Ecriture d’une commande en C.

Objectif : wcl2

Nous souhaitons écrire en C une commande qui effectue le même travail que wc:

- écrire le programme C
- première approche de la gestion des flux en C
- utiliser les primitives système UNIX en C : ex de la gestion des entrées sorties
- implémenter un algorithme qui compte des mots
- réaliser une commande UNIX, qui respecte les standards de l'utilisation du shell.
- Faire le lien entre le C Unix et le shell

Nous appellerons cette commande : wcl2

Plan

1ère étape : écrire une commande avec arguments
2 ème étape : vérification des arguments
3 ème étape : gestion du flux d'erreur (fprintf)
3 ème étape : lecture du fichier (primitive système)
4 ème étape : algorithme
5 ème étape : gestion de la valeur de retour
6 ème étape : améliorations (entrée standard)

Les paramètres de notre commande : wcl2

On souhaite que notre commande s'utilise de la façon suivante :

wcl2 nomdemonfichier

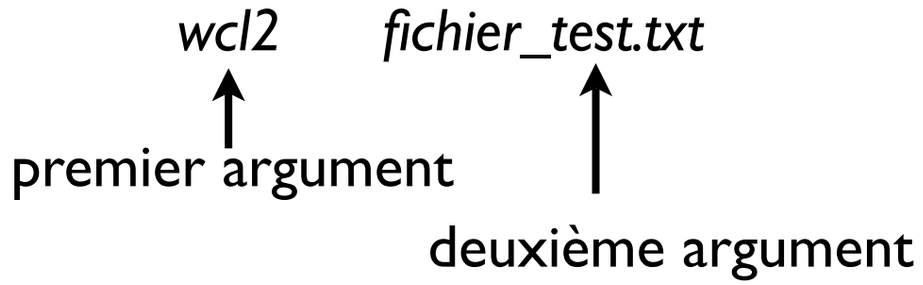
avec *nomdemonfichier* le nom du fichier texte que l'on souhaite ouvrir pour obtenir une statistique sur le nombre de mots.

pour cela, nous allons utiliser le prototype usuel de main :

```
void main(int argc, char**argv)
{


}
```

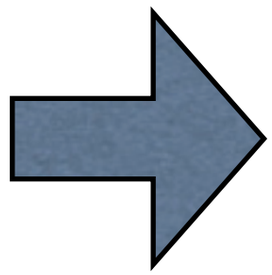
argc désigne le nombre d'arguments de la commande
argv est un tableau de chaînes de caractères , chaque chaîne contenant un des arguments

Récupérer les arguments :



•argc recense le nombre d'arguments (y compris le nom de la commande elle-même)
•argv contient les arguments : on peut donc y acceder à l'intérieur du programme

```
void main(int argc, char**argv)
{

   int i;

   printf("il y a %i arguments\n",argc);
   printf("les voici : \n");
   for (i=0;i<argc;i++)
           printf("%s \n",argv[i]);

}
```



```
./wcl2 fichier.txt param1  param2 -o -h --verbose
il y a 7 arguments
les voici :
./wcl2
fichier.txt
param1
param2
-o
-h
--verbose
```

Un seul argument : gestion du flux d'erreur standard

- La commande doit comporter un paramètre: le nom du fichier a traiter.
- sinon, elle affiche un message d'erreur :

usage : wcl2 nomdefichier

```
void main(int argc, char**argv)
{

   int i;
   if ((argc == 1)||(argc >2)) printf ("usage : wcl2 nomdefichier\n");
   else {

            /* on place l'algorithme de comptage des mots ici*/
    }
}
```

➡ Amélioration: écrire les messages d'erreurs sur le flux d'erreur standard

Pour les flux d'entrée / sortie standards, la bibliothèque C définit les pseudo-fichiers suivants:

- stdin flux d'entrée
- stdout flux de sortie stadard
- stderr flux d'erreur standard

Un seul argument : gestion du flux d'erreur standard

•la librairie C (stdio.h) définit les fonction d'ecriture et de lecture sur le flux :

•int fprintf(FILE * restrict stream, const char * restrict format, ...);
•int fscanf(FILE * restrict stream, const char * restrict format, ...);

•écriture et lecture formatée sur un descripteur associé aux flux ou aux fichiers.

•similaire à printf et scanf, mais le premier paramètre FILE* renseigne le flux ou le fichier dans lequel on souhaite faire une lecture/ecriture formatée.

```
void main(int argc, char**argv)
{

   int i;
   if ((argc == 1)||(argc >2)) fprintf (stderr,"usage : wcl2 nomdefichier\n");
   else {

          /* on place l'algorithme de comptage des mots ici*/
   }
}
```

• le message d'erreur est ainsi redirigé sur le flux stderr :

```
./wcl2 ddddd fffff ggggggg 2> testout.txt
cat testout.txt
usage : wcl2 nomdefichier
```

Lecture/ecriture d'un fichier

Au niveau de l'interface des appels système, un fichier est représenté par un **descripteur**.

#include <fcntl.h>

Les descripteurs sont numérotés par des (petits) entiers.
Avant d'utiliser un fichier, il faut l'ouvrir a l'aide de la commande `open()`.

open est une primitive qui ouvre un fichier et renvoie son descripteur

int open(const char *path, int oflag, mode_t mode);

path : chaîne de caractères désignant le chemin et le nom du fichier a ouvrir
mode : spécifie les protections du fichier (si le fichier doit être créé)
oflag : état d'ouverture du fichier :

```
O_RDONLY        open for reading only
O_WRONLY        open for writing only
O_RDWR          open for reading and writing
O_APPEND        append on each write
O_CREAT         create file if it does not exist
...
```

Lecture/ecriture d'un fichier

open() retourne un descripteur :

```
int fd ;
fd = open ("/home/machine/toto/fichier.txt", O_RDONLY, 0) ;
```

Ici, le fichier est ouvert en lecture seule (on ne peut pas y écrire)
le numéro de descripteur alloué par le système est `fd` .
fd vaut -1 s'il y a eu une erreur à l'ouverture

Quand on a fini d'utiliser un fichier, il faut le fermer . la primitive close() ferme le fichier associé au descripteur retourné par open() :

```
#include <unistd.h>
int close (int descripteur)
```

Le descripteur `fd` n'est plus utilisable, et pourra être réalloué par le système.

ex :

```
    close (fd);
```

close() retourne -1 en cas d'echec (erreur) ou 0 en cas de modification réussie.

Lecture/ecriture d'un fichier

Ce qui nous donne :

```
#include <stdlib.h>
#include <stdio.h>
#include <fcntl.h>

/* Pierre Andry */
/* Ecriture d'une commande C emulant la commande wc (word count)*/

void main(int argc, char**argv)
{
   int fd;  /* descripteur de fichier */


   if ((argc == 1)||(argc >2)) fprintf (stderr,"usage : wcl2 nomdefichier\n");
   else {
            fd = open (argv[1],O_RDONLY,0);
            if (fd == -1) fprintf (stderr,"wcl2 : %s n'existe pas \n",argv[1]);
            else{
            /* on place l'algorithme de comptage des mots ici*/

            }
     close(fd);
   }
}
```

Lecture/ecriture d'un fichier

L'ouverture crée un **pointeur courant** (position dans le fichier), initialisé à 0.

Ce pointeur (invisible directement) peut être déplacé
• indirectement, par une opération de lecture (`read()`)
d'écriture (`write()`).

• directement, par l'opération `lseek` ()

read() permet de lire les octets d'un fichier.

```
ssize_t read (int desc, void *buf,size_t nombre);
```

La primitive lit `nombre` octets dans le fichier associé à `desc` et les place à partir de l'adresse `buf`

La primitive retourne le nombre d'octets lus et stockés dans buf
-1 en cas d'erreur

Ainsi on effectue une lecture par "paquets" de nombre caractères.

Cette lecture va être effectuée en boucle jusqu'a ce que l'on atteigne la fin du fichier.

Lecture/ecriture d'un fichier

Exemple de lecture caractère par caractère (autrement dit, la primitive système read() va être appelée pour chaque caractère du fichier : coûteux en temps, peu coûteux en mémoire):

```
int nblu;
char lecture;
...
/*ouverture dans le descritpeur fd*/
...
     nblu = 1;
     while (nblu > 0) {
           nblu = read (fd,&lecture,1);
     }
/* fermeture de fd*/
```

Exemple de lecture 10 caractères par 10 caractères (autrement dit, la primitive système read() va être appelée tous les 10 caractère du fichier : moins coûteux en temps, plus coûteux en mémoire):

```
int nblu;
char lecture[10];
...
/*ouverture dans le descritpeur fd*/
...
     nblu = 1;
     while (nblu > 0) {
           nblu = read (fd,lecture,10);
     }
/* fermeture de fd*/
```

Lecture/ecriture d'un fichier

Attention !

Les primitives fournies par le noyau (`open`, `close`, `lseek`, `read`, `write`) sont celles de plus bas niveau.

On parle de primitives système

Leur utilisation est souvent délicate (gestion des erreurs, lectures tronquées, etc.)

Algorithme...

Notre problème se résume à compter le nombre de mots présent dans un fichier (qui peut être assimilé à une chaîne de caractères).

Nous travaillons donc avec des symboles (les caractères).
Nous distinguerons :

Les symboles constituants les mots :
S = {a-z,0-9,&,é,”,’,(,§,è,ç,à,...}

Les symboles constituant des séparations entre les mots :
D = { ,.,,,;,!,?,:}

On peut aussi considérer deux états :

état 1 : je parcours un mot
état 2: je parcours un “espace” entre deux mots

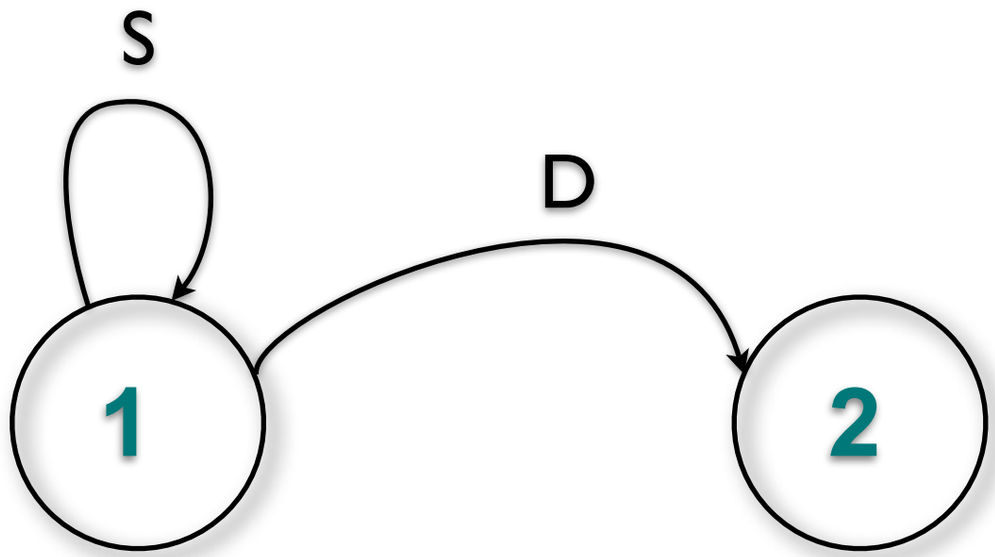
Ainsi notre problème peut être modélisé par un automate :

Algorithme...

Ainsi notre problème peut être modélisé par un automate (cf cours de J.F Rey):

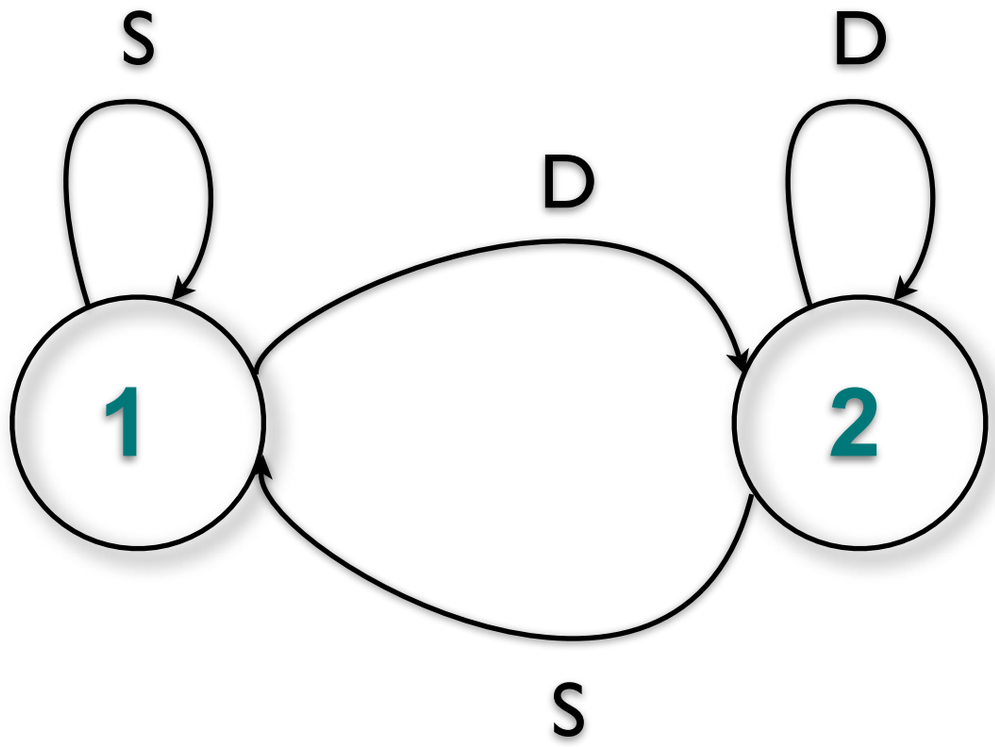
Algorithme...

Ainsi notre problème peut être modélisé par un automate :



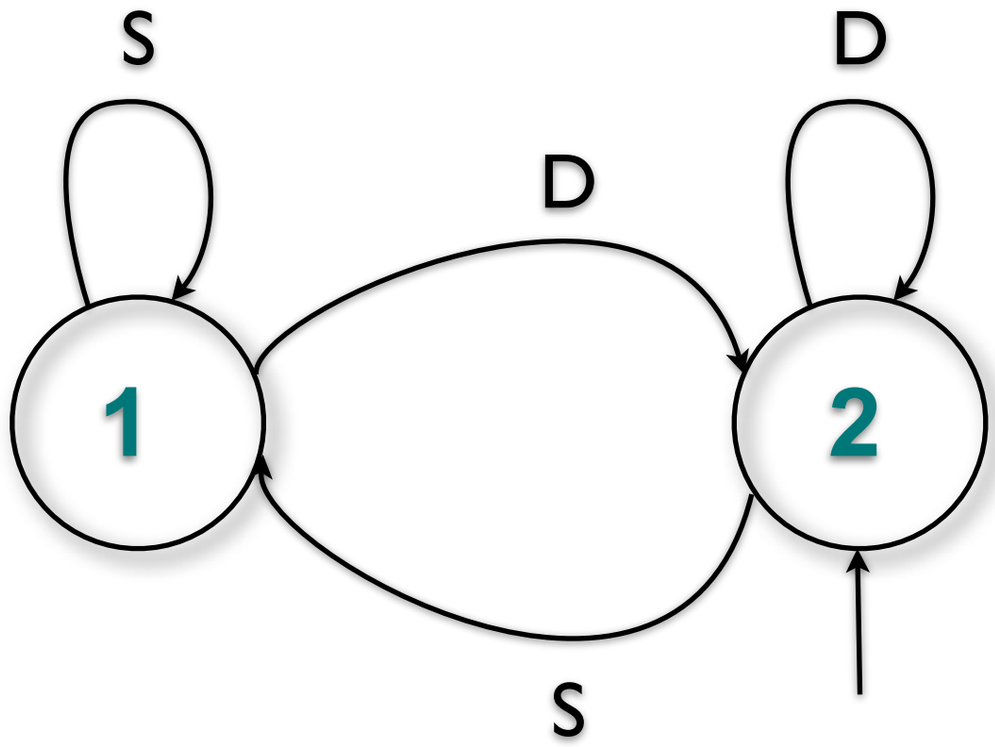
Algorithme...

Ainsi notre problème peut être modélisé par un automate :



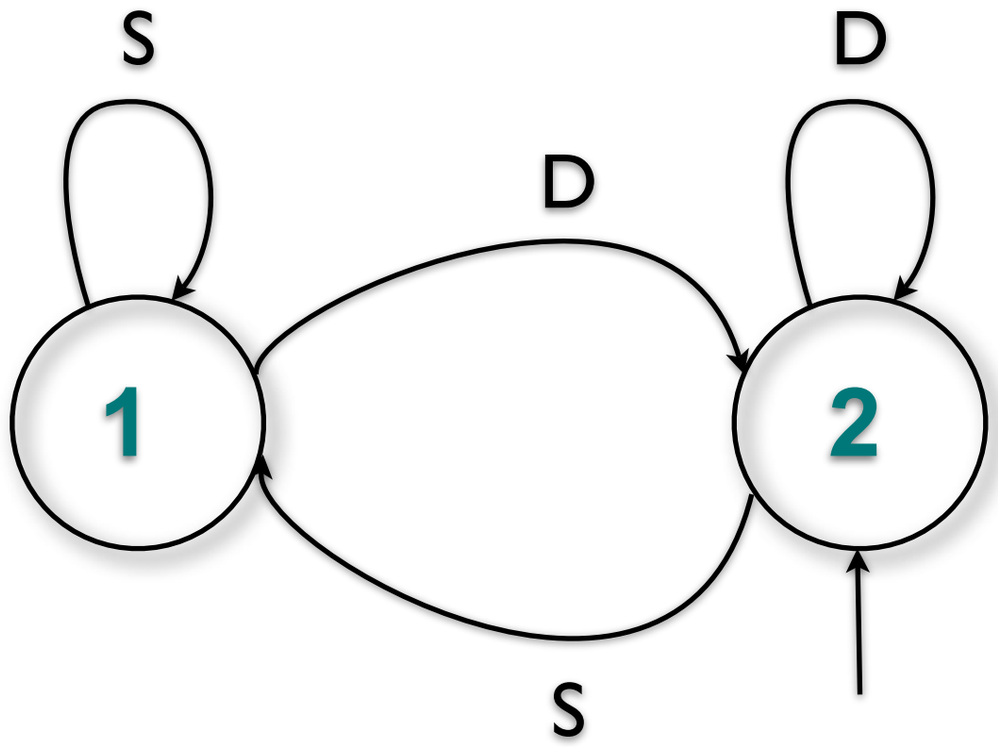
Algorithme...

Ainsi notre problème peut être modélisé par un automate :



Algorithme...

Ainsi notre problème peut être modélisé par un automate :



A chaque fois que l'on passe de 2 à 1 : on incrémente le compteur de mots.

Algorithme...

etat_courant = 2
nbmots = 0
nblu = 1

Algorithme...

etat_courant = 2
nbmots = 0
nblu = 1

tantque (nblu > 0)

> nblu = lire_fichier(lecture) /*lecture des caractère un par un*/

fin tantque

Algorithme...

```
etat_courant = 2
nbmots = 0
nblu = 1


tantque (nblu > 0)

    nblu = lire_fichier(lecture)  /*lecture des caractère un par un*/

     si (lecture == separateur )



    sinon



    fin si

fin tantque
```

Algorithme...

```
etat_courant = 2
nbmots = 0
nblu = 1


tantque (nblu > 0)

    nblu = lire_fichier(lecture)  /*lecture des caractère un par un*/

     si (lecture == separateur ) /* on passe ou on reste dans l'etat 2 */



    sinon                        /*passe ou on reste dans l'etat 1*/



    fin si

fin tantque
```

Algorithme...

```
etat_courant = 2
nbmots = 0
nblu = 1


tantque (nblu > 0)

    nblu = lire_fichier(lecture)  /*lecture des caractère un par un*/

     si (lecture == separateur ) /* on passe ou on reste dans l'etat 2 */

            si (etat_courant == 1)

            fsi
    sinon                          /*passe ou on reste dans l'etat 1*/
            si (etat_courant == 2)


            fsi
    fin si

fin tantque
```

Algorithme...

```
etat_courant = 2
nbmots = 0
nblu = 1


tantque (nblu > 0)

    nblu = lire_fichier(lecture)  /*lecture des caractère un par un*/

     si (lecture == separateur ) /* on passe ou on reste dans l'etat 2 */

            si (etat_courant == 1)
                etat_courant = 2
            fsi
    sinon                       /*passe ou on reste dans l'etat 1*/
            si (etat_courant == 2)
                etat_courant = 1

            fsi
    fin si

fin tantque
```

Algorithme...

```
etat_courant = 2
nbmots = 0
nblu = 1


tantque (nblu > 0)

    nblu = lire_fichier(lecture)  /*lecture des caractère un par un*/

     si (lecture == separateur ) /* on passe ou on reste dans l'etat 2 */

            si (etat_courant == 1)
                etat_courant = 2
            fsi
    sinon                       /*passe ou on reste dans l'etat 1*/
            si (etat_courant == 2)
                etat_courant = 1
                nbmot = nbmot +1
            fsi
    fin si

fin tantque
```

Algorithme...

```
void main(int argc, char**argv)
{
   int fd;
   int nblu;
   char lecture;
   int i;
   int etat_courant;
   int nbmot = 0;

   if ((argc == 1)||(argc >2)) fprintf (stderr,"usage : wcl2 nomdefichier\n");
   else {
            fd = open (argv[1],O_RDONLY,0);
            if (fd == -1) fprintf (stderr,"wcl2 : %s n'existe pas \n",argv[1]);
            else{
                  nblu = 1;
                  etat_courant=2;
                  while (nblu > 0) {

                        nblu = read (fd,&lecture,1);
                        if (nblu > 0) {
                              if (is_separator(lecture) == 1) {  /*  *lecture est un séparateur */
                                    if (etat_courant == 1 ) etat_courant = 2;
                                    }
                              else { /* *lecture est une lettre */
                                    if (etat_courant == 2) {etat_courant = 1; nbmot++;}
                                    }

                              }
                  }
            close(fd);
            }
   }

}
```

Algorithme...

```
etat_courant = 2
nbmots = 0
nblu = 1


tantque (nblu > 0)

    nblu = lire_fichier(lecture)  /*lecture des caractère un par un*/

    si (lecture == separateur ) /* on passe ou on reste dans l'etat 2 */

            si (etat_courant == 1)
                etat_courant = 2
            fsi
    sinon                       /*passe ou on reste dans l'etat 1*/
            si (etat_courant == 2)
                etat_courant = 1
                nbmot = nbmot +1
            fsi
    fin si

fin tantque
```

Algorithme...

```c
/*fonction qui teste la lettre c */
/* retourne 1 si la lettre est un separateur */
/* 0 sinon */

int is_separator (char c){
   int status = 0;

   switch (c) {
            case '\n':
            case ' ' :
            case ',' :
            case ';' :
            case '.' :
            case '!' :
            case '?' :
            case ':' :  status =1 ; break;
            default  :  status = 0; break;
   }

   return status;


}
```

Valeur de retour

Rappel : les commandes systèmes envoient à l'OS une valeur de retour à l'issue de leur execution.
- 0 (pas d'erreur) si tout s'est bien passé
- 1 ou une valeur positive (code d'erreur) si un erreur s'est produite.

La variable $? enregistre la valeur de retour de la dernière commande executée.
Cela permet de tester (dans un script shell par exemple) si la commande s'est terminée avec succès.

exemple :

```
Macintosh-2:commande_c pierre$ ls
makefile testout.txt wcl2     wcl2.h
makefile~   testout.txt~   wcl2.c      wcl2.o
Macintosh-2:commande_c pierre$ echo $?
0
Macintosh-2:commande_c pierre$ ls chabada
ls: chabada: No such file or directory
Macintosh-2:commande_c pierre$ echo $?
1
```

Nous devons donc adapter notre programme pour qu'il retourne ces valeurs au système d'exploitation

Valeur de retour

```
int main(int argc, char**argv)
{
    int fd;
    int nblu;
    char lecture;
    int i;
    int etat_courant;
    int nbmot = 0;

    if ((argc == 1)||(argc >2)) {fprintf (stderr,"usage : wcl2 nomdefichier\n"); return 1;}
    else {
              fd = open (argv[1],O_RDONLY,0);
              if (fd == -1) {fprintf (stderr,"wcl2 : %s n'existe pas \n",argv[1]); return 1;}
              else{
                    nblu = 1;
                    etat_courant=2;
                    while (nblu > 0) {

                          nblu = read (fd,&lecture,1);
                          if (nblu == -1) {fprintf (stderr,"erreur de lecture du fichier \n"); return 1;}
                          if (nblu > 0) {
                                if (is_separator(lecture) == 1) {  /*  *lecture est un séparateur */
                                      if (etat_courant == 1 ) etat_courant = 2;
                                      }
                                else { /* *lecture est une lettre */
                                      if (etat_courant == 2) {etat_courant = 1; nbmot++;}
                                      }

                                }
                    }

              /* on place l'algorithme de comptage des mots ici*/
              close(fd);
              }
    }
printf("%i",nbmot);
return 0;
}
```

Projet :

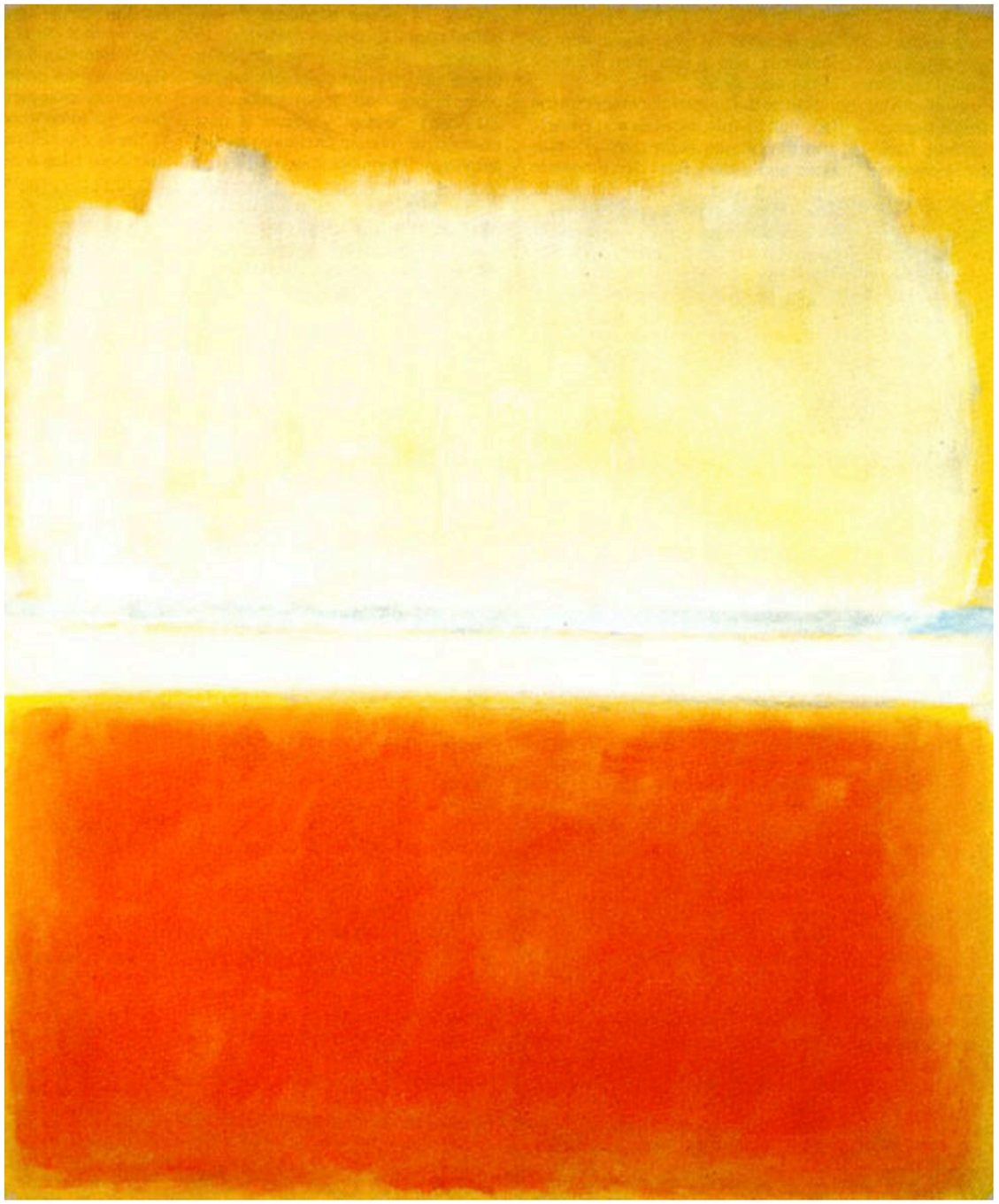
```c
#include <stdlib.h>
#include <stdio.h>
#include <fcntl.h>
#include "wcl2.h"
/*fonction qui teste la lettre pointée par c */
/* retourne 1 si la lettre est un separateur */
/* 0 sinon */
int is_separator (char c){
     int status = 0;
     switch (c) {
                  case '\n':
                  case ' ' :
                  case ',' :
                  case ';' :
                  case '.' :
                  case '!' :
                  case '?' :
                  case ':' :  status =1 ; break;
                  default  :  status = 0; break;
     }
     return status;
}

/* Pierre Andry */
/* Ecriture d'une commande C emulant la commande wc (word count)*/

int main(int argc, char**argv)
{
     int fd,nblu, etat_courant;
     char lecture;
     int nbmot = 0;
     if ((argc == 1)||(argc >2)) {fprintf (stderr,"usage : wcl2 nomdefichier\n"); return 1;}
     else {
                  fd = open (argv[1],O_RDONLY,0);
                  if (fd == -1) {fprintf (stderr,"wcl2 : %s n'existe pas \n",argv[1]); return 1;}
                  else{
                           nblu = 1;
                           etat_courant=2;
                           while (nblu > 0) {

                                    nblu = read (fd,&lecture,1);
                                    if (nblu == -1) {fprintf (stderr,"erreur de lecture du fichier \n"); return 1;}
                                    if (nblu > 0) {
                                             if (is_separator(lecture) == 1) {  /*  *lecture est un séparateur */
                                                      if (etat_courant == 1 ) etat_courant = 2;
                                                      }
                                             else { /* *lecture est une lettre */
                                                      if (etat_courant == 2) {etat_courant = 1; nbmot++;}
                                                      }
                                             }
                           }
                  close(fd);
                  }
     }
printf("%i",nbmot);
return 0;
}
```

```c
#ifndef WCL2__H
#define WCL2__H

int is_separator(char );



#endif /*WCL2__H */
```

Projet :

```
CC=gcc
FLAGS= -g -Wall


all : wcl2

wcl2.o : wcl2.c  wcl2.h
	$(CC) $(FLAGS) -c wcl2.c


wcl2 : wcl2.o
	$(CC) $(FLAGS) -o wcl2 wcl2.o

clean :
	rm -rf *.o wcl2
```

Optimisons...



Ecriture avec la primitive write

De la meme manière que nous avons utilisé la primitive read()
nous pouvons utiliser la primitive write() pour écrire dans un fichier ou un flux

```
ssize_t write(int fildes, const void *buf, size_t nbyte);
```

filedes : entier du descripteur de fichier associé au fichier (ouvert avec open) ou au flux désiré
buf : adresse du message a écrire
nbyte : nombre d'octets (de bytes) que l'on souhaite écrire

Nous savons que les flux 0,1 et 2 sont ouvert pour tout processus UNIX, nous pouvons donc directement écrire dessus

exemple :

```
fprintf (stderr,"usage : wcl2 nomdefichier\n");
```

devient :

```
write (2,"usage : wcl2 nomdefichier\n",26);
```

Optimisons read

La solution proposée utilise un appel a read() pour chaque caractère.
Dans le cas d'un grand fichier, ce peut être coûteux et sous-optimal.
Nous allons donc introduire un appel à read() par paquets de caractères :

```
int main(int argc, char**argv)
{
   ...
   char lecture;
   ...

                        while (nblu > 0) {

                        nblu = read (fd,&lecture,1);
                        ...
                        if (nblu > 0) {

                              if (is_separator(lecture) == 1) {  /*  *lecture est un séparateur */
                                    if (etat_courant == 1 ) etat_courant = 2;
                                    }
                              else { /* *lecture est une lettre */
                                    if (etat_courant == 2) {etat_courant = 1; nbmot++;}
                                    }

                              }
                  }
            close(fd);
...
```

Optimisons read

La solution proposée utilise un appel a read() pour chaque caractère.
Dans le cas d'un grand fichier, ce peut être coûteux et sous-optimal.
Nous allons donc introduire un appel à read() par paquets de caractères :

```
int main(int argc, char**argv)
{
   ...
   char lecture[1];
   ...
                        while (nblu > 0) {

                        nblu = read (fd,lecture,1);
                        ...
                        if (nblu > 0) {
                              for (i=0;i<nblu;i++)
                              if (is_separator(lecture[i]) == 1) {  /*  *lecture est un séparateur */
                                    if (etat_courant == 1 ) etat_courant = 2;
                                    }
                              else { /* *lecture est une lettre */
                                    if (etat_courant == 2) {etat_courant = 1; nbmot++;}
                                    }

                              }
                  }
            close(fd);
...
```

Optimisons read

La solution proposée utilise un appel a read() pour chaque caractère.
Dans le cas d'un grand fichier, ce peut être coûteux et sous-optimal.
Nous allons donc introduire un appel à read() par paquets de caractères :

```
int main(int argc, char**argv)
{
   ...
   char lecture[1];
   ...

                        while (nblu > 0) {

                        nblu = read (fd,lecture,1);
                        ...
                        if (nblu > 0) {
                              for (i=0;i<nblu;i++)
                              if (is_separator(lecture[i]) == 1) {  /*  *lecture est un séparateur */
                                    if (etat_courant == 1 ) etat_courant = 2;
                                    }
                              else { /* *lecture est une lettre */
                                    if (etat_courant == 2) {etat_courant = 1; nbmot++;}
                                    }

                              }
                  }
            close(fd);
...
```

Optimisons read

La solution proposée utilise un appel a read() pour chaque caractère.
Dans le cas d'un grand fichier, ce peut être coûteux et sous-optimal.
Nous allons donc introduire un appel à read() par paquets de caractères :

```
int main(int argc, char**argv)
{
   ...
   char lecture[NB_OCTETS];
   ...

                        while (nblu > 0) {

                        nblu = read (fd,lecture,NB_OCTETS);
                        ...
                        if (nblu > 0) {
                              for (i=0;i<nblu;i++)
                              if (is_separator(lecture[i]) == 1) {  /*  *lecture est un séparateur */
                                    if (etat_courant == 1 ) etat_courant = 2;
                                    }
                              else { /* *lecture est une lettre */
                                    if (etat_courant == 2) {etat_courant = 1; nbmot++;}
                                    }

                              }
                  }
            close(fd);
...
```

Optimisons read

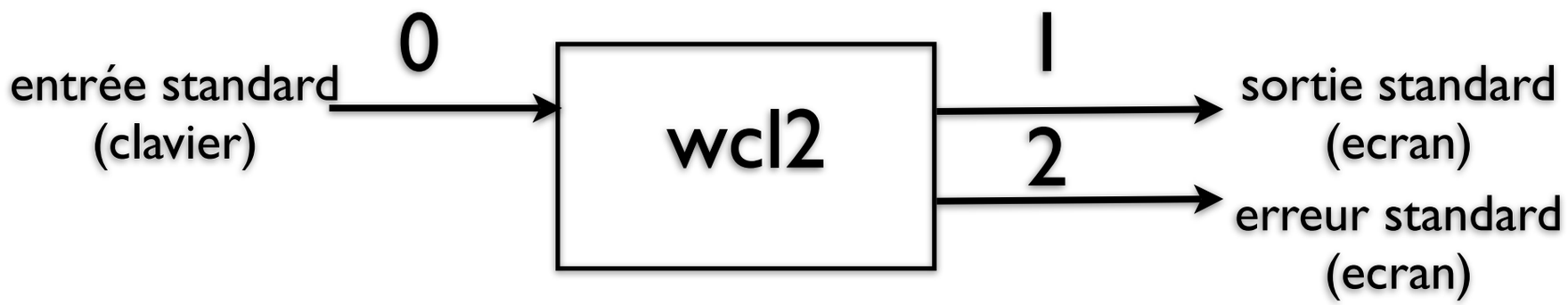
La solution proposée utilise un appel a read() pour chaque caractère.
Dans le cas d'un grand fichier, ce peut être coûteux et sous-optimal.
Nous allons donc introduire un appel à read() par paquets de caractères :

```
#ifndef WCL2__H
#define WCL2__H


#define NB_OCTETS 10
int is_separator(char );




#endif /*WCL2__H */
```

Nous pouvons ainsi moduler, à la compilation la taille du tampon de lecture

Gestion de l’entrée standard

Les process UNIX accèdent aux flux 0,1 et 2

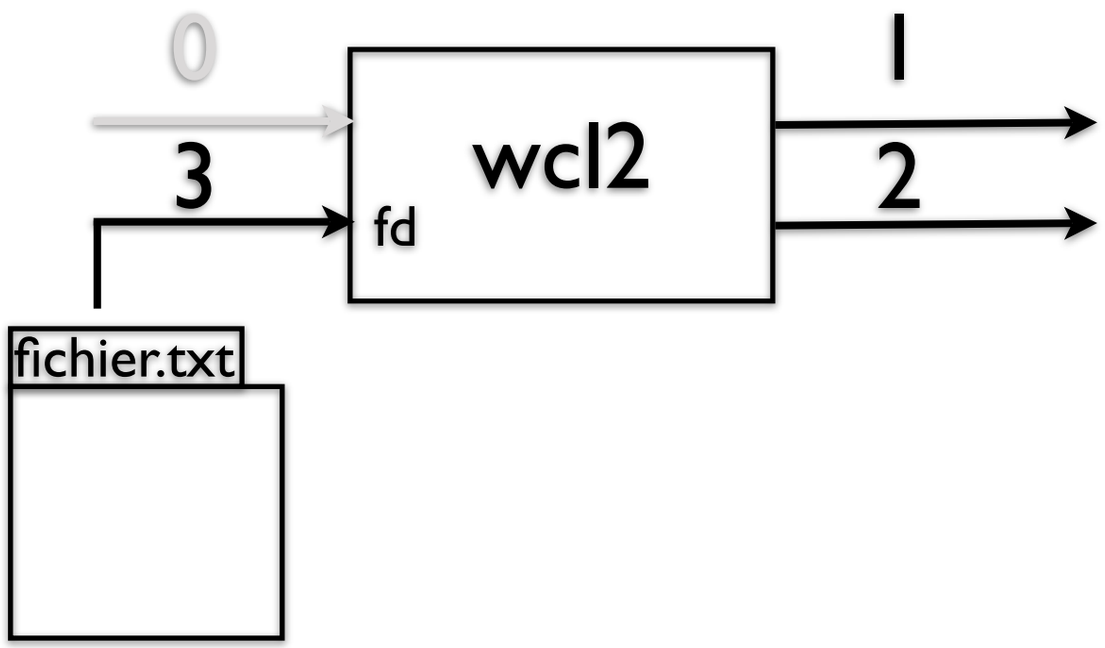


Gestion de l’entrée standard

Nous avons prévu son fonctionnement pour que wcl2 s’applique au fichier en argument

```
Macintosh-2:v1 pierre$ ./wcl2 fichier.txt
12
Macintosh-2:v1 pierre$
```

fichier.txt est ouvert via open() et le descripteur fd (qui a en fait la valeur 3)



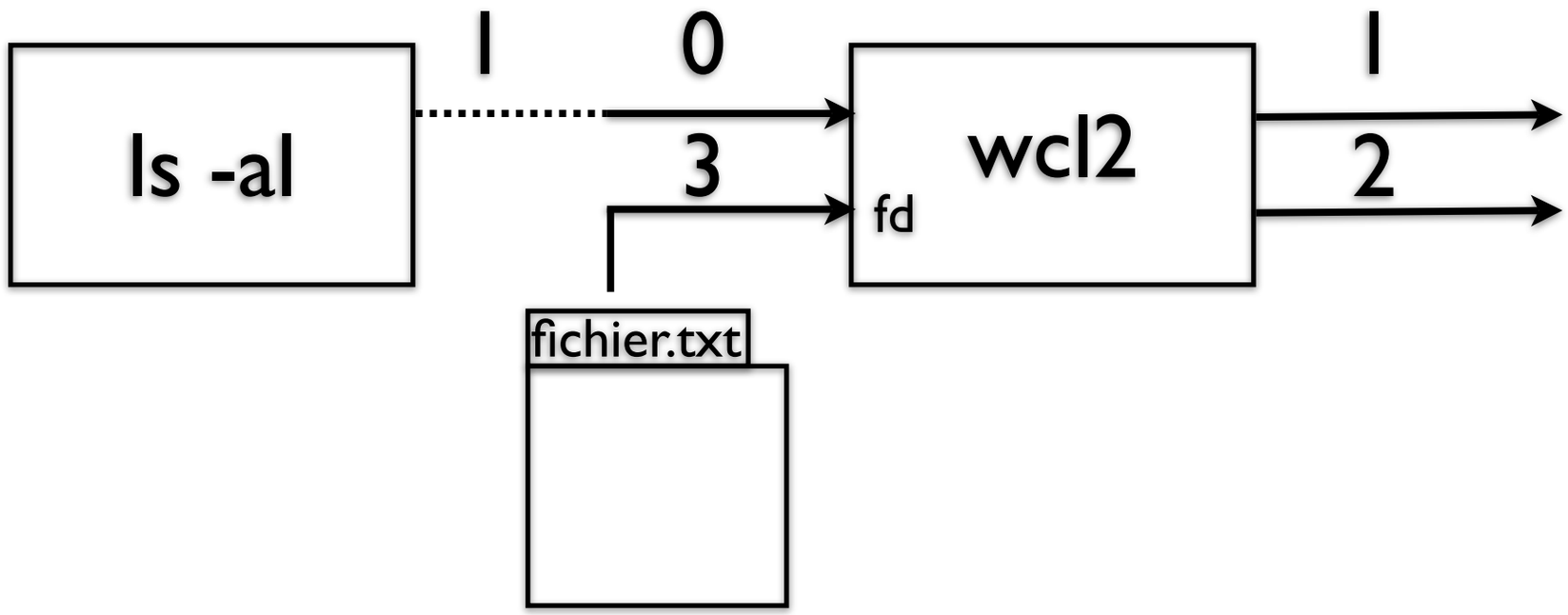
Gestion de l'entrée standard

En l'état, notre commande ne permet pas de prendre en compte l'entrée standard. Ainsi :

```
Macintosh-2:v1 pierre$ ls -al | ./wcl2
usage : wcl2 nomdefichier
Macintosh-2:v1 pierre$
```

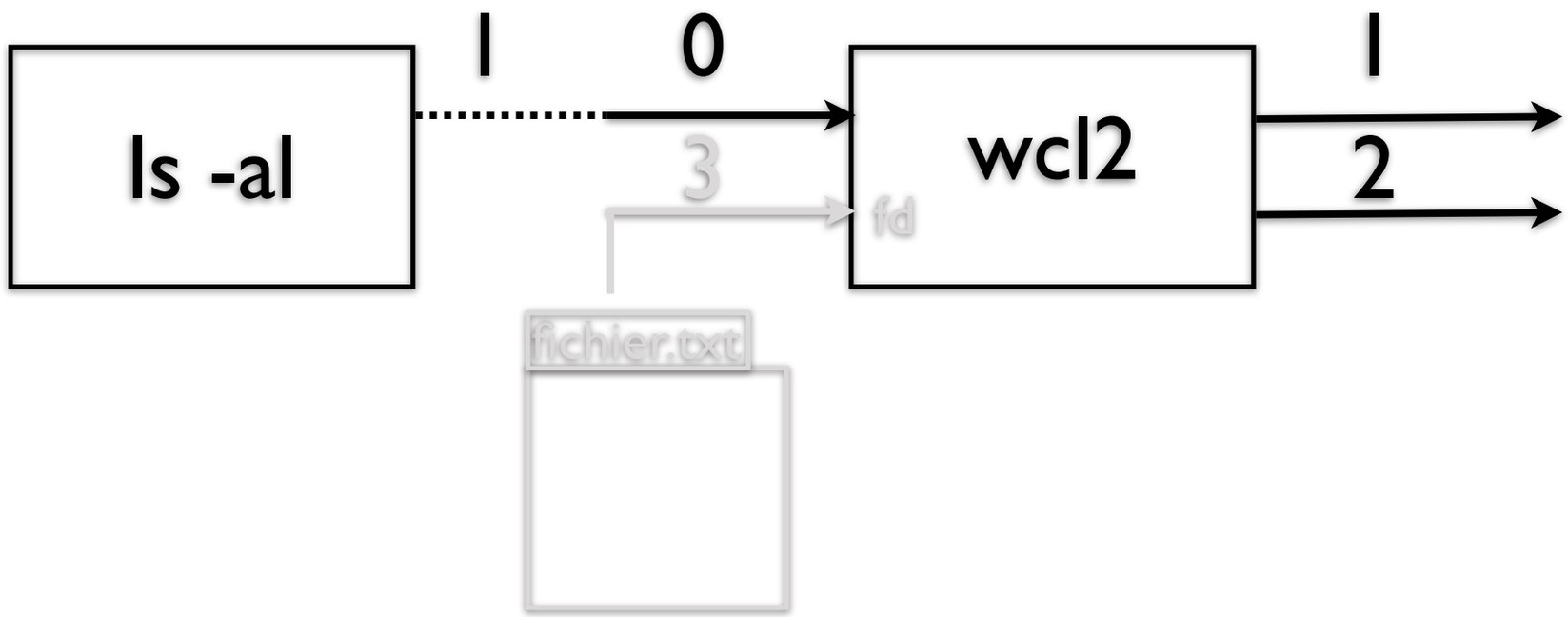
Nous retourne un message d'erreur

Notre commande ne prend pas en compte l'entrée standard.
Elle est inefficace dans le cadre d'un tube, par exemple.



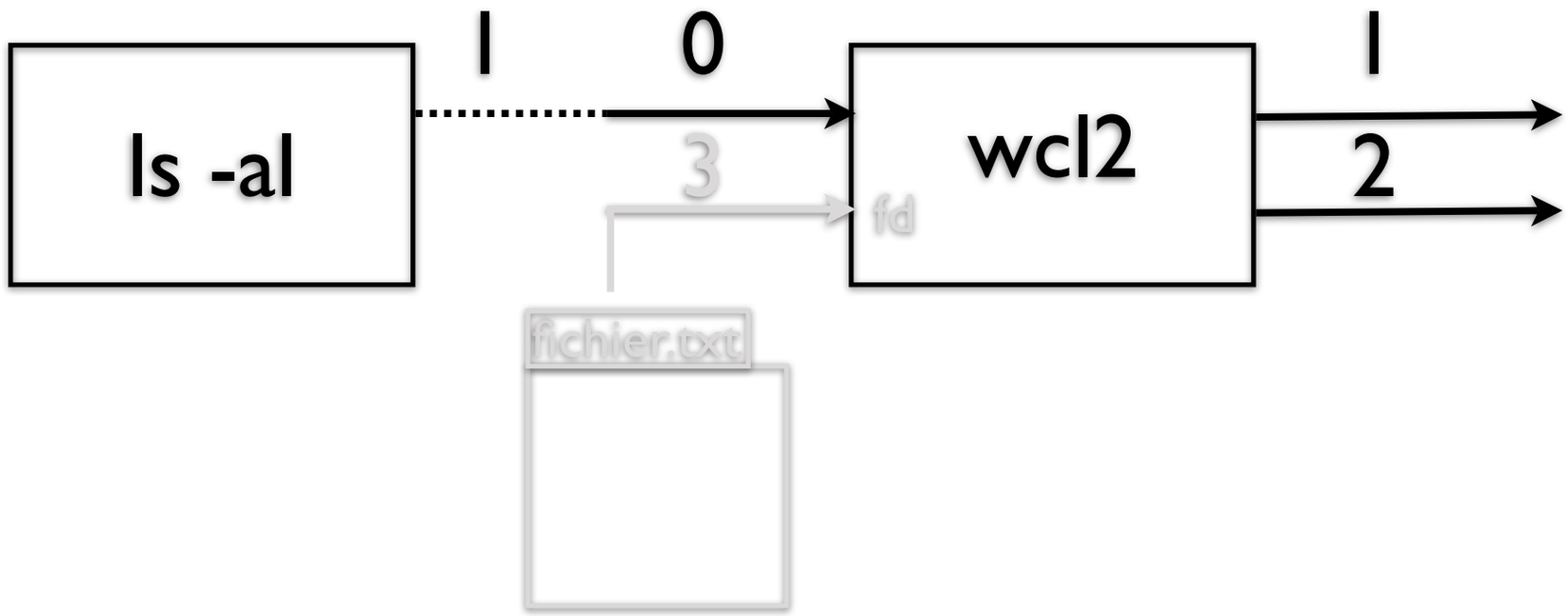
Gestion de l’entrée standard

Si aucun parmaètre n’est présent en entrée, alors il faut lire sur le flux 0, c’est à dire le descripteur 0



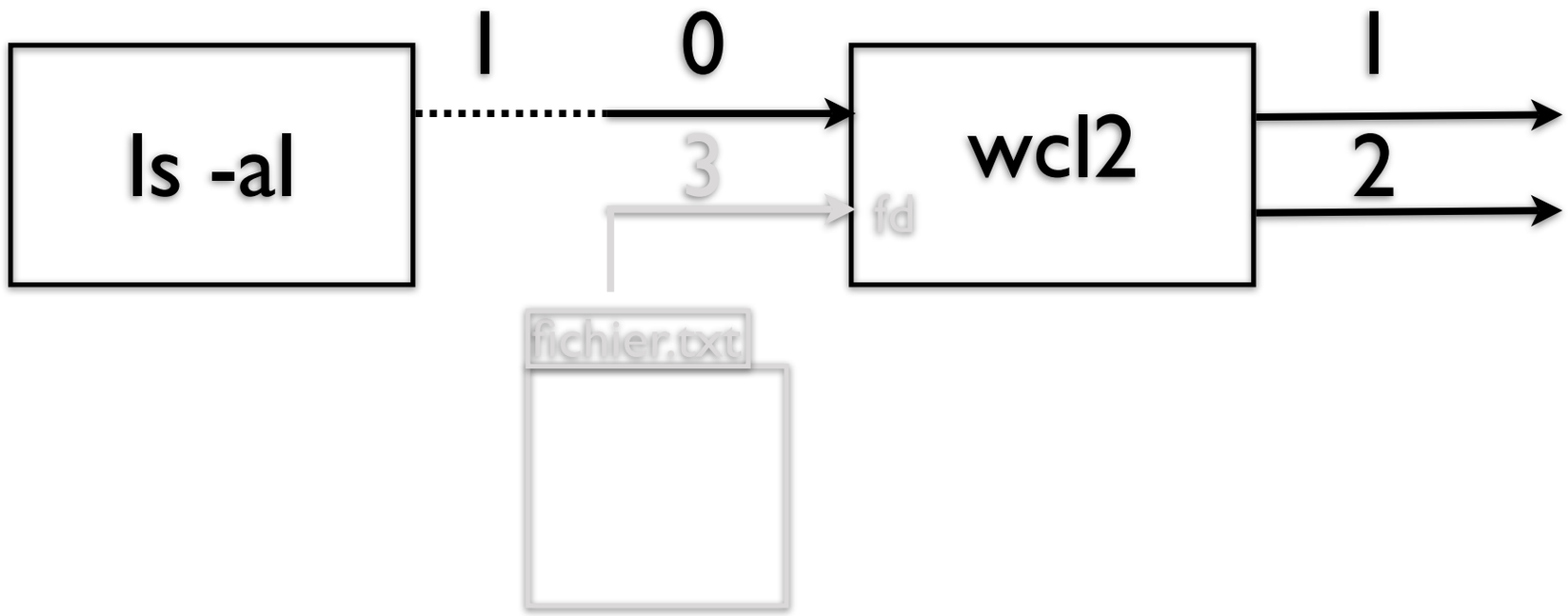
Gestion de l’entrée standard

```c
int main(int argc, char**argv)
{
    int fd = 0; /* par defaut sur l'entrée standard*/
    int nblu;
    char lecture[NB_OCTETS];
    int i;
    int etat_courant;
    int nbmot = 0;

    if (argc >2) {write (2,"usage : wcl2 nomdefichier\n",26); return 1;} /* trop de paramètres*/
    if (argc == 2) {  /* un paramètre , le nom du fichier est passé en argument*/
              fd = open (argv[1],O_RDONLY,0);
              if (fd == -1) {write (2,"wcl2 : le fichier n'existe pas \n",32); return 1;}
    }
    nblu = 1;
    etat_courant=2;
    while (nblu > 0) {
              nblu = read (fd,lecture,NB_OCTETS);
```



Gestion de l’entrée standard

```c
int main(int argc, char**argv)
{
    int fd = 0; /* par defaut sur l'entrée standard*/
    int nblu;
    char lecture[NB_OCTETS];
    int i;
    int etat_courant;
    int nbmot = 0;

    if (argc >2) {write (2,"usage : wcl2 nomdefichier\n",26); return 1;} /* trop de paramètres*/
    if (argc == 2) {  /* un paramètre , le nom du fichier est passé en argument*/
              fd = open (argv[1],O_RDONLY,0);
              if (fd == -1) {write (2,"wcl2 : le fichier n'existe pas \n",32); return 1;}
    }
    nblu = 1;
    etat_courant=2;
    while (nblu > 0) {
              nblu = read (fd,lecture,NB_OCTETS);
```



Test...

A retenir

Distinguez :

les primitives système open() read() write() close() qui permettent d'effectuer des appels système et qui font partie du langage...

...des fonctions de la librairie standard, par exemple fopen(), fscanf(), fprintf(), fclose() qui sont implémentées dans un librairie C standard, ici stdio.h