dena ondo joan denean 0 itzultzen dutela, beraz konbenioa hau denez horrela suposatuko dugu. "./padre 60 hijo" exekutatu ondoren "echo $?" egin dezakegu aurreko aginduak itzuli duen kodea ikusteko. Gero exekutatu "./padre 5 hijo" eta jarraian "echo $?" eta ikusi zer pasatzen den.

Azalpen luzeagoa jarraian, argigarri izan daitekeelakoan:

reloj exekutagarriak parametrotzat hartzen duen segundo kopurura alarma ipintzen du eta seinaleren baten zain pause-n geratzen da. Beste seinalerik ezean SIGALRM helduko zaio eta berau tratatua duenez xtimer funtzioa exekutatuko da.

hijo-k, beste seinalerik ezean 6 segundo itxaron eta amaitzen du.

padre-k bi fork egiten ditu. Lehenengo fork-agatik sortutako prozesu klonak (umeak) exec egitean reloj exekutagarriaz ordezkatuko da. reloj-ek hasiera batean 60 segundo pasako ditu amaitzen, padre-ri deia ondokoa delako: ./padre 60 hijo. Bigarren fork-agatik sortutako klonak (umeak) exec egitean hijo exekutagarriaz ordezkatuko da. hijo-k hasiera batean 6 segundo beharko ditu amaitzen.

padre-k bi ume sortu dituenez, bi wait beharko ditu biei itxaron nahi badie. Eta nola daki zein prozesuk amaitu duen? ba wait-ek amaitu duen prozesu umearen PID-a itzultzen duelako. Beraz, lehenik hijo-k bukatzen badu (probableena, reloj-i 60 segundu pasatzen bazaio) reloj-i SIGKILL bidaltzen zaio eta akabatzen da. Lehenik reloj-ek amaitzen badu, guk beste terminal batetik kill komando bidez akabatzeko seinaleren bat bidali diogulako adibidez, hijo-ri SIGKILL bidaltzen zaio eta akabatzen du.

Azkenik prozesu umearen bizi denbora (hurbilpena behetik) kalkulatu eta inprimatzen da. exit kode bezala 1 itzultzen du lehenik reloj-ek amaitu badu eta 0 lehenik hijo-k amaitu badu. Izan ere, reloj eta hijo-k euren exekuzioa ondo joan bada 0 itzultzen dute eta padre-n status1 1-era lehenik reloj-ek amaitzen duen kasuan ipintzen da.

Ea argigarri zaizuen,

Prozesuetarako Linux API-a atalerako laguntzatxoa

Kaixo,

Erakutsi dizkidazuen zalantzetatik gauza batzuk argitzea komeni dela iruditzen zait eta honetarako, ariketa batzuen erantzunak baliatuko ditut. Erantzun asko azalpen zabalagoak emateko aprobetxatzen ditut, berez hain erantzun luzeak ez lukete behar.

1b) Zer egikaritzen da lehenago, gurasoa edo semea? Zergatik?

Umea. Gurasoak wait(&status) duelako, zeinaren bidez bere umeak bukatu arte zain geratuko baita.

Komentatu, fork()-ak bere programa klon bat sortuko duela exekuzio puntu berdinean, baina umeari zero itzuliko dio eta gurasoari >0, hau da, prozesu umearen PID-a. if-ren baldintzan ez bada konparaziorik egiten, bertako balioa >0 bada TRUE eta 0 bada FALSE. Beraz konturatu, besterik ezean, if-aren bi adarrak exekutako direla paraleloki, adar bat gurasoak eta bestea umeak.

1c) Hurrengo aginduan, wait(&status), zer da status aldagaian jasotzen dena?

status-ean gurasoak umearen egoera jasoko du eta egoera honen balioa umeak exit egitean itzulitako balioa izango da.

2.4) Aldatu aurreko trapper.c 1tik 9arteko seinaleak tratatzeko. Egikaritu ostean SIGUSR1 seinalea bidatzeko. Zein da erabili duzun komandoa?

Kodean 64 zenbakia 9-gatik ordezkatzea da ideia: for(i=1;i<=9;i++) egitea. Modu honetan lehen 9 seinaleak tratatzen dira (ikus kill -l) eta beste seinaleren bat jasoko balitz defektuzko funtzioa exekutatuko luke. Beraz, prozesu batek, besterik ez bada adierazten, defektuz, kill -l -eko seinale guztientzat defektuzko funtzio bat du, askotan programa etetea izango dena.

Seinale bat tratatzea programaren hasieran seiale hau gure funtzio bati lotzea da: signal(1, trapper). Gure funtzioaren barruan, trapper-en barruan, berriz signal azalten da baina hau sistema ezberdinen bateragarritasun kontuengatik da, gure sistemetan hori gabe ere funtzionatzen du, beraz ez eman garrantzirik, kodea irakurtzean ez balego bezala jokatu. Iruditu zait programaren ulermena nahastu egiten dizuela eta.

Kontuan izan baita ere, 64 direla erabili daitezkeen seinaleak, kill -l ageri direnak. Beraz, adibidez, 70. seinalea ez da existitzen eta beraz ezin da tratatu.

Beraz, SIGUSR1 edo 10. seinalea bidaltzen dugunean aldaketaren ostean (1etik 9ra) tratatu ganeko seinalea izango da eta defektuzko funtzioa exekutatuko da, kasu honetan programa etetea. Hau kill komandoaz egingo da, aukera ezberdinak ditugu 10. seinalea bidaltzeko:
kill -s SIGUSR1 1234

kill -10 1234
kill -USR1 1234

6. eta 7. ariketak) alarm-pause eta sleep

6. ariketan alarm(5) egitean hemendik 5 segundurako alarma ezartzen da, eta berehala pause()-ra pasako dela. Pausen seinale bat jaso zai geratuko da programa, 5 segundura jasoko duen seinalea SIGALRM izango da eta aurrera jarraituko du.

7. ariketan sleep(1) egiten denean, segundu batez geratzen da programa eta gero aurrera jarraituko du.

Konturatu alarm-pause bikoteaz eta sleep-ez kasu honetan efektu bera lortzen dugula, beti ere kodetik aparteko seinalerik ez dagoela suposatuz.

8. ariketa) execve

Konturatu execve sistema-deia dela, egin man 2 execve eta besteak bere front-end-ak, adibidez, man 3 execlp.

Konturatu execve agindua ongi burutzen bada bere ondorengo koderik ez dela exekutatuko. Kasu honetan, uneko programa mkdir programaz ordeztua baita, beraz, aurreko prozesuaren exekuzio-lerroa exec aginduarekin amaituko da eta beste programa batena kargatuko eta hasiko da. exec-en ondorengo lerroak programa ordezkapen hau egitean zerbait gaizki joan bada bakarrik exekutatuko dira.

Fijatu apunteetan, lehenik fork() egin eta gero umeari egiten zaiola exec. 12. gardenkiko irudiak ondo ilustratzen du egoera hau. 8. ariketako kasuan zuzenean programa nagusiari egiten zaio exec.

Ea laguntzen dizuen,

cron eta anacron

Kaixo,

Prozesuen kudeaketarako bash komandoen atalean, seigarren ariketan, cron eta anacron-ekin zailtasunak daudela dirudi eta hemen laguntzatxo bat:

c) hileko lehenengo bost egunetan, goizeko 9:00etan.
crontab -e
00 minututan
9 ordutan
1-5: hilabeteko lehen bost egunetan
*: edozein hilabetetan

*: edozein asteko egunetan
00 09 1-5 * * /home/lsi/backup.sh

d) orduro, abuztuko ostiraletan.
crontab -e
00 minututan
*: edozein ordutan
*: edozein hilabeteko egunetan
8: 8garren hilabetean, abuztuan
5: ostiraletan
00 * * 8 5 /home/lsi/backup.sh

Gogoratu cron-en berezitasuna, makina beti piztuta egongo dela suposatzen duela dela, zerbitzarietarako pentsatua, beraz adierazi unean egikarituko du adierazi komandoa. Makina itzalirik badago une horretan, piztean ez dira komando horiek berreskuratuko eta exekutatu gabe pasako dira.

e) Hilabetean behin exekuta dadin (man anacron).
MODUA OROKORRA.
vim /etc/anacrontab
aldia: @monthly
atzerapena (minututan): 15
lanaren identifikatzailea, log fitxategietan identifikatzeko, guk nahi duguna: anacron.hilabetero
@monthly 15 anacron.hilabetero /home/lsi/backup.sh

e) Hilabetean behin exekuta dadin (man anacron).
DEFEKTUZKO KARPETAK ERABILIZ: @hourly, @daily, @weekly eta @monthly denean defektuzko karpetak erabili daitezke (ls /etc/cron.*)
vim /etc/anacrontab
@monthly 15 anacron.hilabetero nice run-parts --report /etc/cron.monthly
nice: adierazi script/programari lehentasuna emateko, besterik ezean, ez bada ezer adierazten 10 lehentasuna emango dio.
run-parts: karpeta bateko programa eta script guztiak exekutatzen ditu
--report parametroaz komandoen irteerak adierazi fitxategira bolkatuko ditu

# Behin /etc/anacrontab-en lerroa gehituta,defektuzko karpetara gehitu zuen scripta:
sudo cp backup.sh /etc/cron.monthly

Gogoratu anacron-ek, makina itzali daitekeela suposatzen duela eta beraz, makina piztean exekutatu gabe pasa diren komandoak berreskuratuko ditu eta exekutatuko. Fijatu baita ere, anacron-en atzerapena ere adierazi behar dela (kasu honetan 15 minutu), pendiente dauden programak piztu berritan exekutatu ezkero makina gehiegi kargatzeko arriskua ekiditeko da.

Ondo izan,