PARTIEL 2017-2018 - L3 (1h30)

Architecture des Ordinateurs et Systèmes d'Exploitation

Exercice 1: Conversions

Question 1.1: **(1 point) Convertir ces nombres décimaux en binaires sur 8 bits : `125`, `-74`**

125 : % 0111 1101 ($ 7D) -74 : % 1011 0110 ($ B6)

Question 1.2: **(2 points) Convertir ces nombres binaires (8 bits signés et non signés) en décimaux : `%1011 1100`, `%1101 1010`**

% 1011 1100 : -68 ou 188 % 1101 1010 : 218 ou -38

Exercice 2: Compilation

Question 2.1: **(2 points) Expliquer succinctement les principales étapes de compilation. (une ou deux phrase(s) par étape).**

1. Pré-compilation : exécution des directives/macros de précompilation + retrait des commentaires
2. Compilation : transformation du langage haut niveau source en assembleur dédié au processeur cible
3. Assemblage : transformation du langage assembleur en langage machine (conservation des noms de fonctions), création de fichiers "objets"
4. Link Edit : concaténation des fichiers objets et résolution des adresses de fonctions disponibles et variables globales

Question 2.2: **BONUS : (2 points) Écrivez le code C résultant de l'opération de pré-compilation sur `main.c`, puis expliquez pourquoi ce code ne pourra pas générer à lui tout seul un exécutable (et comment faire pour corriger cela).**

```
int my_read(int fd, int buf, char *str);
int my_write(int fd, int buf, char *str);
void *malloc(size_t size);
void free(void ptr);

int   main(void)
{
  char *tab;

  tab = malloc(4);
  my_read(0, 3, tab);
  my_write(1, 4, "Test");
```

```
  free(tab);
  return (0);
}
```

main.c

Exercice 3: Architecture

Question 3.1: (1 point) Expliquer succinctement comment le processeur lit une donnée en mémoire (comment les bus sont utilisés).

1. Le processeur met l'adresse qu'il souhaite atteindre sur le bus d'adresse (en parallèle avec 2)
2. Le processeur met sur le bus de contrôle qu'il souhaite lire la mémoire (en parallèle avec 1)
3. La mémoire charge l'adresse demandée et récupère la donnée
4. Le processeur indique sur le bus de contrôle qu'il est prêt à recevoir la donnée
5. La mémoire prend la donnée et l'envoie sur le bus de données

Question 3.2: (1 point) Donner un exemple de code provoquant un overflow (le langage peut être du C, sh, python, ou autre langage de développement).

```
int main(void)
{
  int i = 0;

  while (1)
  {
    i = i + 1;
  }
}
```

overflow.c

```
#! /bin/sh

exit 320
```

overflow.sh

Question 3.3: (2 points) Expliquer succinctement quelles sont les fonctions du processeur, de la mémoire, et des périphériques dans un ordinateur.

— Processeur (CPU) : exécuter des instructions/du code tout en manipulant des données entre la mémoire et les périphériques
— Mémoire (RAM) : stocker et envoyer des données à des adresses précises
— Périphérique (Device) : intérargir avec le monde extérieur à l'ordinateur voire avec le monde physique par des interfaces homme-machine

Exercice 4: Système d'Exploitation

Question 4.1: (2 points) Expliquer la notion de "temps partagé", citer le composant du système qui gère cette notion, et illustrer avec un exemple où le temps est partagé puis un autre exemple où le temps n'est pas partagé.

L'ordonnanceur permet de gérer le temps partagé dans un système d'exploitation.

La notion de temps partagé est la capacité d'un système d'exploitation à partager le temps processeur entre plusieurs tâches différentes. Ces tâches ne fonctionnent peut être pas exactement au même moment, mais elles sont enregistrées dans l'ordonnanceur, et elles passent alternativement en mode exécution. Là où un système d'exploitation coopératif va aider des processus à partager le temps quand ceux-ci le permettent (lors d'une lecture/écriture ou tout autre appel système), un système d'exploitation préemptif va stopper *de force* les processus et partager *de force* le temps processeur.

Sur un PC classique moderne, écouter de la musique et naviguer sur internet en même temps est un exemple où le temps est partagé : le processus jouant la musique et celui navigant sur internet se partagent le temps processeur. Un programme batch tournant tout seul sur une machine, et empêchant tout autre programme de fonctionner tant qu'il n'est pas terminé, est un exemple où le temps n'est pas partagé.

Question 4.2: (2 points) Détailler les différentes parties de l'espace d'adressage d'un processus, puis donner 2 exemples différents où un accès mémoire ne fonctionnera pas (et expliquer pourquoi).

— Pile/Stack : contient les arguments passés aux fonctions
— Tas/Heap : les données allouées dynamiquement (malloc, new, ...)
— BSS (Block Started by Symbol) : variables non initialisées et variables à 0
— Data Segment : données initialisées (copiées de RODATA vers DATA au démarrage du programme)
— RODATA (Read-Only Data) : les données initialisées et les chaînes de caractères pré-déclarées
— Text/Code Segment : contient le code en langage machine du programme

Segmentation Fault (SegFault) : on demande à accéder à une zone mémoire non allouée/réservée par malloc (brk/sbrk) dans le tas, le système refuse et lance une erreur.

Bus Error : on essaye d'écrire dans une zone mémoire marquée comme Read-Only, le système refuse et lance une erreur.

Question 4.3: (2 points) Citer 5 informations contenues dans un i-node.

- La taille (en octets)
- Les adresses des blocs utilisés sur le disque
- L'identification du propriétaire du fichier
- Les droits d'accès (chmod et ACL)
- Le type du fichier (ordinaire, spécial, ...)
- Un compteur de liens
- Les dates d'opérations principales (création, modification, consultation)

Question 4.4: (1 point) Donnez 3 raisons différentes qui peuvent empêcher de créer un fichier dans un dossier.

- Pas le droit de créer fichier
- Périphériques visé pas disponible
- Pas d'espace disque (tous les blocs sont réservés)
- Tous les i-nodes sont utilisés
- Le dossier a été supprimé avant que la création du fichier n'ait été effective

Question 4.5: (2 points) Expliquer ce que fait chaque ligne du script. En admettant qu'une entreprise génère des transactions jusqu'à 20h, indiquez l'heure minimale et l'heure maximale pour lancer ce script et traiter les transactions du jour même.

```
1 #! /bin/sh
2
3 OUTFILE=`mktemp fileXXXX`
4 cat transactions.log | tr "[:upper:]" "[:lower:]" | sort -t";" -k1 -o
    $OUTFILE
5 SUFFIXE=`date +%Y-%m-%d`
6 NAME=comptabilite-${SUFFIXE}
7 grep "$SUFFIXE" $OUTFILE > $NAME
8 rm -f $OUTFILE
```

script.sh

- Ligne 1 : On appelle l'interpréteur de script shell `sh`
- Ligne 3 : On crée un fichier temporaire dont le préfixe est "file" et qui est fini par 4 caractères aléatoires, le nom sera sauvegardé dans la variable OUTFILE
- Ligne 4 : On lit le fichier "transactions.log" et on envoie son contenu sur la sortie standard. Celle-ci est redirigée vers l'entrée de "tr" qui transorme les manjuscules en minuscules et redirige aussi sa sortie standard vers la commande suivante. Finalement, "sort" va trier selon le 1er champs, les champs sont séparés par des point virgules ";", et la sortie sera renvoyée vers le fichier temporaire désigné par la variable OUTFILE
- Ligne 5 : La variable SUFFIXE va prendre comme valeur la date du jour formattée de l'année, du mois, puis du jour séparés par des tirets
- Ligne 6 : La variable NAME va prendre comme valeur le mot "comptabilite" suivi d'un tiret et de la date précédemment stockée dans SUFFIXE
- Ligne 7 : On va chercher avec grep toutes les lignes qui contiennent la date du jour dans le fichier désigné par la variable OUTFILE, puis les mettre dans le fichier désigné par la variable NAME

— Ligne 8 : On supprime le fichier temporaire désigné par la variable OUTFILE

De façon générale, on va extraire les lignes du jour actuel du fichier "transactions.log", les mettre en minuscule, et trier selon le champ 1 (séparé par des " ;"), et rediriger ces lignes triées dans un fichier comptabilite quotidien.

L'heure minimale pour lancer ce script sera après 20h, et l'heure maximale sera minuit/23h59 du jour-même.

Question 4.6: **(2 points) Écrire un script qui listera tous les fichiers commençant par le nom "comptabilite-" suivi d'une date (format aaaa-mm-jj), puis afficher pour chaque fichier la colonne contenant le montant de chacune des lignes, et renvoyer le tout vers le fichier passé en premier paramètre du script.**

```
ID Transaction ; Date ; Montant ; Numero Carte Fidelite ; Remarques
```

Format des fichiers de transactions

```
1445;2017-12-08;42;1001;fait a 11h01
1446;2017-12-08;200;1002;fait a 11h11
1447;2017-12-08;100;1001;fait a 12h04
```

comptabilite-2017-12-08

```
#! /bin/sh

find . -name "comptabilite-[0-9]\{4\}-[0-9]\{2\}-[0-9]\{2\}" -exec cut -f
    3 {} \; >> $1
```

script.sh

```
#! /bin/sh

ls | grep "comptabilite-[0-9]\{4\}-[0-9]\{2\}-[0-9]\{2\}" | awk '{ print
   $3 ; }'
```

script.sh