

Examen de système d'exploitation

Semestre décalé

Département d'informatique IUT Villetaneuse

lundi 10 décembre 2007

(Click here to load the video)

Remarques : tous les documents de cours, td/tp sont autorisés. Le barème est indicatif.

1 Makefile 6 pts

Un projet est organisé selon 6 fichiers : f4.h qui contient les constantes utilisées par tous les programmes, f1.c qui contient une première fonction, f2.c qui contient une deuxième fonction, f3.c qui contient une troisième fonction, f4.c qui contient une fonction main() qui implémente une interface utilisateur en mode texte et f5.c qui contient une fonction main() et qui implémente une interface utilisateur en mode graphique.

Expliquer pourquoi la compilation du fichier Makefile suivant ne va pas bien se passer :

- Comme f4.c et f5.c contiennent une fonction de nom main(), le compilateur verra un conflit et ne pourra pas construire l'exécutable

```
objects = f1.o f2.o f3.o f4.o f5.o
```

```
program = matrice
```

```

CFLAGS = -Wall -O4

all: $(program)

$(program) : $(objects)
    gcc $(CFLAGS) -o $@ $^

%.o: %.c
    gcc $(CFLAGS) -c $<

$(objects): f4.h

clean :
    rm -f $(objects)
    rm -f $(program)

```

Réécrire le fichier Makefile pour qu'il puisse générer deux exécutables, l'un correspondant à l'interface utilisateur en mode texte, l'autre correspondant à l'interface utilisateur en mode graphique. On ne peut pas avoir les deux interfaces dans le même programme.

- On décompose en deux phases la compilation : une phase pour compiler f4.c et une autre pour compiler f5.c comme suit :

```

objects1 = f1.o f2.o f3.o f4.o
objects2 = f1.o f2.o f3.o f5.o

program1 = matrice1
program2 = matrice2

CFLAGS = -Wall -O4

all: $(program1) $(program2)

$(program1) : $(objects1)
    gcc $(CFLAGS) -o $@ $^

$(program2) : $(objects2)
    gcc $(CFLAGS) -o $@ $^

%.o: %.c
    gcc $(CFLAGS) -c $<

```

```
$(objects1): f4.h
```

```
$(objects2): f4.h
```

```
clean :
```

```
    rm -f $(objects1) $(objects2)
```

```
    rm -f $(program1) $(program2)
```

2 Langage de commande bash - 6 pts

Écrire un programme Bash qui prend en paramètre deux entiers i et j ($i \leq j$) et qui affiche des X entre la i ème colonne et la j ème colonne de la ligne 1 du terminal texte. On utilisera :

```
tput cup 0 0
```

Send the sequence to move the cursor to row 0, column 0 (the upper left corner of the screen, usually known as the “home” cursor position). tput m n: move the cursor at position (m,n)

```
tput clear
```

Echo the clear-screen sequence for the current terminal.

```
tput cols
```

Print the number of columns for the current terminal.

- Un algorithme possible est de commencer par écrire $i-1$ blancs sur la ligne, puis $j-i+1$ caractères 'X'. Un autre algorithme est de déplacer le curseur sur la colonne i , puis d'afficher $j-i+1$ caractères 'X'. C'est cet algorithme que nous codons ci-dessous :

```
#!/bin/bash
```

```
if [ $1 -gt $2 ]
```

```
then
```

```
    echo "le deuxieme parametre doit etre superieur ou egal au premier"
```

```
    exit
```

```
fi
```

```
# on efface l'ecran
```

```
tput clear
```

```
# on affiche j-i+1 caracteres 'X'
```

```

for ((i=$1 ; $i<=$2; i=$((i+1))))
do
    tput cup 1 $i
    echo -n 'X'
done

# on passe a la ligne
echo

```

3 Langage de commande bash - 8pts

Un fichier texte est organisé en deux colonnes contenant des entiers. On veut produire sur la sortie standard (le terminal) un fichier à cinq colonnes ou la troisième colonne contiendra la somme des deux premières colonnes, la quatrième colonne contiendra le plus petit entier présent dans le fichier initial et la cinquième colonne contiendra le plus grand entier du fichier initial. Exemple :

```

3 4          3 4 7 1 7
1 2 devient 1 2 3 1 7
7 6          7 6 13 1 7

```

Écrire le programme Bash qui s'inspirera du code suivant :

```

#!/bin/bash

echo ‘‘-----Fichier en entree: $1-----’’
cat < $1 | while true
do
    read ligne
    # traiter la ligne
    ....
    ....
done

```

Le code précédent permet de faire un passage sur le fichier texte \$1. Dans votre cas vous allez avoir besoin d'un deuxième passage sur le fichier texte. N'oubliez pas d'expliquer dans le détail ce que vous faites.

```

#!/bin/bash

# On verifie si le fichier existe
if [ $# -lt 1 ]
then
    echo ‘‘Il faut donner un nom de fichier en argument’’
    exit
fi

if !(test -f $1)
then
    echo ‘‘Fichier $1 inconnu !’’
    exit
fi

# on efface l’ecran
tput clear

# on commence par faire une premiere passe pour
# isoler le max et le min. On suppose que l’entree
# est constituee d’entiers strictement positifs.

min=0
max=0
i=0

echo ‘‘-----Fichier en entree: $1-----’’

cat < $1 | while true
do
    read ligne
    # traiter la ligne
    if [ ‘‘$ligne’’ = ‘‘’’ ]; then break; fi
    one='echo $ligne | cut -d ‘‘ ’’ -f 1'
    two='echo $ligne | cut -d ‘‘ ’’ -f 2'
    if [ $i -eq 0 ] ##### Expliquez
    then
        max=$two
        min=$one
        i=$((i+12)) ##### Expliquez
    fi

```

```

if [ $one -gt $max ]
then
    max=$one
fi
if [ $two -gt $max ]
then
    max=$two
fi
if [ $one -lt $min ]
then
    min=$one
fi
if [ $two -lt $min ]
then
    min=$two
fi
# on ecrit le max et le min dans un fichier
echo "$max $min" > /tmp/bidon
done

# On effectue la deuxieme passe
# on commence par recuperer le max et le min

max='cat /tmp/bidon |cut -d " " -f 1'
min='cat /tmp/bidon |cut -d " " -f 2'

# on reprend
cat < $1 | while true
do
    read ligne
    # traiter la ligne
    if [ "$ligne" = "" ]; then break; fi
    one='echo $ligne | cut -d " " -f 1'
    two='echo $ligne | cut -d " " -f 2'
    echo $one $two $((two+one)) $min $max
done

# Exemple de resultat d'execution :
# -----Fichier en entree: e.txt-----
# 1 2 3 1 9

```

1 3 4 1 9
4 2 6 1 9
5 6 11 1 9
7 3 10 1 9
3 9 12 1 9