— Vous rendrez votre contrôle dans **un seul** fichier, nommé `login.c` (vous remplacerez `login` par votre identifiant).
— Ce fichier DOIT compiler sans erreur avec les options `-Wall -W -Werror`. Un (1) point sera accordé si votre fichier compile sans erreur, quelque soit son contenu.

On se propose dans cet exercice d'implémenter un modèle de programmation parallèle à l'aide d'un tube. Un processus principal envoi des travaux à effectuer dans un tube. D'autres processus vont lire les travaux disponibles depuis ce tube, effectuer le travail demandé et recommencer, tant qu'il y a des travaux à lire depuis le tube.

Dans notre cas, un travail consistera à afficher des détails sur un fichier (son type, son nom et sa taille). Pour définir un travail, vous allez devoir déclarer la structure suivante dans votre programme :

```
#define FILENAME_SIZE 256
typedef struct {
   char fichier[FILENAME_SIZE];
} job_t;
```

Envoyer un travail consistera donc à envoyer les octets d'une variable de ce type dans un tube. Recevoir un travail consistera, à l'inverse, à recevoir les octets permettant de construire une telle structure depuis le tube.

Q 1. Implémentez la fonction :

```
int afficher_details(const char *fichier);
```

qui affiche les détails d'un fichier passé en paramètre avec le format suivant :

```
t,nom_du_fichier,taille_en_octet
```

où `t` doit être le caractère :
— `'-'` pour un fichier régulier ;
— `'d'` pour un fichier répertoire ;
— `'c'` pour un fichier spécial de type caractère ;
— `'b'` pour un fichier spécial de type bloc (pour ces deux derniers cas, vous pouvez tester votre fonction avec les fichiers `/dev/tty` et `/dev/sda` respectivement) ;

Voici des exemples d'affichage d'une telle fonction sur quelques fichiers :

```
b,/dev/sda,0
c,/dev/tty,0
d,/etc/a2ps,4096
-,/etc/bash.bashrc,603
```

Votre fonction doit utiliser l'appel système `stat` pour obtenir les informations à afficher.

Q 2. Implémentez la fonction :

```
int envoyer_job(int fd, const job_t *job);
```

qui envoie le travail décrit par le pointeur `job` dans le fichier décrit par le descripteur `fd` à l'aide de la fonction `write`. En cas d'erreur, la fonction doit afficher un message explicite et retourner -1. Elle devra retourner 0 si tout s'est bien déroulé.

Q 3. Implémentez la fonction :

```
int recevoir_job(int fd, job_t *job);
```

qui lit un travail depuis le fichier décrit par le descripteur `fd` et le stocke à l'adresse `job` à l'aide de la fonction `read`. En cas d'erreur, la fonction doit afficher un message d'erreur explicite et retourner -1. Elle devra retourner 0 si tout s'est bien déroulé.

Q 4. Implémentez la fonction :

```
void traitement_fils(int fd);
```

qui exécute l'algorithme suivant :

1. recevoir un travail depuis le descripteur `fd` ;
2. afficher les détails du fichier dont le nom est contenu dans le travail ;
3. retour en 1, tant qu'il y a du travail dans le descripteur `fd` ;
4. afficher un message indiquant la terminaison ;
5. quitter le processus par un appel à `exit`.

Attention : cette fonction ne **doit pas** créer de processus ni de tube !

Q 5. Implémentez la fonction

```
int creer_fils(int tube[2]);
```

qui crée un nouveau processus. Le processus père retournera le `pid` du fils créé. Le processus fils fermera le descripteur associé à l'entrée du tube `tube` passé en paramètre et lancera le traitement du fils en lui passant en paramètre la sortie du tube `tube`.

Q 6. Implémentez la fonction `main` de votre programme.

Cette dernière prendra en paramètre au moins 2 arguments. Le premier sera un entier et représentera le nombre de processus fils à créer. Les arguments suivants seront des noms de fichiers qui devront être envoyé comme travaux dans un tube.

La fonction devra donc :

— vérifier le nombre d'arguments passés en paramètre ;
— créer autant de processus fils qu'indiqué par le premier arguments (fonction `atoi`) ;
— pour chacun des arguments restant, envoyer un travail correspondant dans le tube ;
— attendre la fin de tous les processus fils (l'ordre n'est pas important) ;
— afficher un message indiquant la fin du traitement.

Exemple d'exécution du programme complet :

```
$ ./tp 3 /dev/sda /dev/tty /etc/*
b,/dev/sda,0
c,/dev/tty,0
-,/etc/abcde.conf,13480
-,/etc/adjtime,16
d,/etc/xml,4096
d,/etc/zsh,4096
Fils de pid 11975: Fini
Fils de pid 11974: Fini
Fils de pid 11976: Fini
Pere: fils de pid 11974 terminé correctement
Pere: fils de pid 11975 terminé correctement
Pere: fils de pid 11976 terminé correctement
Pere de pid 11973: fini
```

Q 7. Bonus

Modifiez la fonction `afficher_details` pour que, si le fichier est un répertoire, elle parcourt récursivement son contenu pour afficher les détails de tous les fichiers situé dans ce répertoire et dans les sous-répertoires. Vous devrez pour cela utiliser les fonction `opendir`, `readdir` et `closedir`.

Dernières recommandations :

— Pensez à vérifiez les erreurs des appels aux fonctions système ;
— Bien que l'exercice forme un tout logique, chaque question peut être traitée de façon totalement INDÉPENDANTE et sera corrigée indépendemment du fonctionnement global du programme. N'hésitez pas à passer une question qui vous pose problème en laissant la fonction vide ou en utilisant printf pour simuler l'affichage qu'elle est censé produire.
— N'oubliez pas que chaque fonction demandée peut (et DOIT) être réutilisée dans certaines des questions suivantes. Vous pouvez obtenir tous les points d'une question, même si vous n'avez pas fait les autres.
— COMMENTEZ vos programmes, pensez aux correcteurs :)
— NE QUITTEZ PAS LA SALLE AVANT DE VOUS ÊTRE ASSURÉS AUPRÈS DE VOTRE ENSEIGNANT DU RENDU EFFECTIF DE VOTRE EXERCICE !
