

Les entrées-sorties

Système

Gilles Roussel

Les entrées-sorties

Entrées-sorties avec les fichiers

La buffering

Les répertoires

Les tubes

Entrées-sorties bas niveau

scanf()

Liens des pages

- Bibliothèque standard (ANSI) de manipulation des entrées-sorties
- Type opaque de manipulation de descripteur **FILE *** défini dans **stdio.h**
- Descripteurs de base **stdin stdout** et **stderr**
- Descripteur associé (entre autre) à une position courante et à un tampon mémoire (*buffer*)
- Fonctions de manipulation *bufferisée* des entrées-sorties
 - **fgetc()** (**getchar()**) lecture d'un caractère
 - **fgets()** (**gets()**) lecture d'une chaîne de caractères
 - **fscanf()** (**scanf()**) lecture d'une chaîne de caractères formatée
 - **fputc()** (**putchar()**) écriture d'un caractère
 - **fputs()** (**puts()**) écriture d'une chaîne de caractères
 - **fprintf()** (**printf()**) écriture d'une chaîne de caractères formatée

Gilles Roussel

Système

Système Les entrées-sorties

Gilles Roussel

Gilles.Roussel@univ-mlv.fr

<http://igm.univ-mlv.fr/ens/Licence/L2/2012-2013/System/>

Licence 2

10 mars 2015

Gilles Roussel

Système

Parenthèse : « Quel fichier d'en-tête ajouter ? »

Système

Gilles Roussel

Les entrées-sorties

Entrées-sorties avec les fichiers

La buffering

Les répertoires

Les tubes

Entrées-sorties bas niveau

scanf()

Liens des pages

- Trouver le fichier `.h` grâce au manuel

```
# man printf
PRINTF(1)  User Commands  PRINTF(1)
NAME
    printf - format and print data
...
■ Le bon !
# man -k printf
...
printf      (1) - format and print data
printf      (3) - formatted output conversion
...
# man 3 printf
PRINTF(3)  Linux Programmer's Manual  PRINTF(3)
NAME
    printf, ...
SYNOPSIS
    #include <stdio.h>
...
```

Gilles Roussel

Système

Parenthèse : « Et les tests d'erreur ? »

Système

Gilles Roussel

Les entrées-sorties

Entrées-sorties avec les fichiers

La buffering

Les répertoires

Les tubes

Entrées-sorties bas niveau

scanf()

Liens des pages

- Il faut tester les valeurs de retour
- **fprintf()** retourne **le nombre de caractères écrits** et **une valeur négative** en cas d'erreur
- **feof()** retourne normalement **0** et **EOF** en cas d'erreur
- **ferror()** retourne **0** s'il n'y a pas eu d'erreur
- **clearerr()** permet de supprimer l'indicateur d'erreur
- Numéro d'erreur placé dans variable globale **errno**
- **strerror()** et **perror()** permettent respectivement d'obtenir une chaîne de caractères associée à l'erreur et l'afficher sur la sortie d'erreur

Gilles Roussel

Système

Parenthèse :

< (f)printf est une source de bugs de sécurité >

Système

Gilles Roussel

Les entrées-sorties

Entrées-sorties vers les fichiers

La buffering

Les répertoires

Les tubes

Entrées-sorties bas niveau

sigint

Les des lignes

read_mem.c

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

int main(int argc, char *argv[]) {
    printf(argv[1]);
    exit(EXIT_SUCCESS);
}

# ./read_mem test
test
# ./read_mem '%s%s%s%s'
(null)ht^bf^c2^bf^c3^af^c2^bf^c2^bf^[%@
```

Gilles Roussel

Système

Entrées-sorties vers les fichiers

Système

Gilles Roussel

Les entrées-sorties

Entrées-sorties vers les fichiers

La buffering

Les répertoires

Les tubes

Entrées-sorties bas niveau

sigint

Les des lignes

- **fopen()** permet d'ouvrir un fichier
FILE *fopen(const char *path, const char *mode);
 - "r" lecture, "r+" lecture/écriture à partir du début du fichier
 - "w" écriture, "w+" lecture/écriture avec tronquage ou création du fichier
 - "a" écriture, "a+" lecture/écriture toujours à partir de la fin du fichier
- Retourne un descripteur ouvert sur le fichier ou **NULL** en cas d'erreur
- Descripteur sur fichier bufferisé par défaut : il faut faire **fflush()** pour assurer l'écriture sur disque
- **fclose()** permet de fermer un descripteur ouvert ; son tampon est vidé automatiquement (**fflush()**)
- Retourne 0 si pas de problème et **EOF** en cas d'erreur

Gilles Roussel

Système

Lecture/écriture binaire

Système

Gilles Roussel

Les entrées-sorties

Entrées-sorties vers les fichiers

La buffering

Les répertoires

Les tubes

Entrées-sorties bas niveau

sigint

Les des lignes

- Rarement utiles dans le cas de entrées-sorties standards
- Écriture d'un tableau contenant **nmemb** éléments de taille **size**
`size_t fwrite(const void *ptr, size_t size, size_t nmemb, FILE *stream);`
- Lecture bloquante d'un tableau de **nmemb** éléments de taille **size**
`size_t fread(void *ptr, size_t size, size_t nmemb, FILE *stream);`
- Retourne le nombre d'éléments lus ou écrits
- Si valeur de retour inférieure à nombre demandé, erreur ou fin de fichier atteinte (pour `fread()`)

Gilles Roussel

Système

Exemple : cat.c

Système

Gilles Roussel

Les entrées-sorties

Entrées-sorties vers les fichiers

La buffering

Les répertoires

Les tubes

Entrées-sorties bas niveau

sigint

Les des lignes

```
#include <stdio.h>
#include <stdlib.h>
#include <libgen.h>
#include <errno.h>
#define BUFFER_SIZE 1024

int main(int argc, char *argv[]) {
    FILE *from; char buffer[BUFFER_SIZE]; int i, n, exit_status = EXIT_SUCCESS;
    if (argc < 2) {
        fprintf(stderr, "usage: %s file1 [file2 ...]\n", basename(argv[0])); exit(EXIT_FAILURE);
    }
    for (i=1; i<argc;i++) {
        if (NULL == (from=fopen(argv[i], "r"))) {
            perror(argv[i]); exit_status = EXIT_FAILURE; continue;
        }
        while ((n=fread(buffer, sizeof(char), BUFFER_SIZE, from)) != 0) {
            if (fwrite(buffer, sizeof(char), n, stdout) != n) { break; }
        }
        if (ferror(from)) {
            perror("read"); clearerr(from); exit_status = EXIT_FAILURE;
        }
        if (ferror(stdout)) {
            perror("write"); clearerr(stdout); exit_status = EXIT_FAILURE;
        }
        fclose(from);
    }
    exit(exit_status);
}
```

Gilles Roussel

Système

Déplacement dans un fichier

Système

Gilles Roussel

Les entrées
sorties

Entrées-
sorties vers
les fichiers

La
bufferisation

Les
répertoires

Les tubes

Entrées-
sorties aux
niveau

sigint

Les des lignes

- **fseek()** permet de se placer à une position particulière dans un fichier

```
int fseek(FILE *stream, long offset, int whence);
```

 - **SEEK_SET** se déplace de offset par rapport au début du fichier
 - **SEEK_CUR** se déplace de offset par rapport à la position courante
 - **SEEK_END** se déplace de offset par rapport à la fin du fichier (valeur négative pour revenir en arrière)
- On peut aller au delà de la fin du fichier ; le fichier change de taille si écriture
- **fseek()** retourne 0 s'il n'y a pas eu d'erreur et -1 dans le cas contraire
- **ftell()** retourne la position courante ou -1 en cas d'erreur

Gilles Roussel

Système

Exemple : tail.c

Système

Gilles Roussel

Les entrées
sorties

Entrées-
sorties vers
les fichiers

La
bufferisation

Les
répertoires

Les tubes

Entrées-
sorties aux
niveau

sigint

Les des lignes

```
#include <stdio.h>
#include <stdlib.h>
#include <libgen.h>
#define BUFFER_SIZE 1024

int main(int argc, char *argv[]) {
    FILE *file; int size, nb; char *end; char buffer[BUFFER_SIZE];
    char usage[] = "usage: %s file nb_byte\n";
    if (argc != 3) {
        fprintf(stderr, usage, basename(argv[0])); exit(EXIT_FAILURE);
    }
    size = strtoul(argv[2], &end, 10);
    if ((*end != '\0') || (argv[2] == end)) {
        fprintf(stderr, usage, basename(argv[0])); exit(EXIT_FAILURE);
    }
    if (NULL == (file=fopen(argv[1], "r"))) {
        perror(argv[1]); exit(EXIT_FAILURE);
    }
    if (fseek(file, -size, SEEK_END) != 0) {
        perror(argv[1]); exit(EXIT_FAILURE);
    }
    while(0!=(nb=fread(buffer, sizeof(char), BUFFER_SIZE, file))) {
        if (fwrite(buffer, sizeof(char), nb, stdout) != nb) {
            perror("write"); exit(EXIT_FAILURE);
        }
    }
    if (ferror(file)) { perror("read"); exit(EXIT_FAILURE); }
    fclose(file);
    exit(EXIT_SUCCESS);
}
```

Gilles Roussel

Système

Bufferisation en entrée

Système

Gilles Roussel

Les entrées
sorties

Entrées-
sorties vers
les fichiers

La
bufferisation

Les
répertoires

Les tubes

Entrées-
sorties aux
niveau

sigint

Les des lignes

- Même quand un seul caractère est lu, plusieurs peuvent être lus et placés dans un tampon mémoire
- Entrée standard surtout bufferisée par le terminal de contrôle
 - Permet d'effacer les erreurs sans envoyer au programme le caractère erroné et le caractère d'effacement
 - Permet d'interpréter différemment certains caractères « spéciaux » (Ctrl-C, Ctrl-Z, etc.)
- **stty** permet de connaître et de changer le comportement du terminal de contrôle
 - **stty -a** affiche les caractéristiques du terminal de contrôle
 - **stty -icanon** permet de passer le terminal de contrôle en mode non-bufferisé
 - **stty icanon** permet de passer le terminal de contrôle en mode bufferisé

Gilles Roussel

Système

Bufferisation en sortie

Système

Gilles Roussel

Les entrées
sorties

Entrées-
sorties vers
les fichiers

La
bufferisation

Les
répertoires

Les tubes

Entrées-
sorties aux
niveau

sigint

Les des lignes

- Données d'abord écrites dans un tampon mémoire
- Tampon mémoire vidé quand un caractère fin de ligne (**\n**) est écrit ou quand le tampon mémoire est plein
- Par défaut :
 - sortie standard sur terminal bufferisée par ligne
 - sortie d'erreur sur terminal non-bufferisée
- **fflush()** permet de vider le tampon mémoire
- **setvbuf()** permet de changer le comportement par défaut (avant toute utilisation)
 - **_IONBF** non bufferisé
 - **_IOLBF** bufferisé par lignes
 - **_IOFBF** bufferisation complète
- **setvbuf()** permet de préciser le tampon mémoire à utiliser

Gilles Roussel

Système

Bufferisation en sortie

Système

Gilles Roussel

Les entrées
sorties

Entrées
sorties vers
les fichiers

La
bufferisation

Les
répertoires

Les tubes

Entrées
sorties bas
niveau

swap()

Liens des lignes

stdio_test.c

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

int main(int argc, char *argv[]) {
    fprintf(stdout, "stdout");
    fprintf(stderr, "stderr");
    fprintf(stdout, "\n");
    exit(EXIT_SUCCESS);
}
```

```
# ./stdio_test
stderrstdout
```

Gilles Roussel

Système

Bufferisation en sortie

Système

Gilles Roussel

Les entrées
sorties

Entrées
sorties vers
les fichiers

La
bufferisation

Les
répertoires

Les tubes

Entrées
sorties bas
niveau

swap()

Liens des lignes

stdio_test2.c

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

int main(int argc, char *argv[]) {
    fprintf(stdout, "stdout");
    fflush(stdout);
    fprintf(stderr, "stderr");
    fprintf(stdout, "\n");
    exit(EXIT_SUCCESS);
}
```

```
# ./stdio_test2
stdoutstderr
```

Gilles Roussel

Système

Bufferisation en sortie

Système

Gilles Roussel

Les entrées
sorties

Entrées
sorties vers
les fichiers

La
bufferisation

Les
répertoires

Les tubes

Entrées
sorties bas
niveau

swap()

Liens des lignes

stdio_test3.c

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

int main(int argc, char *argv[]) {
    setvbuf(stdout, (char *)NULL, _IONBF, 0);
    fprintf(stdout, "stdout");
    fprintf(stderr, "stderr");
    fprintf(stdout, "\n");
    exit(EXIT_SUCCESS);
}
```

```
# ./stdio_test3
stdoutstderr
```

Gilles Roussel

Système

Bufferisation en sortie

Système

Gilles Roussel

Les entrées
sorties

Entrées
sorties vers
les fichiers

La
bufferisation

Les
répertoires

Les tubes

Entrées
sorties bas
niveau

swap()

Liens des lignes

stdio_test4.c

```
#include <stdio.h>
#include <stdlib.h>
static char buffer[1024];

int main(int argc, char *argv[]) {
    setvbuf(stderr, buffer, _IOFBF, 1024);
    fprintf(stdout, "stdout");
    fprintf(stderr, "stderr");
    fprintf(stdout, "\n");
    exit(EXIT_SUCCESS);
}
```

```
# ./stdio_test4
stdout
stderr
```

Gilles Roussel

Système

Pourquoi bufferiser ?

Système

Gilles Roussel

Les entrées
sorties

Entrées
sorties avec
les fichiers

La
bufferisation

Les
répertoires

Les tubes

Entrées
sorties les
réseaux

sleep()

Liens des lignes

- Pas de bufferisation :
 - Affiche le plus vite possible les données soumises
- Par ligne :
 - Évite que les lignes affichées par plusieurs processus se mélangent
- D'une taille donnée
 - Améliore les performances

Gilles Roussel

Système

Améliorer les performances ?

Système

Gilles Roussel

Les entrées
sorties

Entrées
sorties avec
les fichiers

La
bufferisation

Les
répertoires

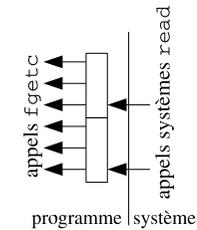
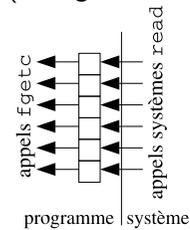
Les tubes

Entrées
sorties les
réseaux

sleep()

Liens des lignes

- Pour lire (ou écrire) besoin d'accéder au matériel
- Programme n'y accède pas directement
- Besoin de « passer la main » au système
 - Réalisé *via* un **appel système** (read() ou write())
 - Entraîne un **changement de contexte** qui est coûteux
- Bufferisation évite de multiples appels systèmes (changements de contexte)



Gilles Roussel

Système

Un benchmark

Système

Gilles Roussel

Les entrées
sorties

Entrées
sorties avec
les fichiers

La
bufferisation

Les
répertoires

Les tubes

Entrées
sorties les
réseaux

sleep()

Liens des lignes

```
#include <stdlib.h>
#include <stdio.h>
#include <sys/times.h>
#include <string.h>
#define MAX_BUFFER_SIZE 3000
int main(int argc, char * argv[]) {
    struct tms start, end; int i;
    char buffer[MAX_BUFFER_SIZE]; int size;
    FILE *file;
    FILE *from=fopen("/dev/urandom","r"), *to=fopen("/dev/null","w");
    if (!access("bench",F_OK)) {
        fprintf(stderr,"bench file already exists !\n"); exit(EXIT_FAILURE);
    }
    file = fopen("bench","w");
    for (size=2;size < MAX_BUFFER_SIZE; size*=2) {
        setvbuf(from,buffer,_IOFBF,size); setvbuf(to,buffer,_IOFBF,size);
        times(&start);
        for (i=0;i<100000000;i++) {
            fputc(fgetc(from),to);
        }
        times(&end);
        fprintf(file,"%d %ld %ld %ld\n",size,
            end.tms_utime-start.tms_utime,end.tms_stime-start.tms_stime,
            end.tms_utime+end.tms_stime-start.tms_utime-start.tms_stime);
        fflush(file);
    }
    fclose(file);fclose(from); fclose(to);
    exit(EXIT_SUCCESS);
}
```

Gilles Roussel

Système

Un benchmark

Système

Gilles Roussel

Les entrées
sorties

Entrées
sorties avec
les fichiers

La
bufferisation

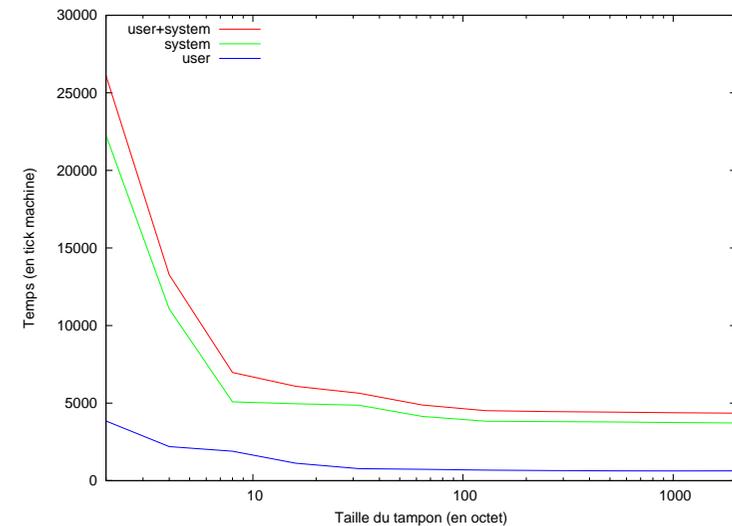
Les
répertoires

Les tubes

Entrées
sorties les
réseaux

sleep()

Liens des lignes



Gilles Roussel

Système

Parenthèse : « Le fichier résultat »

Système
Gilles Roussel
Les entrées sorties
Entrées sorties avec les fichiers
La bufferisation
Les répertoires
Les tubes
Entrées sorties les réseaux
Signal
Les des lignes

```
2 3845 22269 26114
4 2196 11074 13270
8 1899 5078 6977
16 1122 4960 6082
32 773 4868 5641
64 733 4146 4879
128 679 3833 4512
256 653 3804 4457
512 636 3786 4422
1024 629 3749 4378
2048 633 3717 4350
```

Parenthèse : « Le fichier gnuplot »

Système
Gilles Roussel
Les entrées sorties
Entrées sorties avec les fichiers
La bufferisation
Les répertoires
Les tubes
Entrées sorties les réseaux
Signal
Les des lignes

```
#!/usr/bin/gnuplot

set output "bench.eps"
set terminal postscript eps color enhanced solid
set key left top
set xlabel "Taille du tampon (en octet)"
set ylabel "Temps (en tick machine)"
set xr [2:2000]
set yr [0:30000]
set logscale x

set style line 1 linetype 1 linewidth 2
set style line 2 linetype 2 linewidth 2
set style line 3 linetype 3 linewidth 2
plot "bench" using 1:4 title "user+system" with lines ls 1, \
    "bench" using 1:3 title "system" with lines ls 2, \
    "bench" using 1:2 title "user" with lines ls 3
```

Lister les répertoires

Système
Gilles Roussel
Les entrées sorties
Entrées sorties avec les fichiers
La bufferisation
Les répertoires
Les tubes
Entrées sorties les réseaux
Signal
Les des lignes

- Un répertoire est un fichier un peu particulier ; il n'est pas possible de l'ouvrir avec l'API standard
- `opendir()` permet d'obtenir descripteur ouvert sur un répertoire
- Des appels successifs à `readdir()` permettent de lire les entrées du répertoire
- `readdir()` retourne une structure `dirent` qui contient un champ `d_name` qui contient le nom d'une entrée du répertoire
- Attention, l'espace mémoire associé à `d_name` peut être réutilisé à chaque appel.
- `closedir()` permet de fermer un descripteur ouvert sur un répertoire

Exemple : `ls.c`

Système
Gilles Roussel
Les entrées sorties
Entrées sorties avec les fichiers
La bufferisation
Les répertoires
Les tubes
Entrées sorties les réseaux
Signal
Les des lignes

```
#include <sys/types.h>
#include <dirent.h>
#include <stdlib.h>
#include <stdio.h>

int main(int argc, char *argv[]) {
    DIR *dir;
    struct dirent *dirent;
    if ((dir=opendir(".")) == NULL) {
        perror("opendir"); exit(EXIT_FAILURE);
    }
    while((dirent=readdir(dir))!=NULL) {
        printf("%s\n", dirent->d_name);
    }
    if (closedir(dir) == -1) {
        perror("opendir"); exit(EXIT_FAILURE);
    }
    exit(EXIT_SUCCESS);
}
```

Créer un pipe en C

Système

Gilles Roussel
Les entrées-sorties
Entrées-sorties avec les fichiers
La buffering
Les répertoires
Les tubes
Entrées-sorties bas niveau
errno
Lien des slides

- `popen()` permet de lancer une commande et de récupérer un descripteur pour écrire vers l'entrée standard de la commande ou lire depuis la sortie standard de la commande
- Descripteurs bufferisés
- Comme `system()` la commande est lancée *via* un Shell
- `pclose()` permet de fermer le descripteur ainsi ouvert
- `pclose()` est bloquée tant que le processus lancé n'a pas terminé
- `pclose()` retourne la valeur de retour du processus

Gilles Roussel Système

Un exemple

Système

Gilles Roussel
Les entrées-sorties
Entrées-sorties avec les fichiers
La buffering
Les répertoires
Les tubes
Entrées-sorties bas niveau
errno
Lien des slides

```
#define _GNU_SOURCE
#include<stdio.h>
#include<stdlib.h>

int main(int argc, char* argv[]) {
    FILE *to_command;
    int result;
    if (NULL == (to_command = popen("cat","w"))) {
        perror("popen");
        exit(EXIT_FAILURE);
    }
    if (fprintf(to_command,"Test\n")<0) {
        perror("fprintf");
        exit(EXIT_FAILURE);
    }
    if ((result=pclose(to_command)) == -1) {
        perror("pclose");
        exit(EXIT_FAILURE);
    }
    exit(result);
}
```

Gilles Roussel Système

Survol des entrées-sorties bas niveau

Système

Gilles Roussel
Les entrées-sorties
Entrées-sorties avec les fichiers
La buffering
Les répertoires
Les tubes
Entrées-sorties bas niveau
errno
Lien des slides

- Unix propose une interface bas niveau pour les entrées-sorties
- Descripteurs sont des entiers, par exemple : `STDIN_FILENO`, `STDOUT_FILENO` et `STDERR_FILENO`
- `ssize_t write(int fd, void *buf, size_t n)`
 - tente d'écrire `n` octets pointés par `buf` sur le descripteur `fd`
 - retourne le nombre d'octets effectivement écrits ou `-1` en cas d'erreur; `errno` est également modifiée
- `ssize_t read(int fd, void *buf, size_t n)`
 - tente de lire `n` octets sur le descripteur `fd` et les place dans `buf`
 - retourne le nombre d'octets effectivement lus, `0` indique la fin de fichier et `-1` une erreur; `errno` est également modifiée
- Attention, `read()` (`write()` mode non bloquant) peuvent retourner un nombre d'octets différent de celui demandé

Gilles Roussel Système

Survol des entrées-sorties bas niveau

Système

Gilles Roussel
Les entrées-sorties
Entrées-sorties avec les fichiers
La buffering
Les répertoires
Les tubes
Entrées-sorties bas niveau
errno
Lien des slides

- `int open(const char *nom, int flags, mode_t mode)`
ou `int open(const char *nom, int flags)`
 - ouvre le fichier de nom `nom`
 - retourne un descripteur sur le fichier ouvert ou `-1` en cas d'erreur avec modification de `errno`
 - `flags` correspond à une combinaison de drapeaux indiquant le mode d'ouverture : `O_RDONLY` en lecture seule, `O_WRONLY` en écriture seule, `O_RDWR` en lecture et écriture, `O_CREAT` création du fichier s'il n'existe pas, `O_EXCL` avec `O_CREAT` création atomique du fichier qui ne doit pas exister, etc.
 - `mode` correspond au droit à donner au fichier en cas de création, en général `0666`
- `int close(int fd)` ferme le descripteur du fichier ouvert

Gilles Roussel Système

Survol des entrées-sorties bas niveau

Système

Gilles Roussel

Les entrées-sorties

Entrées-sorties avec les fichiers

La buffering

Les répertoires

Les tubes

Entrées-sorties bas niveau

map()

Liens des liens

cat.c

```
#include <stdlib.h>
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#define BUFFER_SIZE 1024
int main(int argc, char * argv[]) {
    char buffer[BUFFER_SIZE];
    int fd, nb;
    fd = open(argv[1],O_RDONLY);
    if (-1 == fd) { perror("open"); exit(EXIT_FAILURE); }
    while(0 != (nb = read(fd,buffer,BUFFER_SIZE))) {
        if (-1 == nb) { perror("read"); exit(EXIT_FAILURE); }
        nb = write(STDOUT_FILENO,buffer,nb);
        if (-1 == nb) { perror("write"); exit(EXIT_FAILURE); }
    }
    if (-1 == close(fd)) { perror("close"); exit(EXIT_FAILURE); }
    exit(EXIT_SUCCESS);
}
```

Gilles Roussel

Système

Survol des entrées-sorties bas niveau

Système

Gilles Roussel

Les entrées-sorties

Entrées-sorties avec les fichiers

La buffering

Les répertoires

Les tubes

Entrées-sorties bas niveau

map()

Liens des liens

- `off_t lseek(int fd, off_t offset, int from)`
 - décale de `offset` la position du descripteur `fd` à partir de `from`
 - `from` peut valoir : `SEEK_CUR` la position courante, `SEEK_SET` le début du fichier ou `SEEK_END` la fin du fichier
 - retourne la position courante du descripteur dans le fichier ou `-1` en cas d'erreur; `errno` est modifiée

Gilles Roussel

Système

Survol des entrées-sorties bas niveau

Système

Gilles Roussel

Les entrées-sorties

Entrées-sorties avec les fichiers

La buffering

Les répertoires

Les tubes

Entrées-sorties bas niveau

map()

Liens des liens

emptyfile.c

```
#include <stdlib.h>
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#define FILE_SIZE 100
int main(int argc, char * argv[]) {
    int fd;
    fd = open(argv[1],O_RDWR|O_CREAT|O_EXCL,0666);
    if (-1 == fd) {
        perror("open"); exit(EXIT_FAILURE);
    }
    if (-1 == lseek(fd,FILE_SIZE-1,SEEK_SET)) {
        perror("lseek"); exit(EXIT_FAILURE);
    }
    if (-1 == write(fd,"",1)) {
        perror("write"); exit(EXIT_FAILURE);
    }
    if (-1 == close(fd)) {
        perror("close"); exit(EXIT_FAILURE);
    }
    exit(EXIT_SUCCESS);
}
```

Gilles Roussel

Système

mmap()

Système

Gilles Roussel

Les entrées-sorties

Entrées-sorties avec les fichiers

La buffering

Les répertoires

Les tubes

Entrées-sorties bas niveau

map()

Liens des liens

- Permet de voir un fichier comme un tableau d'octets en mémoire
- Fichier chargé depuis le disque à la demande de façon paresseuse
- Gérer par `mmap()` et `munmap()`

Gilles Roussel

Système

mmap()

- `void *mmap(void *start, size_t length, int prot, int flags, int fd, off_t offset);`
 - `start` permet de préciser l'adresse en mémoire où placer le « début » du fichier ; 0 laisse le système choisir
 - `length` taille de la zone mémoire, peut être plus grande que la taille du fichier
 - `prot` droit d'accès à la zone : `PROT_EXEC`, `PROT_WRITE`, `PROT_READ` ou `PROT_NONE`
 - `flags` combinaison de mode de chargement du fichier : `MAP_SHARED` modification en mémoire visible sur le fichier, `MAP_PRIVATE` modification en mémoire non reportées sur le fichier et `MAP_FIXED` impose le chargement à l'adresse précisée en premier argument
 - `fd` descripteur du fichier à « mapper »
 - `offset` position du fichier à placer à l'adresse de début
 - retourne l'adresse où le fichier est « mappé » ou `(void*)-1` en cas d'erreur

Gilles Roussel Système

mmap()

- `int munmap(void *start, size_t length)`
 - `start` précise l'adresse du début de la zone mémoire à ne plus « mapper »
 - `length` précise la taille de la zone
 - retourne 0 ou -1 en cas d'erreur
 - `exit()` effectue un `munmap()` de toutes les zones « mappées »

Gilles Roussel Système

mmap()

WC.C

```
#include <stdlib.h>
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include <sys/mman.h>
int main(int argc, char * argv[] ) {
    int fd,size,i,result=0;
    char *ptr;
    fd = open(argv[1],O_RDWR);
    if (-1 == fd) { perror("open"); exit(EXIT_FAILURE); }
    size = lseek(fd,0,SEEK_END);
    if (-1 == size) { perror("open"); exit(EXIT_FAILURE); }
    ptr = mmap(0,size,PROT_READ,MAP_PRIVATE,fd,0);
    if ((void*) -1 == ptr) {perror("mmap"); exit(EXIT_FAILURE); }
    close(fd);
    for (i=0;i<size;i++) {
        if (ptr[i]=='\n') { result++; }
    }
    if (-1==munmap(ptr,size)) { perror("munmap"); exit(EXIT_FAILURE); }
    printf("%d\n",result);
    exit(EXIT_SUCCESS);
}
```

Gilles Roussel Système

Lire ligne par ligne

- Il ne faut pas utiliser `gets()`
- `fgets()` pas si facile à utiliser pour être sûr de lire une ligne non tronquée (entière)
`char *fgets(char *s, int size, FILE *stream);`
- Bibliothèque non-standard (GNU) `libreadline` permet de simplifier la lecture d'une ligne

Gilles Roussel Système

Comment installer une nouvelle bibliothèque ?

- Suite de commandes standard
- Récupérer l'archive des sources : `readline-6.2.tar.gz`
- Décompresser et extraire l'archive avec la commande `tar` (z décompression, x extraction, v verbeux, f depuis un fichier):

```
# tar zxvf readline-6.2.tar.gz
```
- Configurer par rapport à la plate-forme :

```
# ./configure
```

Éventuellement en précisant un répertoire d'installation

```
# ./configure --prefix=${HOME}/lib
```
- Compiler :

```
# make
```
- Installer :

```
# make install
```

Gilles Roussel Système

Utiliser readline

- `readline()` alloue automatiquement la ligne qu'il vient de lire
- Il faut désallouer les lignes !
- `readline()` fournit un ensemble de raccourcis pour la gestion de ligne
- `readline()` permet de sauver les lignes et de les rappeler

Gilles Roussel Système

Un exemple

readline.c

```
#include <stdio.h>
#include <readline/readline.h>
#include <readline/history.h>
#include <stdlib.h>

int main(int argc, char *argv[]) {
    char *line;
    while(NULL!=(line = readline("prompt# "))) {
        if (*line) {
            add_history(line);
        }
        if (fprintf(stdout,"%s\n",line) < 0) {
            break;
        }
        free(line);
    }
    exit(EXIT_SUCCESS);
}
```

```
# gcc -o readline readline.c -I$HOME/lib/include -L$HOME/lib/lib: -lreadline -ltermcap
# LD_LIBRARY_PATH=$HOME/lib ./readline
```

Gilles Roussel Système