Sistema de Entrada/Saída

Ficheiros

José Pedro Oliveira
(jpo@di.uminho.pt)

Grupo de Sistemas Distribuídos
Departamento de Informática
Escola de Engenharia
Universidade do Minho

Sistemas Operativos
2006-2007

Conteúdo

Descritores de ficheiros

Kernel e ficheiros

- No kernel todos os ficheiros abertos são referenciados como FD (*File Descriptors*)
- Um FD é um número inteiro não negativo
- Sempre que um processo abre um ficheiro, o kernel retorna um FD
- Qualquer chamada ao sistema que opere sobre um ficheiro deve receber um FD como argumento (excepções: `open` e `creat`).

Descritores de ficheiros standard

Descritores de ficheiros standard

Por convenção, os interpretadores de comandos (*shells*) em UNIX associam:

- o descritor 0 (zero) ao *standard input*,
- o descritor 1 (um) ao *standard output*,
- o descritor 2 (dois) ao *standard error*;

Constantes POSIX

- `STDIN_FILENO`,
- `STDOUT_FILENO`,
- `STDERR_FILENO`;

Nota: constantes definidas no ficheiro de *header* `<unistd.h>`

Conteúdo

Chamadas ao sistema (*system calls*)

Chamadas ao sistema

open	- abrir ou criar ficheiro
creat	- criar ficheiro
close	- fechar descritor de ficheiro
read	- ler a partir de um descritor de ficheiro
write	- escrever para um descritor de ficheiro
lseek	- reposicionar *offset* de leitura/escrita
stat	- obter informação sobre ficheiro
fcntl	- manipular o descritor de ficheiro
ioctl	- controlo de dispositivo
...	- ...

Chamadas ao sistema: valor de retorno

Valor de retorno das chamadas ao sistema

As invocações das chamadas ao sistema básicas retornam, quase universalmente, o valor -1 em caso de erro e 0 ou um número positivo no caso de sucesso.
Resumindo:

erro	- retornam o valor -1
sucesso	- retornam um valor maior ou igual a 0

Que erro aconteceu?

Quando uma chamada ao sistema retorna o valor -1, deve ser consultada a variável **errno**, declarada no ficheiro de *header* <**errno.h**>, para se obter informação mais detalhada sobre o erro.

Chamadas ao sistemas **open** e **creat**

Synopsis

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

int open(const char *pathname, int flags);
int open(const char *pathname, int flags, mode_t mode);
int creat(const char *pathname, mode_t mode);
```

creat vs **open**

A chamada ao sistema `creat` é equivalente a

```
open(pathname, O_WRONLY | O_CREAT | O_TRUNC, mode)
```

Chamadas ao sistemas **open** e **creat**

Parâmetro **flags**

Flag obrigatória:

- O_RDONLY - abre o ficheiro em modo de leitura
- O_WRONLY - abre o ficheiro em modo de escrita
- O_RDWR - abre o ficheiro em modo de leitura/escrita

Flags opcionais:

- O_CREAT - cria o ficheiro se este não existir
- O_TRUNC - se o ficheiro já existir, se for um ficheiro regular e se for aberto num modo que permita operações de escrita, será truncado a 0 (zero)
- O_APPEND - abre o ficheiro em modo de *append*
- O_SYNC - abre o ficheiro em modo síncrono
- ... - ...

Chamada ao sistema **close**

Synopsis

```
#include <unistd.h>

int close(int fd);
```

Notas sobre terminação de processos

Quando um processo termina, o kernel

- fecha automaticamente todos os ficheiros abertos e
- liberta também todos os *locks* de registo (*record locks*)

a ele associados.

Chamada ao sistema **read**

Synopsis

```
#include <unistd.h>

ssize_t read(int fd, void *buf, size_t count);
```

Valor de retorno

Em caso de sucesso retorna o número de bytes lidos ou 0 na presença de fim do ficheiro. Em caso de erro retorna -1.

Chamada ao sistema **write**

Synopsis

```
#include <unistd.h>

ssize_t write(int fd, const void *buf, size_t count);
```

Valor de retorno

Em caso de sucesso retorna o número de bytes escritos. Em caso de erro retorna -1.

Chamada ao sistema **write**: exemplo

Chamada ao sistema

```
1 #include <unistd.h>
2 int main(void)
3 {
4     write(STDOUT_FILENO, "Ola\n", 4);
5     return 0;
6 }
```

Biblioteca C

```
1 #include <stdio.h>
2 int main(void)
3 {
4     printf("Ola\n");
5     return 0;
6 }
```

Exemplo: criação de um novo ficheiro (1)

Criação de um novo ficheiro (sem tratamento de erros)

```
#include <unistd.h>
#include <fcntl.h>

int main(void)
{
    int fd;

    fd = open("/tmp/teste.txt",
              O_WRONLY | O_CREAT, 0644);
    write(fd, "Exemplo\n", 8);
    close(fd);

    return 0;
}
```

Exemplo: criação de um novo ficheiro (2)

Criação de um novo ficheiro

```
#include <unistd.h>
#include <fcntl.h>

int main(void)
{
    int fd;

    fd = open("/tmp/teste.txt",
              O_WRONLY | O_CREAT | O_TRUNC, 0644);
    write(fd, "Exemplo\n", 8);
    close(fd);

    return 0;
}
```

Exemplo: criação de um novo ficheiro (3)

Criação de um novo ficheiro

```
#include <unistd.h>
#include <fcntl.h>

int main(void)
{
    int fd;

    fd = creat("/tmp/teste.txt", 0644);
    write(fd, "Exemplo\n", 8);
    close(fd);

    return 0;
}
```

Exemplo: leitura de um ficheiro

Listar o conteúdo do ficheiro `/etc/passwd`

```c
#include <unistd.h>
#include <fcntl.h>
#define MAXBUFSIZE  256

int main(void)
{
    int fd, n;
    char buffer[MAXBUFSIZE];

    fd = open("/etc/passwd", O_RDONLY);
    while ( (n = read(fd, buffer, MAXBUFSIZE)) > 0) {
        write(STDOUT_FILENO, buffer, n);
    }
    close(fd);

    return 0;
}
```

Valores de retorno de chamadas ao sistema

Importante

Todos os valores de retorno de chamadas ao sistema devem ser verificados.

Tratamento de erros: informação adicional

- man 3 errno
- man 3 perror
- man 3 strerror

Lista de tarefas

Tarefas

- Testar todos os exemplos anteriores. Será que todos os programas executam correctamente quando executados diversas vezes?
- Executar os programas anteriores com a ferramenta `strace`.
- Acrescentar tratamento de erros a todos os programas anteriores.

Chamada ao sistema **lseek**

Synopsis

```c
#include <sys/types.h>
#include <unistd.h>

off_t lseek(int fildes, off_t offset, int whence);
```

Nota

Esta chamada ao sistema não provoca nenhuma operação de Entrada/Saída (Input/Output); só altera o *offset* da próxima operação de E/S.

Chamada ao sistema **lseek**

Parâmetro **whence**

O cálculo da nova posição é efectuado:

SEEK_SET - a partir do ínicio do ficheiro

SEEK_CUR - a partir da posição corrente

SEEK_END - a partir do fim do ficheiro

Determinar *offset* corrente

```
off_corrente = lseek(fd, 0, SEEK_CUR);
```

Chamadas ao sistema da família **stat**

Synopsis

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

int stat(const char *file_name, struct stat *buf);
int fstat(int filedes, struct stat *buf);
int lstat(const char *file_name, struct stat *buf);
```

Chamadas ao sistema da família **stat**: estrutura

Estrutura `stat`

```
struct stat {
    dev_t     st_dev;     /* device */
    ino_t     st_ino;     /* inode */
    mode_t    st_mode;    /* protection */
    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 */
    blksize_t st_blksize; /* blocksize for filesystem I/O */
    blkcnt_t  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 status change */
};
```

Conteúdo

Criar ficheiro com buraco

Exemplo

```c
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>

int main(void)
{
    int fd = creat("/tmp/hole.dat", 0644);

    write(fd, "ABC\n", 4);

    lseek(fd, 50000, SEEK_SET);

    write(fd, "DEF\n", 4);

    close(fd);

    return 0;
}
```

Criar ficheiro com buraco

```
$ ls -l /tmp/hole.dat
-rw-r--r-- 1 jpo jpo 50004 Nov 15 01:18 /tmp/hole.dat
```

```
$ ls -ls /tmp/hole.dat
16 -rw-r--r-- 1 jpo jpo 50004 Nov 15 01:18 /tmp/hole.dat
```

```
$ du -sk /tmp/hole.dat
16      /tmp/hole.dat
```

Outros comandos - informação sobre sistemas de ficheiros

- `fdisk -l /dev/hda`
- `tune2fs -l /dev/hda1`

Criar ficheiro de acesso directo (1/2)

Parte 1/2

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

static struct registo {
    char codigo[8];
    char nome[20];
} array[] = {
    {"400", "abc"},
    {"500", "def"},
    {"300", "ghi"},
    {"350", "jkl"}
};
```

Criar ficheiro de acesso directo (2/2)

Parte 2/2

```c
int main(void)
{
    int fd, i;

    if ((fd = open("/tmp/reg.dat",
                   O_CREAT | O_WRONLY, 0644)) == -1) {
        perror("open");
        exit(EXIT_FAILURE);
    }

    for (i=0; i<sizeof(array)/sizeof(struct registo); i++) {
        write(fd, &array[i], sizeof(struct registo));
    }

    close(fd);

    return 0;
}
```

Conteúdo do ficheiro de acesso directo

```
$ od  -c  --width=8  /tmp/reg.dat

0000000    4    0    0   \0   \0   \0   \0   \0
0000010    a    b    c   \0   \0   \0   \0   \0
0000020   \0   \0   \0   \0   \0   \0   \0   \0
0000030   \0   \0   \0   \0    5    0    0   \0
0000040   \0   \0   \0   \0    d    e    f   \0
0000050   \0   \0   \0   \0   \0   \0   \0   \0
*
0000070    3    0    0   \0   \0   \0   \0   \0
0000100    g    h    i   \0   \0   \0   \0   \0
0000110   \0   \0   \0   \0   \0   \0   \0   \0
0000120   \0   \0   \0   \0    3    5    0   \0
0000130   \0   \0   \0   \0    j    k    l   \0
0000140   \0   \0   \0   \0   \0   \0   \0   \0
*
0000160
```

Ler ficheiro de acesso directo (1/3)

Parte 1/3

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

typedef struct {
    char codigo[8];
    char nome[20];
} registo_t;
```

Ler ficheiro de acesso directo (2/3)

Parte 2/3

```c
int main(void)
{
    int         fd;
    registo_t   buf;
    struct stat fstat_buf;
    int         numreg;
    int         i;

    if ((fd = open("/tmp/reg.dat", O_RDONLY)) < 0) {
        perror("open");  exit(EXIT_FAILURE);
    }

    if (fstat(fd, &fstat_buf) < 0) {
        perror("fstat");  exit(EXIT_FAILURE);
    }
```

Ler ficheiro de acesso directo (3/3)

Parte 3/3

```c
    numreg = fstat_buf.st_size / sizeof(registo_t);

    /* Listar registos por ordem inversa */

    for (i = numreg - 1; i >= 0; i--) {
        lseek(fd, i * sizeof(registo_t), SEEK_SET);
        read(fd, &buf, sizeof(registo_t));
        printf("%2d: %s, %s\n", i, buf.codigo, buf.nome);
    }

    close(fd);

    return 0;
}
```

Conteúdo

Referências (1/2)

Bibliografia

- **Advanced Programming in the Unix Environment (segunda edição)**
  Autor: W. Richard Steven e Stephen A. Rago
  Homepage: `http://www.apuebook.com/`
  - Capítulo 3 - File I/O
  - Capítulo 4 - Files and Directories
  - Capítulo 5 - Standard I/O Library
- **Linux Programming by Example - The Fundamentals**
  Autor: Arnold Robbins
  Homepage: `http://www.linux-by-example.com/`

Referências (2/2)

Bibliografia

- **The C Programming Language (segunda edição)**
  Autores: Brian W. Kerninghan e Dennis M. Ritchie
  Homepage: `http://cm.bell-labs.com/cm/cs/cbook/`
- **C - A Reference Manual (quinta edição)**
  Autores: Samuel P. Harbison and Guy L. Steele, Jr.
  Homepage: `http://careferencemanual.com/`
- **The GNU Coding Standards**
  Homepage: `http://www.gnu.org/prep/standards.html`