1 Objectifs:

Le **shell** est chargé de faire l'intermédiaire entre le système d'exploitation et l'utilisateur grâce aux lignes de commandes saisies par ce dernier. Son rôle consiste ainsi à lire la ligne de commande, interpréter sa signification, exécuter la commande, puis retourner le résultat sur les sorties.
Le shell est ainsi un fichier exécutable chargé d'interpréter les commandes, de les transmettre au système et de retourner le résultat.

Une ligne de commande est une chaîne de caractère constituée d'une commande, (correspondant soit à un fichier exécutable du système, soit à une fonction interne du shell) et d'arguments optionnels :

```
ls -al /home/pi/
```

ls est la commande **-al** et **/home/if/** représentent les arguments.

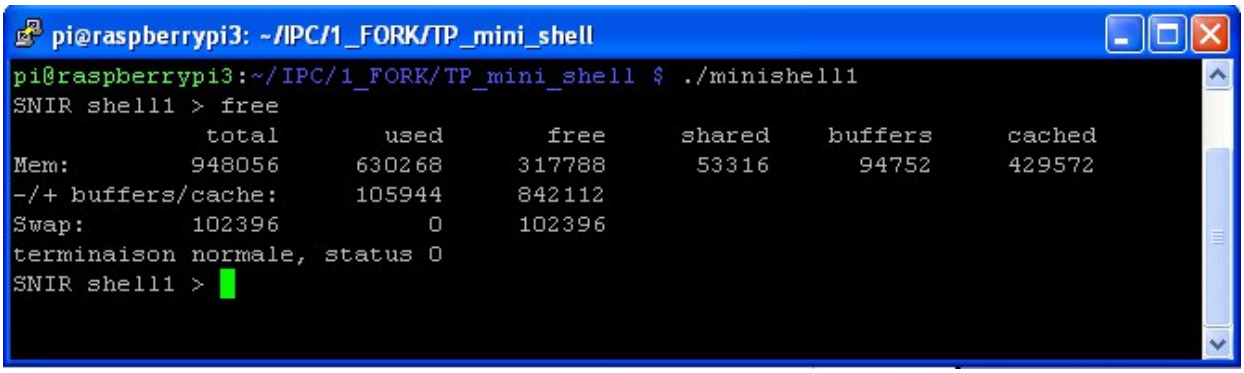
Le but de ce TP est de programmer un **shell minimal en C**.

2 Exécution de programmes au premier plan :

Notre shell sera une simple boucle d'interaction qui affiche une invite de commande **SNIR shell >**, lit une ligne entrée au clavier (**fgets**), l'exécute, et recommence. Le shell quitte quand l'utilisateur tape **Contrôle+D** à la place d'une commande.

3 Travail demandé :

1) Dans un premier temps, on suppose que la commande est simplement un nom de programme à exécuter dans un processus séparé (**fork()** et **execlp()**). Le s*hell* doit attendre que le programme se termine (**waitpid()**) avant de rendre la main à l'utilisateur.



Attention: **waitpid()** peut échouer avec l'erreur **EINTR**, auquel cas il faut renouveler l'appel à waitpid().
Attention: le **nom du programme apparaît deux fois** dans **execlp()** (une fois en tant que programme à exécuter et une fois en tant qu'argument 0 du programme à exécuter). De plus la liste d'arguments doit obligatoirement se terminer par **NULL**.

2) On suppose maintenant que la ligne de commande peut contenir un nombre arbitraire d'arguments, séparés par un ou plusieurs espaces.

Afin de ne pas passer trop de temps sur le problème du découpage d'une ligne en mots, on vous propose une solution toute faite (voir en annexe à la fin du TP).

Modifier votre programme pour découper la ligne de commande et passer le tableau des arguments à **execlp().**

3) Vous allez à partir de maintenant effectuer à partir de votre shell les tests suivants :

```
SNIR shell1 > ps
  PID TTY          TIME CMD
16179 pts/0    00:00:00 bash
16314 pts/0    00:00:00 minishell1
16315 pts/0    00:00:00 ps
terminaison normale, status 0
SNIR shell1 >
```

```
SNIR shell1 > cd ..
impossible d'éxecuter "cd .." (No such file or directory)
terminaison normale, status 1
SNIR shell1 >
```

- Que se passe t'il si on exécute la commande cd . Pour vous aider à comprendre cette situation vous pouvez à partir du shell bash exécuter les commandes **type ps** puis **type cd**.

Modifier votre programme pour que la commande **cd** soit exécutée. Vous ajouterez une commande **exit** pour quitter proprement votre mini shell.

4) Que se passe t'il si on interrompt la commande **cat** avec **contrôle C** ?

```
SNIR shell1 > cat
essai
essai
^C
```

Modifiez votre *shell* pour que, si une commande est en cours d'exécution, un appui sur Contrôle+C interrompe la commande et redonne la main à l'utilisateur sans toutefois quitter le mini *shell*. On utilisera pour cela **sigaction()**.

5) Ajoutez à votre *shell* la redirection de la sortie standard. Si le dernier argument commence par un caractère **>**, on suppose que le reste du mot est le nom du fichier où rediriger la sortie de la commande à exécuter (voir **dup2()**).

Que se passe-t-il si la commande écrit sur la sortie d'erreur?

4 Annexe découpe ligne:

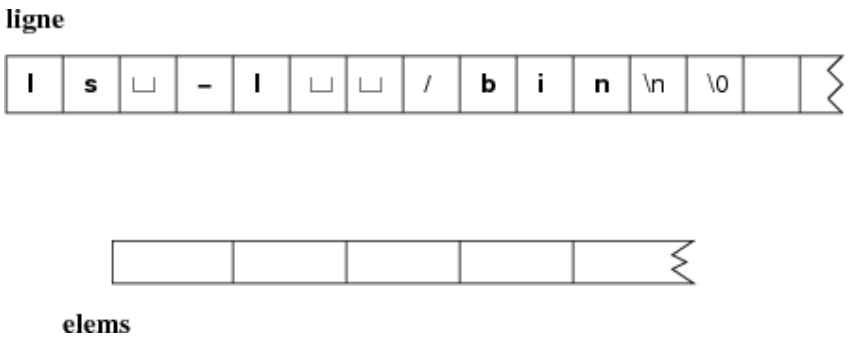
On propose une fonction **decoupe** permettant de découper une ligne en mots:

L'entrée de **decoupe** est la chaîne à découper, contenue dans la variable globale **ligne**.
La sortie de **decoupe** sera une liste de pointeurs vers le début de chaque mot. La liste est stockée dans le tableau global **elems** de **MAXELEMS** pointeurs et est terminée par le pointeur **NULL**.

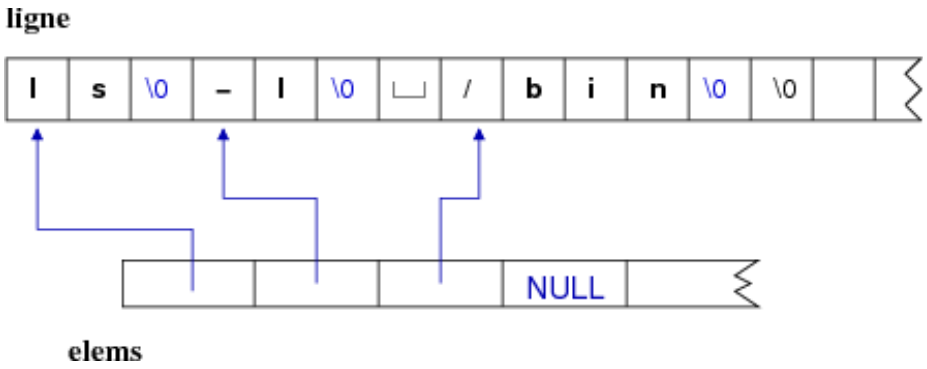
```
char ligne[4096];
#include <ctype.h>
#define MAXELEMS 32
char* elems[MAXELEMS];

void decoupe()
{
  char* debut = ligne;
  int i;
  for (i=0; i<MAXELEMS-1; i++) {
    /* saute les espaces */
    while (*debut && isspace(*debut)) debut++;
    /* fin de ligne ? */
    if (!*debut) break;
    /* on se souvient du début de ce mot */
    elems[i] = debut;
    /* cherche la fin du mot */
    while (*debut && !isspace(*debut)) debut++; /* saute le mot */
    /* termine le mot par un \0 et passe au suivant */
    if (*debut) { *debut = 0; debut++; }
  }
  elems[i] = NULL;
}
```

Lors de son parcours de **ligne**, la fonction **decoupe** transforme les espaces et retours à la ligne (voir **isspace()**) en fin de mots en caractère **\0** et fait pointer **elems** à l'intérieur de **ligne**. L'avantage de cette méthode est qu'il n'est pas nécessaire d'allouer de la mémoire supplémentaire pour stocker les mots isolés de **ligne**.

Supposons que, avant l'appel à **decoupe**, **ligne** contienne la ligne **ls -l /bin**: (avec deux espaces en -l et /bin)



Au retour de **decoupe**, on aura alors:



Ainsi,

elems[0] pointe sur la chaîne **ls**,

elems[1] pointe sur la chaîne **-l** et

elems[2] pointe sur la chaîne **/bin**.

Toutes ces chaînes sont bien terminées par le caractère **\0**.