Scripts Python:

- argparser
- Pipes
- Processos
- Signals
- Sockets

ASIX Curs 2014-2015

Exercici-15 (14-telnet_server_multi.py, 15-telnet_server_client_multi.py)
Exemen Processos

Repàs de python

L'objectiu d'aquest apartat és repassar concetes de Python / programació tractats el curs anterior a l'assignatura de programació. En concret es tracten els temes:

- tractament de fitxers de text
- llistes
- diccionaris
- ordenacions
- optparser: mecanisme que es presenta (nou!) als alumnes per tractar els arguments automatitzadament.

Exercici-1 (1-head_stdin.py)

Mostrar les 5 primeres línies d'un fitxer. El nom del fitxer a mostrar es rep com a argument, sinó es rep, es mostren les cinc primeres línies de l'entrada estàndard.
Sinpsys: $ head [file]

```
#! /usr/bin/python
#-*- coding: utf-8-*-
# --------------------------------------------------------------------
# head [file]
# 5 lines, default=stdin
# --------------------------------------------
# Escola del treball de Barcelona
# ASIX Hisi2 M06-ASO UF2NF1-Scripts
# Curs 2014-2015
# @edt Novembre 2014
# --------------------------------------------
import sys
NUMLIN=5
# Processar argument
fileIn=sys.stdin
if len(sys.argv)==2:
        fileIn=open(sys.argv[1],'r')
# Generar el llistat
lineCounter=0
for oneLine in fileIn:
        lineCounter+=1
        print oneLine,
        if lineCounter==NUMLIN: break
fileIn.close()
sys.exit(0)
```

Exercici-2 (2-tail_stdin.py)

Fer un tail mostrant les 5 últimes línies del fitxer rebut com a argument. Si no es rep cap fitxer es processa la entrada stàndard. s'utilitza 5.
Sinopsys: tail.py [-f fitxer]

```
#!/usr/bin/python
#-*- coding: utf-8-*-
# -------------------------------------------------------------
# tail [file]
# 5 lines, default=stdin
# -------------------------------------------------
# Escola del treball de Barcelona
# ASIX Hisi2 M06-ASO UF2NF1-Scripts
# Curs 2014-2015
# @edt Novembre 2014
# -------------------------------------------------
import sys
NUMLIN=5
# Processar argument
fileIn=sys.stdin
if len(sys.argv)==2:
        fileIn=open(sys.argv[1],'r')
# Generar el llistat
lines=[]
for oneLine in fileIn:
        lines.append(oneLine[:-1])
        if len(lines)>NUMLIN:
                del lines[0]
fileIn.close()
# Mostrar les línies
for oneLine in lines:
        print oneLine
sys.exit(0)
```

Exemple-3 (3-exemple_argsparse.py)

Exemple de codi de utilització de argsparse, llibreria que permet el processament automàtic d'arguments. En l'exemple es defineix un argument -f o --file fileArg i -n o --num maxArg.

```
# /usr/bin/python
#-*- coding: utf-8-*-
import argparse
import sys
```

```
parser = argparse.ArgumentParser(description='tail: mostrar les últimes n línies')

parser.add_argument('-f', '--file', dest='fileArg', default=sys.stdin,help='fitxer a processar, default=stdin',type=file)

parser.add_argument('-n', '--num', dest='maxArg', default=5,type=int,help='número de línies a mostar',metavar='numero-lines')

args = parser.parse_args()
print args
print args.fileArg
print args.maxArg
sys.exit()
```

Exercici-3 (3-tail_optparse.py)

Fer un tail (similar al head anterior) amb un argument addicional indicant el número de línies a mostrar. Els arguments poden anar en qualsevol ordre i faltar. Si no s'indica el fitxer l'ebtrada serà l'stdin. Si no s'indica un valor per el número de l'inies per defecte s'utilitza 5.
Sinopsys: tail.py [-f fitxer] [-n línies].

Presentació de la utilitat Python optparse per processar als arguments rebuts per un programa. Consultar la documentació a Pydoc.

```
#!/usr/bin/python
#-*- coding: utf-8-*-
# tail [-f file] [-n num]
# 5 lines, default=stdin
# --------------------------------------------
# Escola del treball de Barcelona
# ASIX Hisi2 M06-ASO UF2NF1-Scripts
# Curs 2014-2015
# @edt Novembre 2014
# --------------------------------------------
import sys
import argparse
# Processar els arguments amb argparse
parser = argparse.ArgumentParser(description='tail: mostrar les últimes n línies')
parser.add_argument('-f', '--file', dest='fileIn', default=sys.stdin,help='fitxer a processar, default=stdin',type=file)
parser.add_argument('-n', '--num', dest='maxArg', default=5,type=int,help='número de línies a mostar',metavar='NUMEROLINIES')
args = parser.parse_args()
# Generar el llistat
fileIn=args.fileIn
lines=[]
```

```
for oneLine in fileIn:
        lines.append(oneLine[:-1])
        if len(lines)>args.maxArg:
        del lines[0]
fileIn.close()
# Mostrar les linies
for oneLine in lines:
        print oneLine
sys.exit(0)
```

Exemple-4 (4-exemple_argsparse.py)

Exemple de utilització de 'choices' per indicar un conjunt de valors vàlids per a una opció, i com definir el tipus vàlid d'un argument (int, file, etc).

```
# /usr/bin/python
#-*- coding: utf-8-*-
import argparse
import sys

parser = argparse.ArgumentParser(description='tail: mostrar les últimes n línies')

parser.add_argument('-f', '--file', dest='fileArg', help='fitxer a processar,
default=stdin',action='append')

parser.add_argument('-n', '--num', dest='maxArg', default=5,type=int,help='número de
línies a mostar',metavar='NUMLINES',choices=[5,10,15])

args = parser.parse_args()
print args
print
print args.fileArg
print args.maxArg
sys.exit()
```

Exercici-4 (4-tail_nfiles.py)

Fer un tail similar a l'anterior però amb múltiples arguments file a processar. Si no se n'indica cap es procesa stdin. El número de línies a mostrar només pot pendre els valors 5, 10 o 15 (només un d'aquets tres) i no hi ha valor per defectes (és a dir, és un argument obligatòri).
Usar múltiples opcions -f en el optparser per indicar múltiples fitxers a processar.
Sinopsys: tail.py [-f fitxer]... [-n 5 | 10 | 15]

```
#!/usr/bin/python
#-*- coding: utf-8-*-
# tail [-f file]... [-n 5|10|15]
# 5 lines, default=stdin
# -------------------------------------------
# Escola del treball de Barcelona
# ASIX Hisi2 M06-ASO UF2NF1-Scripts
# Curs 2014-2015  Desembre 2014
# @edt Novembre 2014
# -------------------------------------------
import sys
import argparse
# Processar els arguments amb argparse
parser = argparse.ArgumentParser(description='tail: mostrar les últimes n línies')
parser.add_argument('-f', '--file', dest='fileList', help='fitxer a
processar,action="append",default=[])
parser.add_argument('-n', '--num', dest='maxLines', default=5,type=int,help='número de
línies a mostar',metavar='NUMEROLINIES',choices=[5,10,15])
args = parser.parse_args()
fileList=args.fileList
# Generar el llistat d'un fitxer
def fileTail(fileIn,maxLines):
        "Donal un file llistar les ultimes n línies"
        lines=[]
        for oneLine in fileIn:
        lines.append(oneLine[:-1])
        if len(lines)>maxLines:
        del lines[0]
        for oneLine in lines:
        print oneLine
# Pracessar cada un dels arguments fitxers rebuts
if args.fileList==[]:
        fileTail(sys.stdin,args.maxLines)
# Mostrar les dades per stdout
for fileName in args.fileList:
        try:
        fileIn=open(fileName,"r")
        print "File:",fileName, 60*"-"
        fileTail(fileIn,args.maxLines)
        fileIn.close()
        except:
        print "Error en processar el fitxer ",fileName
sys.exit(0)
```

Exemple-5 (5-exemple_objectes.py)

```
#! /usr/bin/python
#-*- coding: utf-8-*-
```

```
# @edt exemples - objectes
# llistar linies de tot el fitxer /et/passwd
import sys
import argparse
# Processar els arguments amb argparse
parser = argparse.ArgumentParser(description='mostrar linies d\'un fitxer')
parser.add_argument('-f', '--file', dest='fileIn', help='fitxer a processar, default=stdin',
type=file, default=sys.stdin)
args=parser.parse_args()
fileIn=args.fileIn
#Construir objecte
class UnixUsuari():
  "Classe unixUsuari: prototipus de /etc/passwd"
  def __init__(self,userLine):
  #metode constructor
        userFields=userLine.split(':')
        #propietats
        self.login=userFields[0]
        self.password=userFields[1]
        self.uid=userFields[2]
        self.gid=userFields[3]
        self.gecos=userFields[4]
        self.homeDirectory=userFields[5]
        self.shell=userFields[6]
  def getGecos(self):
        "retorna el valor de gecos"
        return self.gecos
  def putGecos(self,newGecos):
        "estableix el valor de gecos"
        self.gecos=newGecos
  def miniPrint(self):
        "mostra login, uid i gecos"
        return '%s %s %s' % (self.login,self.uid,self.gecos)
  def __str__(self):
        return '%s:%s:%s:%s' % (self.login,self.uid,self.gecos,self.shell)
# Exemples de utilització de ubjectes, propietats i mètodes
user1=UnixUsuari('login:x:10:15:edt:home:shell')
print user1
print user1.login,user1.uid
user1.putGecos('eduard')
print user1.getGecos()
print user1.miniPrint()
```

Exemple-5 (5-list_usuaris.py)

```
#! /usr/bin/python
#-*- coding: utf-8-*-
# @edt exemples - objectes
```

```
# llistar linies de tot el fitxer /et/passwd
import sys
import argparse
# Processar els arguments amb argparse
parser = argparse.ArgumentParser(description='mostrar linies d\'un fitxer')
parser.add_argument('-f', '--file', dest='fileIn', help='fitxer a processar, default=stdin',
type=file, default=sys.stdin)
args=parser.parse_args()
fileIn=args.fileIn
#Construir objecte
class UnixUsuari():
  "Classe UnixUsuari: prototipus de /etc/passwd"
  def __init__(self,userLine):
  #metode constructor
        userFields=userLine.split(':')
        #propietats
        self.login=userFields[0]
        self.password=userFields[1]
        self.uid=userFields[2]
        self.gid=userFields[3]
        self.gecos=userFields[4]
        self.homeDirectory=userFields[5]
        self.shell=userFields[6]
  def __str__(self):
        return '%s:%s:%s:%s' % (self.login,self.uid,self.gecos,self.shell)
# Posar tots els usuaris en una llista on cada llista es un objecte usuari
listUsers=[]
for line in fileIn:
        listUsers.append(UnixUsuari(line[:-1]))
# Mostrar la llista d'usuria
for oneUser in listUsers:
        print oneUser
```

Exercici-5 (5-sort_usuaris.py)

Cal mostrar els camps d'un compte d'usuari tipus /etc/passwd ordenats pel criteri indicat, que pot ser login, uid o gid.
Es rep com a argument opcional un nom de fitxer de tipus /etc/passwd, si no s'indica es processa l'entrada estàndard. Si no s'indica el criteri d'ordenació per defecte s'utilitza el Uid.
Sinopsys: sort_usuaris [-s login|uid|name] [-f fitxer]

```
#!/usr/bin/python
#-*- coding: utf-8-*-
# usage: sort_usuaris [-s login|uid|gid] [-f fitxer]
#  def: uid, stdin
# ------------------------------------------------------------
```

```
# Escola del treball de Barcelona
# ASIX Hisi2 M06-ASO UF2NF1-Scripts
# Curs 2014-2015
# @edt Desembre 2014
# ------------------------------------------------------------
import sys
import argparse
# Processar els arguments amb argparse
parser = argparse.ArgumentParser(description='mostrar linies d\'un fitxer')
parser.add_argument('-f', '--file', dest='fileIn', help='fitxer a processar, default=stdin',
type=file, default=sys.stdin)
parser.add_argument('-s', '--sort', dest='criteria', help='criteri per ordenar, default=uid',
default="uid",choices=["login","uid","gid"])
args=parser.parse_args()
fileIn=args.fileIn
# Construir objecte
class UnixUsuari():
  "Classe UnixUsuari: prototipus de /etc/passwd"
  def __init__(self,userLine):
  #metode constructor
        userFields=userLine.split(':')
        #propietats
        self.login=userFields[0]
        self.password=userFields[1]
        self.uid=userFields[2]
        self.gid=userFields[3]
        self.gecos=userFields[4]
        self.homeDirectory=userFields[5]
        self.shell=userFields[6]
  def __str__(self):
        return '%s:%s:%s:%s%s' % (self.login,self.uid,self.gid,self.gecos,self.shell)
# Funcions de comparació
def cmp_login(a,b):
        "comparador d'usuaris per login"
        if a.login > b.login:
        return 1
        if a.login < b.login:
        return -1
        return 0
def cmp_uid(a,b):
        "comparador d'usuaris per uid"
        if int(a.uid) > int(b.uid):
        return 1
        if int(a.uid) < int(b.uid):
        return -1
        if a.login > b.login:
        return 1
        if a.login < b.login:
        return -1
        return 0
```

```
def cmp_gid(a,b):
        "comparador d'usuaris per gid"
        if int(a.gid) > int(b.gid):
        return 1
        if int(a.gid) < int(b.gid):
        return -1
        if a.login > b.login:
        return 1
        if a.login < b.login:
        return -1
        return 0
# Crea llista usuaris
listUsers=[]
for line in fileIn:
  listUsers.append(UnixUsuari(line[:-1]))
# Seleccionar el criteri d'ordenacio
if args.criteria=="uid":
  listUsers.sort(cmp_uid)
elif args.criteria=="gid":
  listUsers.sort(cmp_gid)
else:
  listUsers.sort(cmp_login)
# Mostrar els usuaris ordenats
for oneUser in listUsers:
        print oneUser
sys.exit(0)
```

Exemple-5 (main_5-sort_usuaris.py)

```
#! /usr/bin/python
#-*- coding: utf-8-*-
# @edt exemples - objectes
# llistar linies de tot el fitxer /et/passwd
import sys
import argparse
# ------------------------------------------------------------------------
#Construir objecte
class UnixUsuari():
  "Classe UnixUsuari: prototipus de /etc/passwd"
  def __init__(self,userLine):
  #metode constructor
        userFields=userLine.split(':')
        #propietats
        self.login=userFields[0]
        self.password=userFields[1]
        self.uid=userFields[2]
        self.gid=userFields[3]
        self.gecos=userFields[4]
```

```
        self.homeDirectory=userFields[5]
        self.shell=userFields[6]
    def __str__(self):
        return '%s:%s:%s:%s%s' % (self.login,self.uid,self.gid,self.gecos,self.shell)
# Funcions de comparació
def cmp_login(a,b):
        "comparador d'usuaris per login"
        if a.login > b.login:
        return 1
        if a.login < b.login:
        return -1
        return 0
def cmp_uid(a,b):
        "comparador d'usuaris per uid"
        if int(a.uid) > int(b.uid):
        return 1
        if int(a.uid) < int(b.uid):
        return -1
        if a.login > b.login:
        return 1
        if a.login < b.login:
        return -1
        return 0
def cmp_gid(a,b):
        "comparador d'usuaris per gid"
        if int(a.gid) > int(b.gid):
        return 1
        if int(a.gid) < int(b.gid):
        return -1
        if a.login > b.login:
        return 1
        if a.login < b.login:
        return -1
        return 0
# --------------------------------------------------------------------
# Funció per encapsutar tot el codi del programa en una funció main
def main():
    # Processar els arguments amb argparse
    parser = argparse.ArgumentParser(description='mostrar linies d\'un fitxer')
    parser.add_argument('-f', '--file', dest='fileIn', help='fitxer a processar, default=stdin',
type=file, default=sys.stdin)
    parser.add_argument('-s', '--sort', dest='criteria', help='criteri per ordenar, default=uid',
default="uid",choices=["login","uid","gid"])
    args=parser.parse_args()
    fileIn=args.fileIn
    # Crear llista usuaris
    listUsers=[]
    for line in fileIn:
        listUsers.append(UnixUsuari(line[:-1]))
    # Ordenar segons criteri de comparacio
```

```
  if args.criteria=="uid":
        listUsers.sort(cmp_uid)
  elif args.criteria=="gid":
        listUsers.sort(cmp_gid)
  else:
        listUsers.sort(cmp_login)
  # Llistar els usuaris
  for oneUser in listUsers:
        print oneUser
  system.exit(0)

# Execució del main si es crida com a programa
if __name__ == "__main__":
  main()
```

```
$ python
>>>  import main_5-sort_usuaris.py
>>> help(main_5-sort_usuaris.py)
>>> dir(main_5-sort_usuaris.py)
```

Exercici-6 (6-sort_usuaris_grupname.py)

Llistar el “login uid gname” de cada un dels usuaris de fitxers tipus /etc/passwd i /etc/group. Cal poder ordenar per cada un d'aquests conceptes (login, uid i gname). Llistar en tres columnes alineades correctament.

Els arguments indicant els fitxers d’usuaris i de grups són obligatòris. El criteri d’ordenació és opcional, si no s’indica s’utilitzarà el uid.
Llistats en columnes de mida fixa “login uid gname”.

Observar que cal ‘lligar’, relacionar, els fitxers d’usuaris i de grups usant la relació entre el GID de cada un d’ells. Per a cada usuari cal mostrar el seu gname i no el seu GID.

Usar un diccionari amb clau=valor de *gid=gname*. Processar primer el fitxer de grups i crear el
diccionari. Llavors per a cada usuari del fitxer de passwd obtenir tots els camps en un objecte usuari (que inclou un camp gname). A cada usuari assignar-li el gname corresponet. Posar tots els objectes usuari en una llista.
Cal validar aquells casos en que el gid del fitxer de passwds no existeix en el fitxer de group.

Sinopsys: programa [-s login | uid | gid | gname] -u usuaris -g grups.

```
#!/usr/bin/python
#-*- coding: utf-8-*-
'''
# Llistar segons criteri d'ordenació (default login): "login gecos  gname"
# usage: sort_usuaris [-s login|gecos|gname] -u usuaris -g grups
# Mètode: crea un diccionari d={gid:gname}
# ------------------------------------------------------------
# Escola del treball de Barcelona
# ASIX Hisi2 M06-ASO UF2NF1-Scripts
# Curs 2014-2015  Desembre 2014
# ------------------------------------------------------------
import sys
from optparse import OptionParser
class unixUser():
    '''Classe unixUser: prototipus de /etc/passwd'''
    def __init__(self, userLine):
        userLine=userLine[:-1]
        self.login=userLine.split(':')[0]
        self.password=userLine.split(':')[1]
        self.uid=userLine.split(':')[2]
        self.gid=userLine.split(':')[3]
        self.gecos=userLine.split(':')[4]
        self.homeDirectory=userLine.split(':')[5]
        self.shell=userLine.split(':')[6]
        self.gname = userLine.split(':')[7]
    def __str__(self):
        #return self.account+' '+self.passwd+' '+self.uid+' '+self.gid+' '+self.gecos+'
'+self.homeDdirectory+' '+self.shell
        return '%-15s %-1s %5d %-15s %-30s %-30s %-30s' % (self.account[:14],
self.passwd, int(self.uid), self.gname, self.gecos[:29], self.homeDirectory[:29], self.shell)

    def cmp_gid(a, b):
        '''Ordena segons el gid del grup'''
        if a.gid < b.gid:
            return -1
        elif a.gid > b.gid:
            return 1
        else:
            return cmp_login(a, b)
    def cmp_gecos(a, b):
        '''Ordena segons la descripció del usuari'''
        if a.gecos < b.gecos:
            return -1
        elif a.gecos > b.gecos:
            return 1
        else:
            return ordena_login(a, b)
    def cmp_login(a,b):
        '''Comparador d'usuaris segons el login'''
        if a.login > b.login:
```

```
            return 1
        if a.login < b.login:
            return -1
        return 0

usage= '%prog [options] [arg] escriu %prog --help per a mes informacio'
parser = OptionParser(usage)
parser.add_option('-u', '--users', dest = 'fileUser', default = 'sys.stdin', help = 'Nom
del fitxer d\'usuaris')
parser.add_option('-s', '--sort', dest = 'criteria', choices = ['login','gecos','gname'],
default = 'login', help = 'Camp per on ordenar')
parser.add_option('-g', '--group', dest = 'fileGroup', help = 'Nom del fitxer de grups')
(options, args) = parser.parse_args()

# Processar arguments
try:
    if options.fileUser:
        fileUser = open(options.fileUser,'r')
    else:
        print 'Error: no s'ha indicat el fitxer d\'usuaris'
        sys.exit(2)
except IOError:
    print 'El fitxer d\'usuaris no existeix'
    sys.exit(1)
try:
    if options.fileGroup:
        fileGroup = open(options.fit_group,'r')
    else:
        print 'Error! No has indicat el fitxer de grups'
        sys.exit(2)
except IOError:
    print 'El fitxer de grups no existeix'
    sys.exit(1)

#Carrega informació del fitxer de grups en un diccionari
grupList = {}
userList = []
for groupLine in fileGoup:
    if groupLine.count(':') == 3:
        splited = linia.split(':')
        groupList[splited[2]] = splited[0]
fileGroup.close()

#Crea la llista d'usuaris del fitxer de passwd afegint-hi el gname
for userLine in fileUser:
    userLine = userLine.rstrip('\n')
    if userLine.count(':') == 6:
        splited = userLine.split(':')
        gid = splited[3]
        if gid in grups:
```

```
            userLine = userLine+":"+groupList[gid]
        else:
            userLine = userLine+":<unknown>"
        oneUser = unixUser(userLine)
        userList.append(oneuser)
    else:
        print 'Error: Linia mal formada, no processable→ %s' % userLine
fileUser.close()

#Ordenar segons el criteri indicat
if options.camp == 'login':
    usuaris.sort(cmp_login)
elif options.camp == 'grup':
    usuaris.sort(cmp_gid)
else:
    usuaris.sort(cmp_gecos)

# Llistar els usuaris:
print 'Login Password UID Grup\t\tNom\t\t\t\tDirectori\t\tShell'
for oneUser in userList:
    print oneUser
sys.exit(0)
```

Exemple-7 (7-exemple_dicgrups.py)

Exemple de com recòrrer el fitxers de grups i per a cada grup crear un objecte amb el gid, gname i una llista dels usuaris. Observar com posar aquests objectes a un diccionari de grups amb el format {gid:objecteGrup}

```
#! /usr/bin/python
#-*- coding: utf-8-*-
# @edt exemples - objectes
# carregar a uun diccionari de grups el fitxer de grups
import sys
import argparse
# Processar els arguments amb argparse
parser = argparse.ArgumentParser(description='mostrar linies d\'un fitxer')
parser.add_argument(dest='fileGroups', help='fitxer a processar, default=stdin',
type=file,metavar='file-group')
args=parser.parse_args()
# Classe que emmagatzema les dades d'un grup
class boxGroup():
  "Classe boxGroup on s'emmagatzema el gid, gname i llista logins"
  def __init__(self,groupLine):
        "mètode constructor"
        groupFields=groupLine.split(':')
        self.gname=groupFields[0]
```

```
        self.gid=groupFields[2]
        if len(groupFields)<3:
        self.userList=[]
        else:
        self.userList=groupFields[3]
  def __str__(self):
        return '%5d %15s %5d' % (int(self.gid), self.gname, len(self.userList))
# Crear / Populate el dirccionari
dicGroup={}
for lineGroup in args.fileGroups:
  oneGroup=boxGroup(lineGroup[:-1])
  dicGroup[oneGroup.gid]=oneGroup
  print oneGroup
args.fileGroups.close()
print dicGroup
sys.exit(0)
```

Exercici-7 (7-compta_usuaris.py)

Llistar el gid, gname i número d'usuaris de cada grup, ordenat per algun d'aquests conceptes (gid, gname i nº -users). Usar un diccionari amb la clau gid i com a valor un objecte *group*. Aquest objecte conté el gid, gname i la llista d'usuaris que pertanyen al grup.
Mostrar el diccionari ordenat pel criteri corresponent utilitzant llistes índex.

Aquest exercici permet practicar: assignar objectes dins de diccionaris; manipular dades dins d'objectes que estan dins de diccionaris o llistes; recorrer diccionaris; mostrar diccionaris per ordre de clau.

També es practica a generar índex a memòria per realitzar ordenacions per critèris diferents.
Sinopsys: programa [-s gid | gname | nusers] -u usuaris -g grups.

```
#! /usr/bin/python
#-*- coding: utf-8-*-
# -------------------------------------------------------------------
# Escola del treball de Barcelona
# ASIX Hisi2 M06-ASO UF2NF1-Scripts
# Curs 2014-2015  Desembre 2014
# -------------------------------------------------------------------
import sys
import argparse
# Processar els arguments amb argparse
parser = argparse.ArgumentParser(description='mostrar linies d\'un fitxer')
parser.add_argument(dest='fileUsers', help='fitxer a processar, default=stdin',
type=file,metavar="file-users")
parser.add_argument(dest='fileGroups', help='fitxer a processar, default=stdin',
```

```
type=file,metavar='file-group')
parser.add_argument('-s', '--sort', dest='criteria', help='criteri per ordenar, default=uid',
default="gid",choices=["gid","gname","nusers"])
args=parser.parse_args()

# Definició de classes
class UnixUsuari():
  "Classe UnixUsuari: prototipus de /etc/passwd"
  def __init__(self,userLine):
        "mètode constructor"
        userFields=userLine.split(':')
        self.login=userFields[0]
        self.password=userFields[1]
        self.uid=userFields[2]
        self.gid=userFields[3]
        self.gecos=userFields[4]
        self.homeDirectory=userFields[5]
        self.shell=userFields[6]
  def __str__(self):
        return '%-15s %5d %5d %-15s' %
(self.login,int(self.uid),int(self.gid),self.shell[0:15])
class boxGroup():
  "Classe boxGroup on s'emmagatzema el gid, gname i llista logins"
  def __init__(self,groupLine):
        "mètode constructor"
        groupFields=groupLine.split(':')
        self.gname=groupFields[0]
        self.gid=groupFields[2]
        if groupFields[3]==":
        self.userList=[]
        else:
        self.userList=groupFields[3].split(',')
  def __str__(self):
        return '%5d %15s %5d' % (int(self.gid), self.gname[:15], len(self.userList))

# Crear el dirccionari de grups
dicGroup={}
for lineGroup in args.fileGroups:
  oneGroup=boxGroup(lineGroup[:-1])
  dicGroup[oneGroup.gid]=oneGroup
args.fileGroups.close()

# Processar els usuaris
for line in args.fileUsers:
  oneUser=UnixUsuari(line[:-1])
  if oneUser.gid in dicGroup:
        if oneUser.login not in dicGroup[oneUser.gid].userList:
        dicGroup[oneUser.gid].userList.append(oneUser.login)
  else:
        dicGroup[oneUser.gid]=boxGroup("<unknown%s>:x:%s:%s" %
```

```
(oneUser.gid,oneUser.gid,oneUser.login) )
        #sys.stderr.write( "%s:%s:%s:Error-login-uid-gid, el gid no existeix al
/etc/groups\n" % (oneUser.login,oneUser.uid, oneUser.gid))
args.fileUsers.close()

# Generar la llista índex (criteri,k)
header=()
indexList=[]
if args.criteria=="gname":
   header=("gid","gname*","num-users")
   for k in dicGroup:
        indexList.append((dicGroup[k].gname,k))
elif args.criteria=="nusers":
   header=("gid","gname","num-users*")
   indexList=[( len(dicGroup[k].userList),k) for k in dicGroup ]
elif args.criteria=="list":
   pass
else:
   header=("gid*","gname","num-users")
   indexList=[ (int(k),k) for k in dicGroup]

# Ordenar i llistar elselements
indexList.sort()
print '%5s %15s %5s' % header
for c,k in indexList:
   print dicGroup[k]
sys.exit(0)
```

Exercici-8 (8-shells_usuaris.py)

(suprimit per falta de temps!)

Llistar “shell num-users” “login gname” dels usuaris que usen aquest shel. El lllistat mostra ordenades les línies per núemro d'usuaris de cada shell de major a menor número d’usuaris.

Dins de cada línia de detall el llistat mostra el login i el gname dels ususrais que tenen aquest shell assignat per ordre alfabètic de login.

Sinopsys: programa -u fitxer_usuaris -g fitxer_grups.

Pendent de arreglar el codi

```
#!/usr/bin/python
#-*- coding: utf-8-*-
'''
# llistar per a cada sehell el seu nom, no d'usuaris que l'utilitzen i la
# llista d'aquests usuaris en format usuari:gname per a cada usuari.
#
# shell, no users shell:login.group, llista users k utilitxzen aquell shell(login.gname)
# de + 5
```

```
# a - 4
#
3
#
...
# usuaris per ordre alfabetic
# en cas empat no per nom de shell
# prog -u usuaris -g grups
#
# Escola del treball de Barcelona
# ASI Hisi2 Curs 2009-2010
# Octubre 2009
#
# Víctor Piniés
'''
from optparse import OptionParser
import os
import sys
class groups():
#metode constructor
def __init__(self, lin):
lin=lin[:-1]
split=lin.split(':')
self.gname=split[0]
self.gid=split[2]
#opcions d'arguments
usage='%prog [options]\ntype %prog -h or %prog --help for more info' #sera el
usage del programa. es mostrara quan hi hagi un error d'arguments
parser= OptionParser(usage)
parser.add_option('-u', '--fileus', dest='fitxeru',default=sys.stdin, help='Nom del
fitxer d\'entrada usuaris')
parser.add_option('-g', '--filegr', dest='fitxerg', help='Nom del fitxer d\'entrada
groups')
(options, args) = parser.parse_args()
#mirar si fitxer existeix
fitxeru=options.fitxeru
fitxerg=options.fitxerg
try:
if fitxeru != sys.stdin:
fdu=open(fitxeru,"r")
else:
fitxeru = sys.stdin
if os.path.exists(fitxerg):
fdg=open(fitxerg,'r')
else:sys.exit(1)
except IOError:
print "el fitxer %s no existeix" % (fitxer)
sys.exit(1)
dshell={}
dgroups={}
```

```
#recorro fitxer grups i guardo nom i gid
for linia in fdg:
gr=groups(linia)
dgroups[int(gr.gid)]=gr
fdg.close()
#recorro fitxer usuaris
for linia in fdu:
linia=linia[:-1]
if linia.count(':')==6:
split=linia.split(':')
gid, login, shell= int(split[3]), split[0], split[6]
if shell not in dshell.keys():
dshell[shell]=[]
if gid not in dgroups.keys():
dgroups[gid]=groups("nosiste:"+gid+":"+login)
dshell[shell].append(login+'.'+dgroups[gid].gname)
fdu.close()
#llista index on guardo no users + shell(clau dic)
li=list()
for k in dshell.keys():
li.append((len(dshell[k]),k))
#ordenar al reves
li.sort(reverse=True)
#printar per no users de + a - i ordenar per nom usuaris
for el in li:
key=el[1]
nusers=el[0]
dshell[key].sort()
print '%-20s\t%5d\t%-100s'%(key, nusers, dshell[key])
```

Pipes

Aquest apartat mostra la utilització de named pipes del bash amb la utilitat mkfifo. També la constricció de programes Python per a realitzar pipes utilitzant una de les seves llibreries: popen.

Exemple de Named Pipes:

A continuació es poden veure tres exemples de creació i utilització de named pipes. Cal observar:

- *mkfifo* és l'ordre bash que permet la creació de named pipes
- *subproces.Popen* és una de les llibreries (varia segons la versió) que es poden usar en Python per generar named pipes.

Exemple-1: Named pipe en consola bash

```
[a]$ mkfifo /tmp/dades
        [b]$ cut -f1-3 < /tmp/dades
[a]$ cat /etc/passwd
```

Exemple-2: Named pipe (2) en consola bash

```
[a]$ mkfifo /tmp/dades
        [b]$ tail -f < /tmp/dades
[a]$ cat /var/log/messages
```

Exemple-3: Popen usant Python

Programa Python *9-exemple_pipe.py* que utilitza les llibreries subproces.Popen. Executa un subproces ls via un pipe i llegeix del pipe.

El concepte clau a entendre és que si el Popen executa una ordre i finalitza, el stdout d'aquesta ordre es pot llegir amb un buble o reallines() perquè hi ha el fi de fluxe al final (generat en acabar el procés executat pel Popen).

En canvi, si el Popen executat una ordre que es manté en execució 8un bash, un access a psql obert) i es passa al popen via write al stdin cada ordre a executar, cal saber d'avantmà quantes línies de resposta generarà el popen per poder-les llegir del seu stdin. Si es fa un bucle o un readlines() es quedarà 'enganxat' perquè la ordre executada (via write) no acaba la seva sortida amb el control+d de fi de fluxe.

Per tant, ens trobem com en els sockets, o sabem com serà la resposta o bé cal un mecanisme de senyalització del final de la resposta.

```
#!/usr/bin/python
#-*- coding: utf-8-*-
'''
# llista un directori
# -------------------------------------------------------------
# Escola del treball de Barcelona
# ASIX Hisi2 M06-ASO UF2NF1-Scripts
# @edt Curs 2014-2015  Desembre 2014
# -------------------------------------------------------------
'''
import sys
from subprocess import Popen,  PIPE
import argparse

parser = argparse.ArgumentParser(description='%prog [options] [arg]\ntype %prog -h or
%prog --help per a mes informacio')
parser.add_argument(dest='ruta', help='Ruta a llistar',metavar='ruta-a-llistars')
args = parser.parse_args()

# Executa el pipe/subprocess
command = ["ls", args.ruta]
pipeData = Popen(command, stdout=PIPE, stderr=PIPE)
print pipeData.stdout.read(),

sys.exit(0)
```

```
pipeData = Popen("bash", shell=True, stdin=PIPE, stdout=PIPE, stderr=PIPE)
pipeData.stdin.write("telnet localhost 80 \n")
pipeData.stdout.readline()
pipeData.stdout.readline()
pipeData.stdout.readline()
pipeData.stdin.write("GET / HTTP/1.0 \n\n")
line = pipeData.stdout.readline()
while line!='':
  print "->", line
  line = pipeData.stdout.readline()
  print "_", line
```

```
pipeData = Popen("bash", shell=True, stdin=PIPE, stdout=PIPE, stderr=PIPE)
pipeData.stdin.write("ls ; echo FI\n")
line = pipeData.stdout.readline()
while "FI" not in line:
   print "->", line
   line = pipeData.stdout.readline()
```

```
pipeData = Popen("bash", shell=True, stdin=PIPE, stdout=PIPE, stderr=PIPE)
```

```
pipeData.stdin.write("ls ; echo FI\n")
line = pipeData.stdout.readline()
while "FI" not in line:
  print "->", line
  line = pipeData.stdout.readline()
```

```
#!/usr/bin/python
#-*- coding: utf-8-*-
'''
# exemple signal + popen
# ------------------------------------------------------------
# Escola del treball de Barcelona
# ASIX Hisi2 M06-ASO UF2NF1-Scripts
# @edt Curs 2014-2015  Desembre 2014
# ------------------------------------------------------------
'''
import sys, signal
from subprocess import Popen,  PIPE

# Definir els senyals a gestionar
def terminate(signum, frame):
        print 'Signal handler called with signal', signum
        print "closing program..."
        sys.exit(0)
signal.signal(signal.SIGALRM, terminate)   #14
signal.signal(signal.SIGUSR1, terminate)   #10
signal.alarm(60)

# Executa el pipe/subprocess
pipeData = Popen("bash", shell=True, stdin=PIPE, stdout=PIPE, stderr=PIPE)

while True:
        cmd= raw_input()
        pipeData.stdin.write("%s ; echo FI\n" % (cmd))
        line = pipeData.stdout.readline()
        while "FI" not in line:
           print "->", line,
           line = pipeData.stdout.readline()

sys.exit(0)
```

Exercici-9 (9-pipe_psql.py)

Consulta postgres 'hardcoded' a la base de dades training.
Sinopsys: programa

Variant: acceptar una sentència *sql* com a argument o com una línia provenint de stdin.
Problema: Explicar el perill del *sql injectat*, que no s'ha de realitzar mai directament.

També que no es programa així a pèl sinó que existeixen llibreries Python de programació *sql*.

```
#!/usr/bin/python
#-*- coding: utf-8-*-
'''
# Consultar dades que tenim a Postgres mitjançant una pipe amb PSQL.
# Select * from clientes.
# ------------------------------------------------------------
# Escola del treball de Barcelona
# ASIX Hisi2 M06-ASO UF2NF1-Scripts
# Curs 2014-2015  Desembre 2014
# ############################################

'''
from subprocess import Popen,  PIPE

# Generar el popen
command = "psql -qtA -F ';' training"
pipeData = Popen(command, shell = True, stdin=PIPE, stdout=PIPE, stderr=PIPE)
pipeData.stdin.write("select * from clientes;\q\n")

# Mostrar la sortida per stdout
for line in pipeData.stdout.readline()
   print line,

sys.exit(0)
```

```
>>> cmd="psql -d training -c \"select * from cliente where cliecod=2111;\n\" -h
192.168.2.37"
>>> pipeData=Popen(cmd, shell=True, stdin=PIPE, stdout=PIPE, stderr=PIPE)
>>> pipeData.stdout.readlines()
[' cliecod |  nombre  | repcod | limcred  \n', '---------+----------+--------+----------\n', '
2111 | JCP Inc. |       103 | 50000.00\n', '(1 fila)\n', '\n']
```

```
>>> cmd="psql -qtA training -h 192.168.2.37">>> pipeData=Popen(cmd, shell=True,
stdin=PIPE, stdout=PIPE, stderr=PIPE)
>>> pipeData.stdin.write("select * from cliente;\n")
>>> pipeData.stdout.readline()'2111|JCP Inc.|103|50000.00\n'
>>> pipeData.stdout.readline()
'2102|First Corp.|101|65000.00\n'
…. mes registres
```

Exemple-9 (9-pipe_psql_jose.py)

En aquest exemple s'utilitza sql injectat i per tant, es pot fer qualsevol maldat a la base de dades, ja que el que escriu l'usuari a la línia de comandes és el que s'executa.

```
#!/usr/bin/python
#-*- coding: utf-8-*-
# Jose Luis Grande Castro
```

```
# isx41011398
# @edt exemples - objectes
#

import sys
import argparse
from subprocess import Popen,   PIPE
box = argparse.ArgumentParser() # Constructor d'objecte
box.add_argument(dest='sentencia', help='Insereix la sentencia sql per fer la consulta.')
# posar tots els usuaris en una llista on cadea llista és un objecte usuari
args = box.parse_args()
sentencia=args.sentencia


cmd="psql -qtA -d training -c \"%s;\" -h 192.168.2.59"%(sentencia)
pipeData=Popen(cmd, shell=True, stdin=PIPE, stdout=PIPE, stderr=PIPE)

for line in pipeData.stdout:
        print line[:-1]
sys.exit(0)
```

Exercici-10 (10-pipe_psql_multi.py)

Ampliar l'exercici anterior fent que es puguin consultar les dades de cada un dels clients indicats com a arguments.
Sinopsys: programa client[...]

Ampliació: Rep un argument que és la base de dades amb la que conectar i un o més codis de client a consultar. també ha de llistar quants clients consultats i el total del Crèdit acumulat d'aquests clients.
Sinopsys: programa -b baseDades client[...]

```
Exemple d'execució:
$ programa.py -b bd -c 3240 -c 2185 -c 3741
  3240 | PEPITO  | 32 | 100.000,00
  2185 | JUANITO | 27 | 150.000,00
  3741 | PEDRITO | 15 | 250.000,00
  nº clients = 3    Total credit = 500.00,00
```

```
#! /usr/bin/python
# prog cliecod...
# -------------------------------------------
# @edt Desembre 2014
# ASX M06-ASO Scripts
# -------------------------------------------
import sys, argparse
from subprocess import Popen, PIPE
parser = argparse.ArgumentParser(description='llistar dades de cada client')
parser.add_argument(dest='clientList', help='num clie a la BD (cliecod)', nargs='+')
```

```
args = parser.parse_args()
#clientList=['2111','2102','2103','2123']

cmd="psql -qtA -d training -h 192.168.2.37"
pipeData=Popen(cmd, shell=True, stdin=PIPE, stdout=PIPE, stderr=PIPE)

for cliecod in args.clientList:
        sql="select * from cliente where cliecod=%s;\n" % (cliecod)
        pipeData.stdin.write(sql)
        print pipeData.stdout.readline()[:-1]
sys.exit(0)
```

Exercici-11 (11-pipe_ping.py)

Codi pendent de revisar!

Determinar l'estat de la xarxa, indicar els canvis d'estat de start a stop dels equips que contesten (o no) als nmap. Volem saber quines màquines de la xarxa hi ha vives. Validar cada 5 minuts quines hi ha connectades.
Portar un registre amb: PCxx : dia : start | stop : hora. Per saber qui hi ha connectat: (nmap)ping -b 192.168.0.0. Podem fer el ping cada 5 minuts. Si una màquina contesta: alta de nou, o si ja havia contestat abans no fer res. Si no contesta: si era start abans, fer stop. Si abans no hi era no fer res.
Per utilitzar sleep usar la llibreria time.
Mètodes de fluxes: open, write, read, flush, close.

```
#!/usr/bin/python
#-*- coding: utf-8 -*-

'''
Fem ping a una xarxa per veure quins pc's estan encesos i quins no
o quins s'encenen o quins s'apaguen. Guardem en un registre si s'encenen o s'apaguen.
Fem servir nmap.
#
# Escola del treball de Barcelona
# ASI Hisi2 Curs 2009-2010
# Novembre 2009
'''

from subprocess import Popen, PIPE
import sys
import time

#nmap -n -sP 192.168.0.0/24 #ping scan amb nmap
command= ["nmap -n -sP 192.168.0.0/24 | grep 'Host' |cut -d ' ' -f2 "]
llista1=[]

while True:
    #llegim sortida nmap i guardem a la llista2
    pipe=Popen(command, stdout=PIPE, shell=True)
    llista2=[]
    for ip in pipe.stdout:
        ip=ip[:-1]
        llista2.append(ip)
```

```
    #comprovem si les ip's de la llista2 estan a la llista1
    for ip in llista2:
        if ip not in llista1:
            print time.strftime("%d/%m/%y %H:%M"), ip, "up"

    #comprovem si les ip's de la llista1 estan a la llista2
    for ip in llista1:
        if ip not in llista2:
            print time.strftime("%d/%m/%y %H:%M"), ip, "down"

    #guardem llista2 a la llista1
    llista1=llista2
    #borrem tuberia
    del pipe
    #esperem 5 segons
    time.sleep(3)

sys.exit(0)
```

Signals

En aquest apartat es veurà com manipular a través de programes de Python senyals, com generar des dels programes senyals i com capturar i reaccionar als senyals rebuts.

```
 1) SIGHUP    2) SIGINT    3) SIGQUIT    4) SIGILL    5) SIGTRAP
 6) SIGABRT    7) SIGBUS    8) SIGFPE    9) SIGKILL    10) SIGUSR1
11) SIGSEGV    12) SIGUSR2    13) SIGPIPE    14) SIGALRM    15) SIGTERM
16) SIGSTKFLT    17) SIGCHLD    18) SIGCONT    19) SIGSTOP    20) SIGTSTP
21) SIGTTIN    22) SIGTTOU    23) SIGURG    24) SIGXCPU    25) SIGXFSZ
26) SIGVTALRM    27) SIGPROF    28) SIGWINCH    29) SIGIO    30) SIGPWR
31) SIGSYS    34) SIGRTMIN    35) SIGRTMIN+1    36) SIGRTMIN+2    37) SIGRTMIN+3
38) SIGRTMIN+4    39) SIGRTMIN+5    40) SIGRTMIN+6    41) SIGRTMIN+7    42) SIGRTMIN+8
43) SIGRTMIN+9    44) SIGRTMIN+10    45) SIGRTMIN+11    46) SIGRTMIN+12    47)
SIGRTMIN+13
48) SIGRTMIN+14    49) SIGRTMIN+15    50) SIGRTMAX-14    51) SIGRTMAX-13    52)
SIGRTMAX-12
53) SIGRTMAX-11    54) SIGRTMAX-10    55) SIGRTMAX-9    56) SIGRTMAX-8    57) SIGRTMAX-7
58) SIGRTMAX-6    59) SIGRTMAX-5    60) SIGRTMAX-4    61) SIGRTMAX-3    62) SIGRTMAX-2
63) SIGRTMAX-1    64) SIGRTMAX
```

Exemple-12 (12-example_sigalarm.py)

El programa s'apaga automàticament passats n segons utilitzant una alarma. Fa un bucle infinit que per exemple simula una acció I/O bloquejada i passats n segons genera un senyal sigalarm. Aquest és capturat per un handler defnit en el programa que finalitza l'execució generant una excepcció.

```
# Exemple de gestio del senyal sigalarm
import signal, os

# Funcions handler de senyals rebuts
def handler(signum, frame):
        print 'Signal handler called with signal', signum
        raise IOError("Couldn't open device!")

# Set the signal handler and a 5-second alarm
signal.signal(signal.SIGALRM, handler)
signal.alarm(5)

# This open() may hang indefinitely / Bucle infinit
while True:
        pass
# fd = os.open('/dev/ttyS0', os.O_RDWR)

# Disable the alarm
```

```
signal.alarm(0)
```

```
#! /usr/bin/python
# Signal Handler examples
# @edt Desembre 2014
import signal, sys

# Funcions handler de senyals rebuts
def terminate(signum, frame):
        print 'Signal handler called with signal', signum
        print "closing program..."
        sys.exit(0)
def happyDay(signum, frame):
        print "Congratulations!", signum, "signal selected"
        print "Have a nice day!"
def sigReload(signum, frame):
        print "MATRIX reloaded!"
        print "where is sion?"

# Set the signal handler and a 5-second alarm
signal.signal(signal.SIGALRM, terminate)  #26
signal.signal(signal.SIGUSR1, terminate)  #10
signal.signal(signal.SIGUSR2, happyDay)   #12
signal.signal(signal.SIGHUP, sigReload)   # 1
signal.signal(signal.SIGINT, signal.SIG_IGN) # ignore ctrl+c #2
signal.alarm(60)

# Iterate indefinitely
while True:
        pass
```

Exercici-12 (12-pipe_signal.py)

Codi pendent de revisar!

Bsat en el programa 8-popen que detecta equips engegats a la xraxa via nmap.

Per matar el programa a hores d'ara s'utilitza ctrl+c i s'aborta a lo bèstia. Cal fer una finalització ordenada. Cal definir en el programa una senyal que quan es rebi (fent un kill) aturi ordenadament el programa. Mirar el mòdul signal de python. Per ignorar el ctrrl+c signal.signal(signal.SIGINT,signal.SIG_IGN)

Ampliar l'exercici anterior fent que el programa s'apagui automàticament passats n segons. Utilitzar una alarma. Passats n segons genera un senyal sigalarm que ha d'apagar ordenadament el programa.

```
#!/usr/bin/python
#-*- coding: utf-8 -*-
'''
```

```
# prog
'''
from subprocess import Popen, PIPE
import sys
import time
import signal

def cierra(signum, frame):
        sys.exit(0)

signal.signal(signal.SIGALRM, cierra)
signal.signal(signal.SIGUSR1, cierra)
signal.signal(signal.SIGINT, signal.SIG_IGN)
signal.alarm(1200)

nmap = "nmap -sP -n 192.168.0.0/24 | egrep 'Host'"
lista_Ant = []
while True:
        tubo = Popen(nmap, stdout=PIPE, shell=True)
        lista_ON = []
        for linea in tubo.stdout:
                element = linea.split(' ')

                if element[0] == 'Host':
                        ip = element[1]
                        lista_ON.append(ip)
                        if not ip in lista_Ant:
                                print "%15s:%10s %10s START" % ( ip,
time.strftime("%d/%m/%y"), time.strftime("%H:%M:%S") )
                        else:
                                lista_Ant.remove(ip)

        for ip_antigua in lista_Ant:
                print "%15s:%10s %10s STOP" % ( ip_antigua,
time.strftime("%d/%m/%y"), time.strftime("%H:%M:%S") )
        lista_Ant = lista_ON
#       print "## Buscando cambios en Red Local ##"

        del tubo
        time.sleep(5)
```

Processos: fork / execv

Les Unix eines tradicionals per generar processos fills és a través de fork i execv. Permeten 'bifurcar' programes i carregar nous programes en el procés actual.

Exemple-13: (13-exemple_fork.py

```
#! /usr/bin/python
#-*- coding: utf-8 -*-
import os,sys

print "hola"
pid = os.fork()
if pid!=0:
        print "programa pare"
else:
        print "programa fill
```

Exemple-13 (13-exemple_execv.py)

```
#! /usr/bin/python
#-*- coding: utf-8 -*-
import os,sys

print "hola"
pid = os.fork()
if pid!=0:
        print "programa pare", os.getpid()
        sys.exit(0)

print "programa fill", os.getpid()
os.execv("/usr/bin/ls",[""])

# aquest codi no s'executarà mai
print "tururut!"
```

Exercici-13 (13-execv.py)

Realitzar un procés pare que genera un procés fill i finalitza. El procés fill carrega un nou procés en el seu lloc i l'executa. usar com a nou procés l'ordre binària del sistema /usr/bin/ls.

```
#! /usr/bin/python
```

```
#-*- coding: utf-8 -*-
# @ edt Desembre 2014
# ASIX-M06
# --------------------------------------------
import os,sys

newProc="/usr/bin/ls"
pid=os.fork()
if pid!=0:
        print "programa pare, (%s,%s)" % (os.getpid(),pid)
        sys.exit(0)

print "...",os.getpid()
os.execv(newProc,[""])
sys.exit(0)
```

Exercici-14 (14-fork_execv_signal.py)

Realitzar un procés pare que genera un procés fill i finalitza. El procés fill carrega un nou procés en el seu lloc i l'executa. El proces que carrega el fill és un dimoni que es queda executant at infinitum i es governa via signals (el mateix que el exercici 11-signal.py)

```
#!/usr/bin/python
#-*- coding: utf-8 -*-
# @edt Desembre 2014
# --------------------------------------------
import os,sys

pid=os.fork()
if pid==0:
        print "daemon en execució..."
        os.execv("python 11-signal.py",[""])
print "(%s,%s)" % (os.getpid(),pid)
#os.wait()
sys.exit(:0)
```

```
#! /usr/bin/python
import signal, sys
def terminate(signum, frame):
        print 'Signal handler called with signal', signum
        print "closing program..."
        sys.exit(0)
def happyDay(signum, frame):
        print "Congratulations!", signum, "signal selected"
        print "Have a nice day!"
def sigReload(signum, frame):
        print "MATRIX reloaded!"
        print "where is sion?"

# Set the signal handler and a 5-second alarm
signal.signal(signal.SIGALRM, terminate)  #14
signal.signal(signal.SIGUSR1, terminate)  #10
signal.signal(signal.SIGUSR2, happyDay)   #12
```

```
signal.signal(signal.SIGHUP, sigReload)   # 1
signal.signal(signal.SIGINT, signal.SIG_IGN) # ignore ctrl+c #2
signal.alarm(6)

while True:
        pass
```

Altres temes a tractar dels processos:

- execv...
- wait
- wait all

Proposta Exercici-15 (15-proc_signal.py)

?? Construir un daemon que es quedi running a perpetuitat i genera per stdout missatges segons els senyals sigusr1, sigusr2, alarm?, sigterm el fa acabar indicant quants senyals a rebut. Sigalam li concedeix al programa 5 minuts de vida (a partir de que el rep, és pot anar prorrogant…)

```
```

Sockets

Un altre mecanisme de comunicació IPC és la utilització de sockets. Podem analitzar els sockets existents amb ordres com netstat, ss, iptraf, wiresark i nap. Amb la utilitat nc (netcat) podem observar també una comunicació client servidor simple

En el server:
$ nc -l 55500

En el client:
$ cat /etc/passwd | nc localhost 555000

En el server:
$ nc -l 55500

En el client:
$ nc localhost 555000
hola
bye
^d

Exemple-15 (15-exemple_sockets.py)

```
# Echo server program
import socket

HOST = ''          # Symbolic name meaning all available interfaces
PORT = 50007       # Arbitrary non-privileged port
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.bind((HOST, PORT))
s.listen(1)
conn, addr = s.accept()
print 'Connected by', addr
while True:
  data = conn.recv(1024)
  if not data: break
  conn.send(data)
conn.close()
```

```
# Echo client program
import socket

HOST = 'daring.cwi.nl'  # The remote host
PORT = 50007      # The same port as used by the server
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect((HOST, PORT))
```

```
s.send('Hello, world')
data = s.recv(1024)
s.close()
print 'Received', repr(data)
```

Exercici-15 (15-telnet_server.py, 15-telnet_server_client.py)

El servidor rep una petició del client (que és una ordre del bash) i l'executa i retorna el resultat al client. El servidor per executar l'ordre farà un pipe. Client capta el teclat un ls i l'envia al servidor, aquest l'executa i retorna el resultat al servidor. Servidor: crear el socket, lligar-ho amb bind (que li diu a quin port ha d'escoltar). Fer un listen per escoltar el port (aquí espera sol a rebre una connexió). Llavors l'accepta amb accept. Rebre amb recv i enviar amb send. Tancar amb close.

```
#!/usr/bin/python
# -*- coding:UTF-8 -*-
'''
# Sockets: Telnet server program
# Escola del treball de Barcelona
# ASI Hisi2 Curs 2009-2010
# Novembre 2009
'''
import socket
from subprocess import Popen, PIPE
import sys

HOST = ''
PORT = 50007
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.bind((HOST, PORT))
s.listen(1)
conn, addr = s.accept()
print 'Connected by', addr

while True:
   data = conn.recv(1024)
   if not data:
      print '***cerrando programa***'
      break
   command=data
   popen=Popen(command, shell=True, stdout=PIPE, stderr=PIPE)
   linia=popen.stdout.readline()
   while linia:
      conn.send(linia)
      linia=popen.stdout.readline()
   conn.send(chr(4)) #marca d fin

conn.close()
s.close()
sys.exit(0)
```

```
#!/usr/bin/python
```

```
# -*- coding:UTF-8 -*-
'''
# Sockets: Telnet client program
# Escola del treball de Barcelona
# ASI Hisi2 Curs 2009-2010
# Novembre 2009
'''
import socket
import sys

HOST = '192.168.0.90'    # The remote host
PORT = 50007          # The same port as used by the server
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect((HOST, PORT))

try:
   while True:
      comando=raw_input('Pon un comando:')
      if not comando:
         s.close()
         sys.exit(0)
      s.send(comando)
      data = s.recv(1024)
      while data:
         if chr(4) in data:
            sys.stdout.write(data[:-1])
            break
         sys.stdout.write(data)
         data = s.recv(1024)

except EOFError:
   s.close()
   sys.exit(0)
```

Exercici-15 (14-telnet_server_multi.py, 15-telnet_server_client_multi.py)

```
#!/usr/bin/python
#-*- coding: utf-8-*-
# Echo server multiconnection program
'''
# Sockets: Telnet server multiples connexionsa client program
# Escola del treball de Barcelona
# ASI Hisi2 Curs 2009-2010
# Novembre 2009
'''

from subprocess import Popen,PIPE
import socket
import sys
import select
import signal
import os
```

```
def final(signum,final):
   for conn in conns:
      conn.close()
   sys.exit(0)

#senyals per acabar el programa
#ctrl c es SIGINT es 2
signal.signal(2, signal.SIG_IGN)
#signal -15 es SIGTERM
signal.signal(15, final)

pid=os.getpid()
print "Pid del programa: ",pid

HOST = ''             # Symbolic name meaning all available interfaces
PORT = 50007          # Arbitrary non-privileged port
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.bind((HOST, PORT))
s.listen(1) #nº cua / FIFO

conns=[s]

while 1:
   actius,x,y=select.select(conns,[],[])
   for actual in actius:
      if actual==s:
         conn, addr = s.accept()
         print 'Connected by', addr
         conns.append(conn)
      else:
         data = actual.recv(1024)
         if not data:
            actual.close()
            conns.remove(actual)
         else:
            command=[data]
            tub=Popen(command,stdout=PIPE,shell=True)
            linia = tub.stdout.readline()
            while linia:
               actual.sendall(linia) #el sendall garanteix que ariiva tot
               linia = tub.stdout.readline()
            actual.sendall(chr(4),socket.MSG_DONTWAIT)

s.close()
sys.exit(0)
```

```
#!/usr/bin/python
#-*- coding: utf-8-*-
# Echo client program
'''
# Sockets: Telnet client program
# Escola del treball de Barcelona
# ASI Hisi2 Curs 2009-2010
```

```
# Novembre 2009
'''
import socket
import sys

HOST = '192.168.0.78'    # The remote host
PORT = 50007              # The same port as used by the server
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect((HOST, PORT))

try:
   command=str(raw_input('Introduce comando:'))

   while True:
      if command=='':    break
      s.sendall(command)     #enviar dades
      data = s.recv(1024)        #llegir dades que venen pel socket

      while data:
         if chr(4) not in data:
            sys.stdout.write(data)
            data=s.recv(1024)
         else:
            sys.stdout.write(data[:-1])
            break

      print 'Received', repr(command),' OK'
      command=str(raw_input('Introduce comando:'))

   s.close()

except EOFError:
   s.close()
   sys.exit(0)
```

Exemen Processos

EXAMEN 02: Processos: Volem fer un servidor que rebi fitxers enviats des d'un programa client i els guardi en el disc. El servidor acceptarà la connexió d'un sol client al mateix temps, després d'atendre'l acceptarà la connexió d'un altre i així succesivament fins que rebi un senyal per a finalitzar.Per simplificar només es transmetran fitxers de text. Servidor: prog_servidor Client: prog_cliente -H adreça_servidor -p port_servidor -f fitxer1 -f fitxer2 ...

Instruccions
● El resultat serà dos programes, anomenats servidor i client, guardats en el directori $HOME/examen_c5_sockets
● Es pot consultar tota mena de documentació escrita i online.
● L'examen és individual i no es pot compartir informació.
● S'han d'atendre, obligatòriament, tots els requeriments llevat dels del mode no interactiu
que són opcionals.

Enunciat del problema
Volem crear un servidor de ftp trivial. Haurem de crear, també, un client per comprovar que el
servidor funciona correctament.
Requeriments per al servidor:
● Se li passa com a argument (p n) el número de port que ha d'escoltar.
● Admet un sol client a l'hora.
● Accepta les següents ordres del client:
○ ls [path]. Retorna al client el llistat de fitxers del directori actiu.
○ get [path]nom_fitxer. Retorna al client el contingut de nom_fitxer.
● Quan el client acaba, el servidor es queda esperant una altra connexió.
● Acaba ordenadament quan rep un SIGTERM.

Requeriments per al client (mode interactiu):
● Se li passen com a arguments opcionals el host (H a.b.c.d) i el port (p n) amb qui s'ha de
connectar. Si es donen els dos arguments fa la connexió immediatament.
● Admet ordres de l'usuari per l'stdin:
○ connect host:port. Permès només si encara no ha fet la connexió.
○ ls [path]. Li envia l'ordre al servidor i espera un llistat de fitxers que mostra per stdout.
○ get [path]nom_fitxer. Li envia l'ordre al servidor i espera una seqüència de bytes que grava en el directori actiu amb el nom indicat.
○ quit. Tanca la connexió amb el servidor i acaba.
● Cada vegada que el programa espera alguna cosa de l'usuari li escriu un prompt (client_ftp_trivial>).

Requeriments addicionals per al client (mode no

interactiu) - opcional:
En invocar el client se li pot passar com arguments, a més de 'H a.b.c.d p n', una de les dues
ordres (c ls [path]|get [path]nom_fitxer) que espera el servidor, la qual executa i acaba.

Observacions:
Caldrà que t'inventis algun protocol de capa 7 per marcar el final del llistat i/o fitxer i, per tant,
de la transmissió.

Instruccions

- El resultat és un fitxer .py, anomenat *examen.py*, guardat en el directori *$HOME*, que conté el programa que resol l'enunciat expresat.
- Es pot consultar tota mena de documentació escrita i on-line.
- L'examen és individual i no es pot compartir informació.
- S'han d'atendre tots els requeriments de l'enunciat. Les decisions de disseny es documentaran dins del mateix script.

Enunciat

Es vol obtenir una relació dels usuaris del sistema en que apareix llistat login, uid, nom dels grups als quals pertany l'usuari; cada línia un usuari diferent.
La relació serà ordenada per algún d'aquest criteris: login (ascendent), uid (ascendent numèrica) o nombre de grups de l'usuari (descendent numèrica).
La invocació del programa serà de la forma:

```
$ ./examen.py -u fitxer_usuaris -g fitxer_grups -s {login|uid|ngrups}
```

L'anomenat fitxer_usuaris té l'estructura com el fitxer `/etc/passwd`.
L'anomenat fitxer_grups té l'estructura com el fitxer `/etc/group`.
Un requeriment indispensable del programa és que només pot recòrrer una vegada cada fitxer.
Internament el programa guardarà les dades que li calguin de cada usuari com un objecte d'una classe que definirà el programador.
S'ha de comprovar la validesa dels arguments proporcionats: que els fitxers existeixen, que el criteri d'ordenació és un dels especificats.
S'ha de documentar en el programa: l'autor, la data, l'objectiu del programa i la forma d'invocar-lo. A més tots els comentaris que calguin per explicar el codi.