Examen court de système d'exploitation

Département Informatique – IUT Villetaneuse

Lundi 18 mars 2013 – 90 min

Remarques : tous les documents de cours, td/tp sont autorisés. Le barème est indicatif. Écrire directement sur la copie.

PRÉNOM :
NOM :
GROUPE :

1 Questions de cours - 3 pts

Quels renseignements obtient on avec les options de compilation suivantes de GCC : -O3, -Wall, -Zipper ?

-03 : option d'optimisation de code (O = Optimize)
-Wall : on liste tous les warnings
-Zipper : c'est une invention

2 Expressions régulières - 3 pts

Soit le fichier suivant :

```
Mar 14 09:55:43 MacBook-Air-de-Christophe-Cerin kernel[0]: AppleUSBEthernet: Ethernet address 10:9a:dd:40:95:4a
Mar 14 09:55:46 MacBook-Air-de-Christophe-Cerin kernel[0]: Ethernet [AppleUSBEthernet]: Link down on en1
Mar 14 09:55:48 MacBook-Air-de-Christophe-Cerin kernel[0]: Ethernet [AppleUSBEthernet]: Link up on en1, 100-Megabit, Full-duplex
Mar 14 09:55:51 dyn225 kernel[0]: IOPMrootDomain: idle revert
Mar 14 09:55:51 dyn225 kernel[0]: AppleMCP89TMS::powerGatingDown(0): Done
Mar 14 09:55:53 dyn225 kernel[0]: R-state changed 2->0
Mar 14 10:21:34 dyn225 kernel[0]: CODE SIGNING: cs_invalid_page(0x1000): p=81966[GoogleSoftwareUp] clearing CS_VALID
Mar 14 11:20:17 dyn225 kernel[0]: CODE SIGNING: cs_invalid_page(0x1000): p=82195[GoogleSoftwareUp] clearing CS_VALID
Mar 14 12:19:00 dyn225 kernel[0]: CODE SIGNING: cs_invalid_page(0x1000): p=82402[GoogleSoftwareUp] clearing CS_VALID
Mar 14 13:17:43 dyn225 kernel[0]: CODE SIGNING: cs_invalid_page(0x1000): p=82631[GoogleSoftwareUp] clearing CS_VALID
```

1) Écrire une expression régulière qui permette de récupérer tous les événements s'étant déroulés à 9h du matin.
egrep '09 :[[:digit :]][[:digit :]] :[[:digit :]][[:digit :]]' signing.txt
2) Écrire une expression régulière qui permette de récupérer tous les événements s'étant déroulés entre 9h et 11h du matin (9 et 11 compris).
egrep '(09|10|11) :[[:digit :]][[:digit :]] :[[:digit :]][[:digit :]]' signing.txt

3) Écrire une expression régulière qui permette de récupérer tous les événements ne s'étant pas déroulés ni entre 9h et 11h du matin (9 et 11 compris) ni entre 13h et 14h (13 et 14 compris).
Attention : 11 et 14h non comprises

```
egrep '[^(09|10|13)]:[[:digit:]][[:digit:]]:[[:digit:]][[:digit:]]' signing.txt
```

Ne convient pas pour une autre raison. Expliquer pourquoi :

```
$ egrep '[^(09|10|13)]:[[:digit:]][[:digit:]]:[[:digit:]][[:digit:]]' signing.txt
Mar 14 09:55:43 MacBook-Air-de-Christophe-Cerin kernel[0]: AppleUSBEthernet:
       Ethernet address 10:9a:dd:40:95:4a
```

Il vaut donc mieux spécifier le motif à partir du début de ligne.

3 Langage de commande bash - 9 pts

La commande `last` retourne des informations sur les personnes s'étant connectées à une machine. Exemple d'informations retournées :

```
christophecerin  ttys002  Thu Mar 14 13:58   still logged in
christophecerin  ttys001  Thu Mar  7 09:46   still logged in
christophecerin  ttys002  Fri Mar  1 10:15 - 09:46 (5+23:30)
christophecerin  ttys006  Tue Feb 26 15:58 - 15:58  (00:00)
christophecerin  ttys005  Tue Feb 26 08:33   still logged in
christophecerin  ttys004  Mon Feb 25 21:25 - 15:58  (18:32)
christophecerin  ttys002  Sun Feb 24 10:09 - 15:58 (2+05:49)
christophecerin  ttys002  Fri Feb 22 17:01 - 17:02  (00:00)
christophecerin  ttys003  Tue Feb 12 08:58   still logged in
christophecerin  ttys001  Tue Feb 12 08:29 - 09:37 (22+01:08)
christophecerin  ttys001  Mon Feb 11 17:37 - 08:29  (14:51)
christophecerin  ttys001  Mon Feb 11 12:51 - 17:37  (04:45)
christophecerin  ttys001  Mon Feb  4 10:49 - 11:34  (00:45)
christophecerin  ttys002  Fri Feb  1 16:27 - 12:56 (10+20:28)
christophecerin  ttys001  Fri Feb  1 09:58 - 16:43  (06:45)
christophecerin  ttys000  Fri Feb  1 09:58   still logged in
christophecerin  console  Fri Feb  1 09:52   still logged in
```

On y remarque que l'utilisateur `christophecerin` a beaucoup de terminaux ouverts. . .certains depuis longtemps.
On vous demande d'écrire un script bash qui prend en parametre un nom d'utilisateur, qui fait la somme des heures de connection encore ouvertes de cet utilisateur et l'affiche.
On pourra utiliser ou s'inspirer des fonctions suivantes qui permettent de convertir des dates au format interne d'Unix. Ici on calcule le nombre de secondes écoulées entre le 1er janvier 1970 et la date passées en paramètre (cette valeur est un timestamp) et on retrouve la date associée à un timestamp.

```
date2stamp () {
    date --utc --date "$1" +%s
}

stamp2date (){
    date --utc --date "1970-01-01 $1 sec" "+%Y-%m-%d %T"
}

cerin@amman:~$ date2stamp "2006-10-01 15:00"
1159714800
cerin@amman:~$ stamp2date "1159714800"
2006-10-01 15:00:00

#!/bin/bash

# Ce programme calcule le nombre d'heures passées sur une machine par un
# utilisateur (spécifié par $1). Plus précisemment, on travaille à
# partir des informations générées par la commande last. Le résultat de
# cette commande est rangé dans le fichier /tmp/last. Le fichier
# /tmp/now contient la date à laquelle on a lancé la commande et cette
# information nous sert pour calculer le nombre de secondes écoulés
# depuis les dates de connections à la machine.

# D'un point de vue algorithmique, on parcourt le fichier /tmp/last
# ligne à ligne. On recherche sur cette ligne si la première colonne
# contient le nom de l'utilisateur recherché. Si c'est le cas, on
# s'interesse à la présence des mots clés "still logged in" sur la
# ligne. Ceci est un affaiblissement du problème. Quand on trouve les
# mots clés, on fabrique une date en récupérant les informations sur les
# colonnes 5,6 et 7 et cette information est au format de la fonction
# date2stamp qui retourne le nombre de seconde écoulées depuis le
# 1-1-1970 et la date issue des colonnes 5, 6, 7. Cette valeur est
# sommée dans la variable Res. En sortie de boucle on affiche Res, non
# pas sous forme de secondes, mais sous la forme H:M:S.

# En résumé, la solution proposée est un afaiblissement du problème
# général. nous vous invitons à enrichir la solution pour caper le
# problème dans sa plus grande généralité.

# Attention : ce code a été développé sous Linux. La commande last
# est différente, semble t'il, sous MacOS.

if [ "$#" -lt 1 ]
then
    echo "Il faut un seul argument (nom d'utilisateur)"
    exit 1
fi
```

```
date2stamp () {
    date --utc --date "$1" +%s
}
stamp2date (){
    date --utc --date "1970-01-01 $1 sec" "+%Y-%m-%d %T"
}

convertsecs() {
 ((h=${1}/3600))
 ((m=(${1}%3600)/60))
 ((s=${1}%60))
 printf "%02d:%02d:%02d\n" $h $m $s
}

last > /tmp/last
now=`date "+%Y-%m-%d %H:%M"`
#echo $now
date2stamp "$now">/tmp/now
now=`cat /tmp/now`
#echo "Maintenant : $now"

Res=0
while read line
do
#   echo "$line"
    w=`echo "$line" | cut -f 1 -d " "`
    if [ "$w" = "$1" ]
    then
       w=`echo "$line"|grep "still logged in"`
       if [ "$w" != "" ]
       then
           # trouve. On fabrique la date : ancienne methode
           # d1=`date "+%Y-%m-%d"`
   # d2=`echo "$line"|awk '{ print $7 }'`
   # d="$d1 $d2"
   #echo "$d1 $d2"
           # trouve . On fabrique la date : nouvelle methode
   d=`echo "$line"|awk '{ print 2013"-"$5"-"$6" "$7 }'`
   # Il faut substituer les mois par leurs numéros
           # Ainsi "Mar" => 03
           d=`echo "$d"|sed 's/-Mar-/-03-/'`
   # TODO : substitution pour les autres mois
   date2stamp "$d">/tmp/res
   y=`cat /tmp/res`
   Res=`echo "($now-$y+$Res)"|bc -l`
           echo "$d1 $d2 $now $y $Res"
```

```
       fi
    fi
done < /tmp/last
echo "Total en secondes : $Res"
echo -n "Format H:M:S : "
convertsecs "$Res"

#
# Exemple d'execution :
#
# cerin@amman:~$ last
# cerin    pts/3        lipn-ssh         Tue Mar 19 11:53 - 12:56  (01:03)
# cerin    pts/3        lipn-ssh         Tue Mar 19 08:42 - 11:02  (02:19)
# cerin    pts/3        lipn-ssh         Tue Mar 19 08:41 - 08:41  (00:00)
# cerin    pts/3        lipn-ssh         Mon Mar 18 20:52 - 21:52  (01:00)
# cerin    pts/3        lipn-ssh         Sat Mar 16 11:03 - 13:29  (02:25)
# saad     pts/4        localhost.locald Thu Mar 14 17:15 - 17:29  (00:13)
# cerin    pts/3        lipn-ssh         Thu Mar 14 14:03 - 17:31  (03:28)
# cerin    pts/2        :0               Thu Mar 14 12:58   still logged in
# saad     pts/2        lipn-ssh         Thu Mar  7 17:15 - 17:21  (00:05)
# cerin    pts/2        lipn-ssh         Thu Mar  7 15:24 - 17:15  (01:51)
# fortier  pts/2        lipn-ssh         Tue Mar  5 17:58 - 18:03  (00:05)
# root     pts/2        lipn-sf1         Tue Mar  5 17:56 - 17:57  (00:01)
# cerin    pts/1        :0               Tue Mar  5 15:23   still logged in
#
# wtmp begins Tue Mar  5 15:23:52 2013
#cerin@amman:~$ /bin/bash last.sh  cerin
#  1363703580 1363265880 437700
#  1363703580 1362496980 1644300
#Total en secondes : 1644300
#Format H:M:S : 456:45:00
```

4 Langage de commande bash - 5pts

1- Faites une explication du script ci dessous.
2- Compléter le corps de la fonction `verif` pour que le script ci-dessous affiche uniquement les noms des étudiants écrits tout en majuscule, ou bien tout en minuscule et de longueur au plus 8 caractères. On commencera par décrire l'algorithme suivi. Pour le fichier `ListeGroupes.csv` le programme affichera :

```
$ more ListeGroupes.csv
10501311,Mr,B0,DELNEUF,MAXIME
10602789,Mr,B1,DELPIERRE,JEREMY
10601320,Mr,A0,DESSAINT,FLORENT
10601814,Mr,A1,DUARTE,MATHIEU
```

```
10603349,Mr,A1,DUFAG,ALEXANDRE
10601894,Mr,A0,DUMONTIER,STEPHANE
10601882,Mr,A2,DURAND,PIERRE
10601945,Melle,B0,BELALIMAT,LEILA
$ /bin/bash majuscule.sh ListeGroupes.csv
------Fichier en entree: ListeGroupes.csv-------------
------Resultat :-------------
DURAND DUFAG DUARTE DESSAINT DELNEUF

#!/bin/bash

# L'algorithme qui est suivi est le suivant :
#
#
#
#
#
#
#
#
#
#
#
#
#
#
#
#
#
#

verif()
{
```

```
}
# Le format des fichiers (ici ListeGroupes.csv) est le suivant :
#10501311,Mr,B0,DELNEUF,MAXIME
#10602789,Mr,B1,DELPIERRE,JEREMY
#10601320,Mr,A0,DESSAINT,FLORENT
#10601814,Mr,A1,DUARTE,MATHIEU
#10603349,Mr,A1,DUFAG,ALEXANDRE
#10601894,Mr,A0,DUMONTIER,STEPHANE
#10601882,Mr,A2,DURAND,PIERRE
#10601945,Melle,B0,BELALIMAT,LEILA
#10603349,Mr,A1,DUFAG,ALEXANDRE

# Commentaires sur le programme principal :
#
#
#
#
#
#
#
#
#
#
#
#
#

if [ -e /tmp/solution ]; then
   rm /tmp/solution
fi
echo "------Fichier en entree: $1-------------"
cat < $1 | while true    #pour toutes les lignes du fichier
do
    read ligne
    if [ "$ligne" = "" ]; then break; fi
    # traitement de $ligne
    nom=‘echo $ligne | cut -d "," -f 4‘
```

```
    verif "$nom"
    if [ $? ]
    then
       echo -n "$nom " >> /tmp/solution
    fi
done

echo
echo "------Resultat :-------------"
cat /tmp/solution
rm /tmp/solution


#!/bin/bash

# L'algorithme qui est suivi pour la fonction test() est le suivant :
# on teste la longueur. Si elle est plus grande que 8, on sort avec
# "faux". Sinon, on parcourt une premiere fois la chaine passee
# en parametre et si on ne detecte que des minuscules alors on peut
# retourner 1 ; sinon  on doit verifier si la chaine est une chaine
# de majuscule et retourner 1 si c'est le cas et 0 sinon.
#

verif()
{
 bool=1
 if [ ${#1} -leq 8 ]
 then
    bool=0
 else
    for ((i=0 ; i<${#1}; i++ ))
    do
      c=${1:$i:1}
      if [ "$c" \> "z" -a "$c" \< "a" ]
      then
         bool=0;break;
      fi
    done
    # si bool=1 c'est que la chaine est en minuscule
    if [ $bool -eq 1 ]
    then
      return 1;
    fi
    # si bool=0 c'est que la chaine n'est pas en minuscule
    # et on verifie si elle comporte que des majuscules
    bool=1
    for ((i=0 ; i<${#1}; i++ ))
    do
```

```
      c=${1:$i:1}
      if [ "$c" \> "Z" -a "$c" \< "A" ]
      then
         bool=0;break;
      fi
    done
 fi
 # si bool=1 c'est que la chaine est en majuscule
 if [ $bool -eq 1 ]
 then
   return 1;
 else
   return 0
 fi

}

# Le format des fichiers (ici ListeGroupes.csv) est le suivant :
#10501311,Mr,B0,DELNEUF,MAXIME
#10602789,Mr,B1,DELPIERRE,JEREMY
#10601320,Mr,A0,DESSAINT,FLORENT
#10601814,Mr,A1,DUARTE,MATHIEU
#10603349,Mr,A1,DUFAG,ALEXANDRE
#10601894,Mr,A0,DUMONTIER,STEPHANE
#10601882,Mr,A2,DURAND,PIERRE
#10601945,Melle,B0,BELALIMAT,LEILA
#10603349,Mr,A1,DUFAG,ALEXANDRE

if [ -e /tmp/solution ]; then
   rm /tmp/solution
fi
echo "------Fichier en entree: $1-------------"
cat < $1 | while true    #pour toutes les lignes du fichier
do
    read ligne
    if [ "$ligne" = "" ]; then break; fi
    # traitement de $ligne
    nom=`echo $ligne | cut -d "," -f 4`
    verif "$nom"
    if [ $? ]
    then
       echo -n "$nom " >> /tmp/solution
    fi
done

echo
echo "------Resultat :-------------"
```

```
cat /tmp/solution
rm /tmp/solution
```

Remarque : pour détecter si une chaîne est composée que de majuscules, on peut aussi s'inspirer du code suivant qui utilise une expression régulière :

```
#!/bin/bash

echo "La chaine en entree : $1"
j=‘echo "$1" | egrep "^[[:upper:]]+$"‘
echo $j

if [ "$j" != "" ]
then
    echo "la chaine contient que des majuscules";
else
    echo "faux : la chaine ne contient pas que des majuscules";
fi
```