Université Joseph Fourier, Grenoble
Licence d'Informatique L3

Introduction aux Systèmes et Réseaux

Document technique n°2 : Bibliothèques d'entrées-sorties

1 Introduction

Dans ce cours, nous avons évoqué plusieurs classes de primitives pour la manipulation de fichiers :

- les opérations de base (appels système) fournis par le noyau du système d'exploitation (`open`, `read`, `write`, `close`, ...) ;
- les opérations fournies par la bibliothèque C standard (`fopen`, `fread`, `fwrite`, `fscanf`, `fprintf`, ...) ;

L'objectif de ce document est double. Il vise, d'une part, à préciser les différences entre les types de primitives de manipulation des fichiers décrits ci-dessus. D'autre part, il explique les restrictions inhérentes à ces primitives pour la manipulation de fichiers (surtout les fichiers spéciaux tels que les tubes et *sockets*) et présente la bibliothèque RIO, qui libère le programmeur d'applications de la gestion de certaines difficultés.

2 Primitives d'E/S de la bibliothèque standard

Les primitives de la bibliothèque standard (`fopen`, `fread`, `fwrite`, `fprintf` ...) sont basées sur les primitives élémentaires du système (appels système : `open`, `read`, `write` ...) et manipulent des flux de données (représentés par un pointeur de type `FILE*`). Un flux de données est associé à un descripteur de fichier ainsi qu'à un tampon (qui n'est pas visible ni manipulé directement par le programmeur). Le tampon sert à minimiser le nombre d'appels aux primitives du système d'exploitation (qui peuvent pénaliser les performances d'une application).

On peut obtenir (une référence vers) un flux à partir d'un descripteur de fichier via la primitive `fdopen` (ou directement à partir d'un nom de fichier via `fopen`). Inversement, on peut obtenir le numéro du descripteur de fichier associé à un flux via la primitive `fileno`. Les (pointeurs vers les) flux associés à l'entrée standard, la sortie standard et la sortie d'erreur sont respectivement `stdin`, `stdout` et `stderr` [1]. Un appel à `fprintf(stdout, ...)` est équivalent à `printf(...)`. De même, un appel à `fscanf(stdin, ...)` est équivalent à un appel à `scanf(...)`.

Il y a plusieurs modes possibles pour le déclenchement de l'échange de données entre le tampon interne à un flux et le noyau du système d'exploitation : la condition peut-être le remplissage/vidage complet du tampon ou l'arrivée d'un caractère de fin de ligne. On peut aussi désactiver le tampon. En général, par défaut, l'entrée et la sortie standard sont tamponnées en mode ligne, la sortie d'erreur n'est pas tamponnée et tous les autres flux sont tamponnés en mode complet. En conséquence, certaines entrées-sorties peuvent se

1. Les numéros de descripteurs correspondants sont respectivement associés aux constantes suivantes : `STDIN_FILENO`, `STDOUT_FILENO`, `STDERR_FILENO`, dont les valeurs numériques sont généralement 0, 1 et 2.

produire plus tard que ce qui est escompté par le programmeur, et le résultat peut s'avérer déroutant (traces désordonnées ou incohérentes pour une application multiprocessus ...). Pour remédier à ce problème dans le cas d'une écriture, on peut utiliser un caractère retour chariot (`\n`) à la fin d'une E/S formatée (dans le cas d'un flux de sortie tamponné en mode ligne) ou, plus généralement, forcer le vidage immédiat d'un tampon à écrire via la primitive `fflush`.

Les primitives d'entrées-sorties de la bibliothèque standard sont très pratiques mais posent certains problèmes [2] pour les communications par *sockets* ou tubes. Leur utilisation est donc déconseillée pour l'envoi ou la réception de données sur ce type de canaux de communication. Il est recommandé d'utiliser plutôt les primitives RIO dans ce cas. Si vous avez besoin de manipuler des données formatées, vous pouvez néanmoins utiliser `snprintf` (resp. `sscanf`) avant de les envoyer (resp. après les avoir reçues) via une *socket* ou un tube [3].

3 La bibliothèque RIO

On a vu en cours les opérations d'entrée-sortie sur les fichiers : `read` et `write`. On présente ici un ensemble d'opérations dites "robustes" qui utilisent `read` et `write` mais fournissent une interface plus commode et plus efficace. Elles sont utilisées pour les fichiers et les tubes, mais également pour les communications par *sockets* sur les réseaux. Ces opérations constituent la bibliothèque RIO [4] (*Robust Input-Output*). Le programme de ces opérations fait partie du fichier `csapp.c` disponible dans le placard ; voir aussi `csapp.h`.

L'objectif de cette présentation est double :

1. point de vue de l'utilisateur : présenter des fonctions plus sûres et plus commodes que les primitives élémentaires,
2. point de vue du concepteur : en examinant le détail de la programmation de ces fonctions, mieux comprendre le fonctionnement des entrées-sorties.

Tout le monde est intéressé par le premier aspect (ces fonctions seront réutilisées lors du TP sur les *sockets* et de la dernière Apnée) ; selon votre intérêt vous pourrez regarder plus ou moins en détail la réalisation.

Un résumé de l'ensemble des primitives est disponible en section 3.4.

2. Ces problèmes ne seront pas décrits en détails dans le cadre de ce cours. Des explications sont fournies dans le livre *Computer Systems : a Programmer's Perspective*.

3. Comme `printf`, `snprintf` produit une chaîne de caractères formatée mais au lieu d'être envoyée sur la sortie standard, cette chaîne est stockée dans un tampon. Selon le même principe, `sscanf` effectue une lecture formatée à partir d'une chaîne stockée en mémoire.

4. La bibliothèque RIO a été développée par W. R. Stevens (*Unix Network Programming*, vol. 1, Prentice Hall, 1998). La version donnée ici est celle de R. E. Bryant et D. O'Hallaron (*Computer Systems : a Programmer's Perspective*, Prentice Hall, 2003).

3.1 Motivation

La bibliothèque RIO répond à deux problèmes posés par les appels systèmes `read` et `write`, qui compliquent la tâche du programmeur et obscurcissent le code d'une application.

Appels système interruptibles Un processus bloqué en attente de la terminaison d'un appel système en cours (par exemple, une lecture dans un tube ou une *socket* via `read`) peut être interrompu par l'arrivée d'un signal (ou, plus précisément, par l'exécution du traitant de signal correspondant). Dans ce cas (si le traitant de signal n'a pas eu pour effet de détruire le processus), l'appel système se termine prématurément, retourne -1 et positionne la variable `errno` à la valeur `EINTR`. Le programmeur doit explicitement gérer ce cas dans son code : il s'agit typiquement de tester la valeur de retour de la primitive (et éventuellement `errno`) et de relancer l'opération si nécessaire.

Attente d'une quantité minimale de données Lorsqu'un programme effectue une lecture sur un descripteur de fichier particulier (clavier du terminal, sortie d'un tube, *socket*), l'émetteur des données (un autre processus dans le cas d'un tube et d'une *socket* ou bien l'utilisateur pour le clavier) peut produire ces données assez lentement et/ou de manière progressive pour diverses raisons. En conséquence, un appel à `read` peut retourner un nombre d'octets lus (`res`) inférieur à celui indiqué en paramètre (`count`). Dans de nombreux cas, le processus lecteur a vraiment besoin d'obtenir le nombre d'octets initialement demandé (par exemple, ce nombre peut correspondre à la taille fixe d'un en-tête de message) afin de poursuivre son traitement. Le programmeur doit donc prendre cette situation en compte et effectuer un nouvel appel à `read` en spécifiant le nombre d'octets restant à obtenir.

La bibliothèque RIO permet de décharger les programmeurs d'applications de la prise en compte des contraintes évoquées ci-dessus.

3.2 Interface de programmation

Il y a deux versions des opérations Rio : lecture et écriture non tamponnées (*unbuffered*) et lecture tamponnée (*buffered*).

3.2.1 Opérations non tamponnées

Les opérations non tamponnées sont notamment utiles pour la lecture et l'écriture de données sur un canal de communication réseau (*socket*). Elles ont comme interface :

```
#include "csapp.h"
ssize_t rio_readn(int fd, void *usrbuf, size_t n);
ssize_t rio_writen(int fd, void *usrbuf, size_t n);
```

Elles transfèrent (au plus) `n` octets entre le fichier de descripteur `fd` et le tampon `usrbuf`. Elles renvoient le nombre exact d'octets lus et écrits, et -1 en cas d'erreur. Noter que le type `ssize_t` est celui des entiers avec signe (pour inclure la valeur -1 signalant une erreur) alors que le type `size_t` est celui des entiers positifs ou nuls.

3.2.2 Opérations tamponnées

Les primitives de lecture tamponnée utilisent un tampon de lecture intermédiaire pour stocker des octets. Pour utiliser une telle primitive, le programmeur doit déclarer un tampon (de type `rio_t`) et l'associer à un descripteur de fichier ouvert. Cette association est réalisée par un appel à la fonction :

```
void rio_readinitb(rio_t *rp, int fd)
```

Les primitives de lecture tamponnée ont pour interface :

```
ssize_t rio_readnb(rio_t *rp, void *usrbuf, size_t n);
ssize_t rio_readlineb(rio_t *rp, void *usrbuf, size_t maxlen);
```

On peut remarquer que les noms de ces primitives se terminent par la lettre `b` (pour *buffered*). La fonction `rio_readnb(rio_t *rp, void *usrbuf, size_t n)` lit au plus `n` octets depuis un fichier (associé au tampon `rp`) vers un tampon destinataire `usrbuf`.

La fonction `rio_readlineb(rio_t *rp, void *usrbuf, size_t maxlen)` interprète les octets lus comme des caractères. Elle lit une ligne de texte (y compris le caractère de fin de ligne) depuis un fichier (associé au tampon `rp`) vers un tampon destinataire `usrbuf`. Un caractère `NULL` est placé en fin du texte. La fonction lit au plus `maxlen` -1 caractères (si la ligne lue est plus longue, elle est tronquée à `maxlen` -1, et toujours terminée par `NULL`).

3.3 Détails d'implémentation

3.3.1 Opérations non tamponnées

Voici le programme des procédures `rio_readn` et `rio_writen`. Examiner le fonctionnement de ces opérations. Noter que chacune des fonctions teste les cas d'erreur et redémarre l'entrée-sortie si celle-ci a été interrompue par un signal.

```
ssize_t rio_readn(int fd, void *usrbuf, size_t n) {
    size_t nleft = n;
    ssize_t nread;
    char *bufp = usrbuf;

    while (nleft > 0) {
        if ((nread = read(fd, bufp, nleft)) < 0) {
            if (errno == EINTR) /* interrupted by sig handler return */
                nread = 0;      /* and call read() again */
            else
                return -1;      /* errno set by read() */
        }
        else if (nread == 0)
            break;              /* EOF */
        nleft -= nread;
        bufp += nread;
    }
    return (n - nleft);         /* return >= 0 */
}
```

```
ssize_t rio_writen(int fd, void *usrbuf, size_t n) {
    size_t nleft = n;
    ssize_t nwritten;
    char *bufp = usrbuf;

    while (nleft > 0) {
        if ((nwritten = write(fd, bufp, nleft)) <= 0) {
            if (errno == EINTR)  /* interrupted by sig handler return */
                nwritten = 0;    /* and call write() again */
            else
                return -1;       /* errorno set by write() */
        }
        nleft -= nwritten;
        bufp += nwritten;
    }
    return n;
}
```

3.3.2 Opérations tamponnées

Les primitives de lecture tamponnée utilisent un tampon de lecture intermédiaire pour stocker des octets. Ce tampon est défini par le format suivant :

```
#define RIO_BUFSIZE 8192
typedef struct {
    int rio_fd;                /* descriptor for this internal buf */
    int rio_cnt;               /* unread bytes in internal buf */
    char *rio_bufptr;          /* next unread byte in internal buf */
    char rio_buf[RIO_BUFSIZE]; /* internal buffer */
} rio_t;
```

Le tampon est initialisé par la fonction ci-après.

```
void rio_readinitb(rio_t *rp, int fd) {
    rp->rio_fd = fd;
    rp->rio_cnt = 0;
    rp->rio_bufptr = rp->rio_buf;
}
```

Cette primitive associe un descripteur de fichier `fd` à un tampon de lecture de type `rio_t` situé à l'adresse `rp`. Elle doit être appelée une fois pour chaque descripteur ouvert.

Le programme des fonctions de lecture tamponnée est donné ci-après. Ces fonctions utilisent une primitive commune de lecture `rio_read` (dont le programme figure à la fin), qui a la même sémantique que la primitive `read`, mais qui lit dans le tampon et réalimente celui-ci à partir du fichier quand il est vide, en appelant `read`.

```
/* lecture tamponnée d'une ligne */
ssize_t rio_readlineb(rio_t *rp, void *usrbuf, size_t maxlen) {
    int n, rc;
    char c, *bufp = usrbuf;

    for (n = 1; n < maxlen; n++) {
        if ((rc = rio_read(rp, &c, 1)) == 1) {
            *bufp++ = c;
            if (c == '\n')
                break;
        } else if (rc == 0) {
            if (n == 1)
                return 0; /* EOF, no data read */
            else
                break;    /* EOF, some data was read */
        } else
            return -1;    /* error */
    }
    *bufp = 0;
    return n;
}

/* lecture tamponnée d'une suite d'octets */
ssize_t rio_readnb(rio_t *rp, void *usrbuf, size_t n) {
    size_t nleft = n;
    ssize_t nread;
    char *bufp = usrbuf;
    while (nleft > 0) {
        if ((nread = rio_read(rp, bufp, nleft)) < 0) {
            if (errno == EINTR) /* interrupted by sig handler return */
                nread = 0;      /* call read() again */
            else
                return -1;      /* errno set by read() */
        }
        else if (nread == 0)
            break;              /* EOF */
        nleft -= nread;
        bufp += nread;
    }
    return (n - nleft);         /* return >= 0 */
}
```

```
/*
 * cette fonction est une enveloppe (wrapper) pour la primitive read.
 * Elle transfère min(n, rio_cnt) octets depuis un tampon interne vers
 * un tampon chez l'utilisateur. n est le nombre d'octets dont le
 * transfert est demandé, et rio_cnt le nombre d'octets non lus dans
 * le tampon interne. rio_read remplit le tampon (en utilisant read) si
 * elle trouve le tampon vide.
*/
static ssize_t rio_read(rio_t *rp, char *usrbuf, size_t n) {
    int cnt;

    while (rp->rio_cnt <= 0) {  /* refill if buf is empty */
        rp->rio_cnt = read(rp->rio_fd, rp->rio_buf,
                           sizeof(rp->rio_buf));
        if (rp->rio_cnt < 0) {
            if (errno != EINTR) /* interrupted by sig handler return */
                return -1;
        }
        else if (rp->rio_cnt == 0)  /* EOF */
            return 0;
        else
            rp->rio_bufptr = rp->rio_buf; /* reset buffer ptr */
    }
    /* Copy min(n, rp->rio_cnt) bytes from internal buf to user buf */
    cnt = n;
    if (rp->rio_cnt < n)
        cnt = rp->rio_cnt;
    memcpy(usrbuf, rp->rio_bufptr, cnt);
    rp->rio_bufptr += cnt;
    rp->rio_cnt -= cnt;
    return cnt;
}
```

3.4 Résumé des primitives de la bibliothèque RIO

⚠ *Attention aux différences entre les primitives dont le nom commence par une majuscule et celles dont le nom commence par une minuscule.*

3.4.1 Écriture non tamponnée

`ssize_t rio_writen(int fd, void *usrbuf, size_t n)` :
Tente d'écrire d'écrire `n` octets dans le fichier associé au descripteur `fd`. Retourne -1 (et modifie la variable `errno`) en cas d'erreur. Sinon, retourne `n`.

`void Rio_writen(int fd, void *usrbuf, size_t n)` :
Tente d'écrire d'écrire `n` octets dans le fichier associé au descripteur `fd`. Affiche un message et arrête le programme en cas d'erreur.

3.4.2 Lecture non tamponnée

`ssize_t rio_readn(int fd, void *usrbuf, size_t n)` :
Tente de lire `n` octets dans le fichier associé au descripteur `fd`. Retourne -1 (et modifie la variable `errno`) en cas d'erreur. Sinon, retourne le nombre d'octets effectivement lus (ce nombre peut-être inférieur à `n` si le descripteur correspond à un tube ou une *socket* dont l'autre extrémité a été fermée).

`ssize_t Rio_readn(int fd, void *usrbuf, size_t n)` :
Identique à `rio_readn` mais affiche un message et arrête le programme en cas d'erreur.

3.4.3 Lecture tamponnée

`void rio_readinitb(rio_t *rp, int fd)` :
Associe un tampon à un descripteur de fichier. Strictement identique à `Rio_readinitb`.

`ssize_t rio_readlineb(rio_t *rp, void *usrbuf, size_t maxlen)` :
Tente de lire une ligne (délimitée par `\n`) dans le tampon (qui sera rechargé au préalable si nécessaire). Retourne -1 (et modifie la variable `errno`) en cas d'erreur. Retourne 0 si aucun caractère n'a pu être lu. Sinon, retourne le nombre de caractères lus. La ligne lue ne peut dépasser `maxlen-1` caractères (elle est tronquée si nécessaire). Le nombre (`n`) de caractères retourné inclue le délimiteur `\n` (s'il est présent). L'octet `usrbuf[n]` est rempli avec le délimiteur de fin de chaîne `\0` (cet octet n'est pas inclus dans le nombre d'octets retourné par la fonction).

`ssize_t Rio_readlineb(rio_t *rp, void *usrbuf, size_t maxlen)` :
Identique à `rio_readlineb` mais affiche un message et arrête le programme en cas d'erreur.

`ssize_t rio_readnb(rio_t *rp, void *usrbuf, size_t n)` :
Tente de lire `n` octets dans le tampon (qui sera rechargé au préalable si nécessaire). Retourne -1 (et modifie la variable `errno`) en cas d'erreur. Sinon, retourne le nombre d'octets effectivement lus (ce nombre peut-être inférieur à `n` si le descripteur correspond à un tube ou une *socket* dont l'autre extrémité a été fermée).

`ssize_t Rio_readnb(rio_t *rp, void *usrbuf, size_t n)` :
Identique à `rio_readlineb` mais affiche un message et arrête le programme en cas d'erreur.