Programación de Sistemas CCPG1008

Federico Domínguez, PhD.

Unidad 5 – Sesión 4: Gestión de llamadas I/O

Contenido

1. Funciones “*wrapper*” para I/O
2. Gestión de errores
3. Metadata de archivos

Funciones "*wrapper*" para I/O

En algunas casos, funciones de bajo nivel como *read* y *write* retornan **sin haber completado su tarea**.

Si *read* o *write* retornan un valor menor a *n*

- la tarea de transferencia de información esta incompleta, hay que hacer algo!
- Causas: EOF en *read*, interrupciones de sistema, errores

```
#include <unistd.h>

ssize_t read(int fd, void *buf, size_t n);
                    Returns: number of bytes read if OK, 0 on EOF, −1 on error

ssize_t write(int fd, const void *buf, size_t n);
                    Returns: number of bytes written if OK, −1 on error
```

El libro de texto proporciona ejemplos **simples y robustos** de funciones "*wrapper*" para *read* y *write*.

Implementadas en *csapp.c* (código proporcionado libremente por los autores de *Computer Systems: A Programmer's Perspective* en CSAPP) muestran como gestionar algunos eventos durante las llamadas de *read* y *write*.

```
#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);
```

Returns: number of bytes transferred if OK, 0 on EOF (`rio_readn` only), −1 on error

code/src/csapp.c

```
1   ssize_t rio_readn(int fd, void *usrbuf, size_t n)
2   {
3       size_t nleft = n;
4       ssize_t nread;
5       char *bufp = usrbuf;
6
7       while (nleft > 0) {
8           if ((nread = read(fd, bufp, nleft)) < 0) {
9               if (errno == EINTR) /* Interrupted by sig handler return */
10                  nread = 0;      /* and call read() again */
11              else
12                  return -1;      /* errno set by read() */
13          }
14          else if (nread == 0)
15              break;              /* EOF */
16          nleft -= nread;
17          bufp += nread;
18      }
19      return (n - nleft);         /* Return >= 0 */
20  }
```

code/src/csapp.c

La función *rio_read* usa un buffer interno para **minimizar** las llamadas a *read*.

code/src/csapp.c

```
1  static ssize_t rio_read(rio_t *rp, char *usrbuf, size_t n)
2  {
3      int cnt;
4
5      while (rp->rio_cnt <= 0) {  /* Refill if buf is empty */
6          rp->rio_cnt = read(rp->rio_fd, rp->rio_buf,
7                             sizeof(rp->rio_buf));
8          if (rp->rio_cnt < 0) {
9              if (errno != EINTR) /* Interrupted by sig handler return */
10                 return -1;
11         }
12         else if (rp->rio_cnt == 0)  /* EOF */
13             return 0;
14         else
15             rp->rio_bufptr = rp->rio_buf; /* Reset buffer ptr */
16     }
17
18     /* Copy min(n, rp->rio_cnt) bytes from internal buf to user buf */
19     cnt = n;
20     if (rp->rio_cnt < n)
21         cnt = rp->rio_cnt;
22     memcpy(usrbuf, rp->rio_bufptr, cnt);
23     rp->rio_bufptr += cnt;
24     rp->rio_cnt -= cnt;
25     return cnt;
26 }
```

code/src/csapp.c

También se proveen funciones "*wrapper*" que usan *rio_read* en lugar de *read* para minimizar el uso de llamadas I/O.

Funciones como *rio_readlineb* leen una línea completa de texto de un archivo o *socket* usando el carácter '\n' como delimitador de línea.

```
#include "csapp.h"

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

Returns: nothing

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

Returns: number of bytes read if OK, 0 on EOF, −1 on error

Gestión de errores

Una función de sistema de Linux usa la variable global *errno* para indicar errores en tiempo de ejecución.

Una función de sistema de Linux usualmente retorna -1 cuando encuentra un error.

Idealmente, al recibir -1, un programa debería revisar el contenido de la variable *errno (en errno.h)*.

```
1        if ((pid = fork()) < 0) {
2            fprintf(stderr, "fork error: %s\n", strerror(errno));
3            exit(0);
4        }
```

La función *strerror* (en *string.h*) retorna una descripción del valor en la variable *errno*.

Es posible simplificar la gestión de errores usando funciones “*wrapper*”.

```
1   void unix_error(char *msg) /* Unix-style error */
2   {
3       fprintf(stderr, "%s: %s\n", msg, strerror(errno));
4       exit(0);
5   }
```

```
1       if ((pid = fork()) < 0)
2           unix_error("fork error");
```

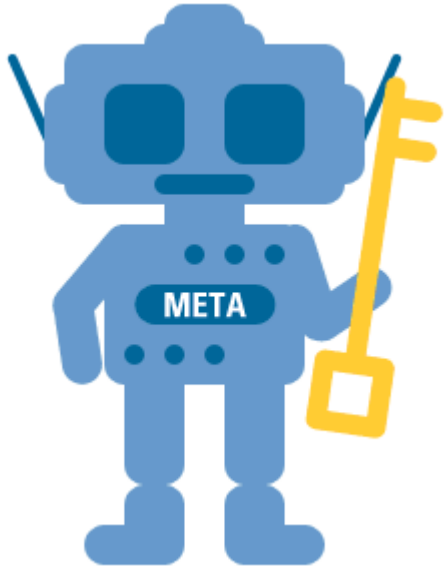
En *csapp.c* se definen funciones "*wrapper*" de llamadas al sistema con gestión de errores.

```
1   pid_t Fork(void)
2   {
3       pid_t pid;
4
5       if ((pid = fork()) < 0)
6           unix_error("Fork error");
7       return pid;
8   }
```

Llamadas al sistema con gestión de error tienen el nombre con mayúscula.

```
1       pid = Fork();
```

Metadata de archivos



Información como tamaño o permisos de un archivo puede ser obtenida usando *stat* y *fstat*.

Ambas funciones llenan una estructura *stat* con información sobre el archivo.

```
#include <unistd.h>
#include <sys/stat.h>

int stat(const char *filename, struct stat *buf);
int fstat(int fd, struct stat *buf);
```

Returns: 0 if OK, −1 on error

statbuf.h (included by sys/stat.h)

```
/* Metadata returned by the stat and fstat functions */
struct stat {
    dev_t         st_dev;      /* Device */
    ino_t         st_ino;      /* inode */
    mode_t        st_mode;     /* Protection and file type */
    nlink_t       st_nlink;    /* Number of hard links */
    uid_t         st_uid;      /* User ID of owner */
    gid_t         st_gid;      /* Group ID of owner */
    dev_t         st_rdev;     /* Device type (if inode device) */
    off_t         st_size;     /* Total size, in bytes */
    unsigned long st_blksize;  /* Blocksize for filesystem I/O */
    unsigned long st_blocks;   /* Number of blocks allocated */
    time_t        st_atime;    /* Time of last access */
    time_t        st_mtime;    /* Time of last modification */
    time_t        st_ctime;    /* Time of last change */
};
```

statbuf.h (included by sys/stat.h)

```
if(stat(filename,&sbuf) < 0) //Verificar si el archivo existe
        printf("Archivo no existe!\n");
else
        s = sbuf.st_size; //Guardar el tamaño del archivo en s
```

Referencias

Libro texto *Computer Systems: A programmers perspective*. Secciones 10.4 – 5 (en 2da edición)