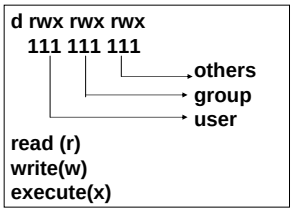
Programação em UNIX: Ficheiros e Pipes

Sistemas Operativos, 1º Semestre 2004-2005

Privilégios de Acesso aos Ficheiros

```
$ ls -l
drwxr-xr-x   2 luis   man      4096 Oct 14 16:28 FORKS
drwxr-xr-x   2 luis   man      4096 Oct 16 16:05 SIGNALS
-rw-r-----   1 luis   man      1967 Oct  8 13:09 ex10.c
-rw-r-----   1 luis   man      2210 Oct  8 13:09 ex11.c
```



File Access Permissions

```
$ chmod 777    file
$ chmod 700    *.c
$ chmod 660    file
```

```
$ chmod {ugoa}{+-}{rwx}
$ chmod o+r          file
$ chmod g+rwx        file
$ chmod  g-wx        file
```

Files in Unix

- UNIX and NT try to make every resource (except CPU and RAM) look like a file.
- Then can use a common interface:
  - open Specifies file name to be used
  - close Release file descriptor
  - read Input a block of information
  - write Output a block of information
  - lseek Position file for read/write
  - ioctl Device-specific operations

System-level functions: creat

NAME

creat - *create a new file*

SYNOPSIS

int creat(char *path, int mode);

NOTE:
creat(path, mode) is equivalent to:
open(path,O_WRONLY|O_CREAT|O_TRUNC, mode);

System-level functions: open

NAME

open - *open or create a file*

SYNOPSIS

#include <fcntl.h>

int open(char *path, int flags[, int mode])

FLAGS:

O_RDONLY	open for reading only.
O_WRONLY	open for writing only.
O_RDWR	open for reading and writing.
O_APPEND	write-append mode.
O_CREAT	creats the file, if it does not exist.
O_TRUNC	truncate the file

MODE:

Por exemplo: 0644 (equivale a: rw-r--r--)

System-level functions: read

NAME

read - *read n bytes from a file*

SYNOPSIS

int read(int fd, char *buf, int nbyte)

NOTE:
returns the number of bytes read.
On failure, returns -1.

System-level functions: write

NAME

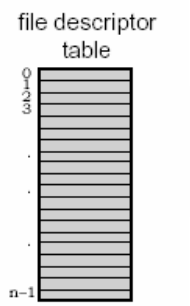
write - *write n bytes into a file*

SYNOPSIS

int write(int fd, char *buf, int nbyte)

NOTE:
returns the number of bytes written.
On failure, return -1.

stdin (0), stdout (1), stderr (2)




```
while ((n = read(0, buf, sizeof(buf))) > 0)
  if (write(1, buf, n) != n) {
        (void)write(2, note, strlen(note));
        exit(EXIT_FAILURE);
  }
```

System-level functions: close, unlink

NAME
close - close a file
SYNOPSIS
int close (int fd)

NAME
unlink - *remove directory entry*
SYNOPSIS
int unlink(char *path)

System-level functions: lseek

NAME
lseek - *move the position of the file pointer*

SYNOPSIS
#include <sys/types.h>
#include <unistd.h>
off_t lseek(int fd, off_t offset, int whence);
NOTE:
whence must be one of the following constants:
SEEK_SET
SEEK_CUR
SEEK_END

lseek Example

- **Example: printing a text file backwards:**

```
fd = open("textfile", O_RDONLY);
/* go to last char in file */
fptr = lseek(fd, (off_t)-1, SEEK_END);
while (fptr != -1) {
        read(fd, buf, 1);
        write(1, buf, 1);
        fptr = lseek(fd, (off_t)-2, SEEK_CUR);
}
```

System-level functions: tell

NAME
tell - tell the current position of the file pointer

SYNOPSIS
#include <sys/types.h>
#include <unistd.h>
long tell(int fd)

NOTE:
tell(fd) is equivalent to lseek(fd,0L, SEEK_CUR).

Library Functions

```
FILE *fopen (const char *path, const char *mode);

size_t fread( void *ptr, size_t size, size_t nmemb, FILE *stream);

size_t fwrite( const void *ptr, size_t size, size_t nmemb, FILE *stream);

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

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

int feof( FILE *stream);

int fflush(FILE *stream);

int fsync(int fd);
```

Example: Files (ex5.c)

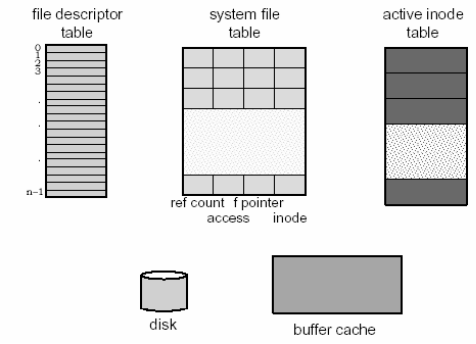
```
#include <stdio.h>
#include <fcntl.h>
#define MAXBUF 512
int main(argc,argv)
int argc;
char **argv;
{
char buffer[MAXBUF];
int  nread;

int  fd=0;     /* assume stdin */

if(argc==2){   /* read from specified file */
      fd = open(argv[1],O_RDONLY);
      }

do{
      nread = read(fd,buffer,MAXBUF);
      write(1,buffer,nread);    /* write to stdout */
}while(nread > 0);
return(0);
}
```

Example: Files (ex6.c)

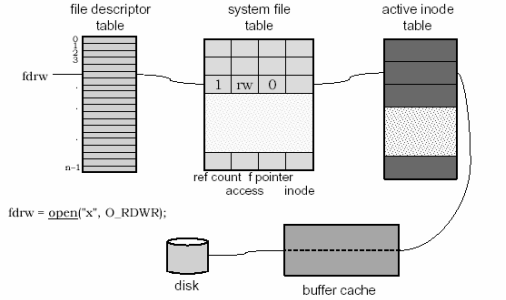
```
#include <stdio.h>
#include <fcntl.h>
#include <sys/types.h>
#include <unistd.h>
#define MAXBUF 20
int main(argc,argv)
int argc;
char **argv;
{
int  buffer[MAXBUF];
int  nread,aux,status;
int  fd,i,done,proc_id;
fd =
      open("xpto.dat",O_RDWR|O_CREAT,0600
      );
if(fd < 0){
      printf("error when opening the file ...\n");
      exit(-1);
      }
for(i=0;i<MAXBUF;i++)
      buffer[i]=i+1;
write(fd,(char*)buffer,MAXBUF*sizeof(int));
lseek(fd,(off_t)0,SEEK_SET);
done = 0;


proc_id=fork();
```

```
if(proc_id == 0){      /* proc. filho */
      while(!done){
            sleep(1);
            nread=read(fd,&aux,sizeof(int));
            if(nread>0)
                  printf("[filho]: %d\n",aux);
            else
                  done = 1;
            }
        exit(1);
      }
else{                  /* proc pai */
      while(!done){
            sleep(1);
            nread=read(fd,&aux,sizeof(int));
            if(nread>0)
                  printf("[pai]: %d\n",aux);
            else
                  done = 1;
            }
      wait(&status);
      }
return(0);
}
```

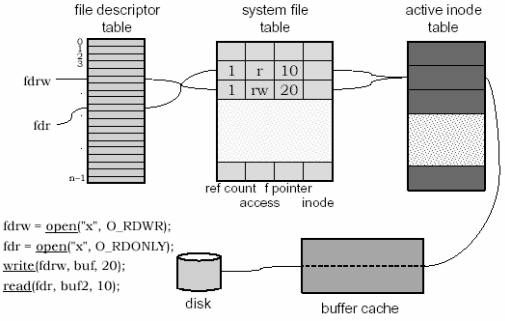
Representing an Open File (1)



Representing an Open File (2)



Representing an Open File (3)



Comunicação entre Processos usando Pipes

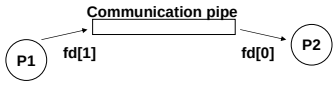
Pipes

NAME
pipe - create an interprocess communication channel

SYNOPSIS
int pipe(int fd[2])

DESCRIPTION
The pipe() system call creates an I/O mechanism called a pipe and returns two file descriptors, fd[0] and fd[1].
fd[0] is opened for reading and fd[1] is opened for writing.
A read only file descriptor fd[0] accesses the data written to fd[1] on a FIFO (first-in-first-out) basis.

RETURN VALUES
pipe() returns:
0 on success.
-1 on failure.



Exemplo: Pipes (ex7.c)

```
#include <stdio.h>
#include <fcntl.h>
char msg[]="bom dia ...";
int main(argc,argv)
int argc;
char **argv;
{
char buffer[20];
int  nread;
int  fd[2];
int y=open("xpto",O_RDWR);
pipe(fd);

printf("%d .. %d\n", fd[0],fd[1]);

if(fork() == 0){                                    /* proc. filho */
      write(fd[1],msg,sizeof(msg));
      }
else{                                               /* proc. pai */
      read(fd[0],buffer,sizeof(msg));
      printf("recebi do pipe a msg: %s\n",buffer);
      }
return(0);
}
```

Como é que podemos ter comunicação nos dois sentidos?

Usando apenas um pipe??

Exemplo: Pipes (ex7b.c)

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

char *msg[2]={"ping","pong"};

int main(int argc,char **argv){
char buffer[20];
int  nread;
int  fd1[2];
int  fd2[2];
int  i;
int  status;

pipe(fd1);
pipe(fd2);

if(fork() == 0){                    /* proc. filho */
      close(fd1[0]);
      close(fd2[1]);
      for(i=0;i<10;i++){
                  write(fd1[1],msg[0],strlen(msg[0])+1);
                  read(fd2[0],buffer,strlen(msg[1])+1);
                  printf("[filho]: %s \n",buffer);

                  }
      }
```

```
else{                                       /* proc. pai */
      close(fd1[1]);
      close(fd2[0]);
      for(i=0;i<10;i++){
                  read(fd1[0],buffer,strlen(msg[0])+1);
                  printf("[pai]: %s\n",buffer);
                  write(fd2[1],msg[1],strlen(msg[1])+1);
                  }
      wait(&status);
}
return(0);
}
```

Redireccionamento de Ficheiros: dup()

O redireccionamento é obtido usando a função **dup()**, que duplica um descriptor de ficheiro e o coloca na primeira posição livre da tabela.

- ***dup*** **returns a new file descriptor referring to the same file as its argument**

```
int dup(int fd)
```

I/O Redirection

% who > file &

```
if (fork( ) == 0) {
    char *args[ ] = {"who", 0};
    close(1);
    open("file", O_WRONLY|O_TRUNC, 0666);
    execv("who", args);
    printf("you screwed up\n");
    exit(1);
}
```

Exemplo: dup()

```
#include <stdio.h>
#include <fcntl.h>
char msg[]="Ola', processo filho !";
int main(argc,argv)
int argc;
char **argv;
{
char buffer[20];
int  nread;
int  fd[2];
int  fd_file;
int  status;
fd_file = open("temp1",O_RDWR|O_CREAT,0666);
pipe(fd);
if(fork() == 0){              /* proc. filho */
      close(1);
      dup(fd_file);                        /* 1 <- fd_file */
      close(fd[1]);
      printf("ESTOU A REDIRECCIONAR O STDOUT PARA O  FICHEIRO temp1 !\n");
      read(fd[0],buffer,sizeof(msg));
      printf("recv msg from pipe: %s\n",buffer);
      }
else{                         /* proc. pai */
      close(fd[0]);
      write(fd[1],msg,sizeof(msg));
      wait(&status);
      }
return(0);
}
```

dup2()

int dup2(int oldfd, int newfd);

- dup2 makes newfd a copy of oldfd
- If the second argument is the file descriptor of an open file, the file is first closed, then associated with the file of the first argument.

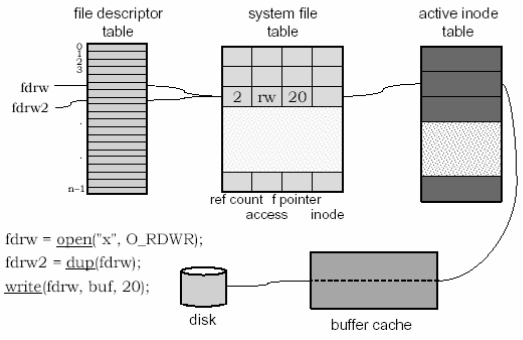
dup Example

```
/* redirect stdout and stderr to same file */
/* assumes file descriptor 0 is in use */
close(1);
open("file", O_WRONLY|O_CREAT, 0666);
close(2);
dup(1);

/* alternatively, replace last two lines with: */
dup2(1, 2);
```

who | sort

```
#include <stdio.h>
#include <fcntl.h>
int main(argc,argv)
int argc;
char **argv;
{
int  fd[2];
pipe(fd);
if(fork() == 0){                        /* proc. filho */
      close(fd[0]);                     /* close read part */
      dup2(fd[1],1);                    /* redirecciona stdout */
      close(fd[1]);
      execlp("who","who",NULL);
      }
else{                                   /* proc. pai */
      close(fd[1]);                     /* close write part */
      dup2(fd[0],0);                    /* redirecciona stdin */
      close(fd[0]);
      execlp("sort","sort",NULL);
      }
return(0);
}
```

Representing an Open File (4)



Representing an Open File (5)