Processamento de Ficheiros (usando a API do kernel)

---

1. Considere a seguinte implementação de um comando `mycat` (semelhante ao `cat` da shell Bash) utilizando directamente a API do Unix ("system calls") em vez da Biblioteca Standard do C (clib).

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/uio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#define BUFFER_SIZE 1024

int main(int argc, char* argv[]) {
  if (argc != 2) {
    printf("usage: cat filename\n");
    exit(EXIT_FAILURE);
  }
  int fd = open(argv[1], O_RDONLY);
  if (fd == -1) {
    printf("error: cannot open %s\n", argv[1]);
    exit(EXIT_FAILURE);
  }
  char buffer[BUFFER_SIZE];
  int nbytes = read(fd, buffer, BUFFER_SIZE);
  while (nbytes > 0) {
    write(STDOUT_FILENO, buffer, nbytes);
    nbytes = read(fd, buffer, BUFFER_SIZE);
  }
  close(fd);
  exit(EXIT_SUCCESS);
}
```

Leia atentamente o código e pesquise nas páginas de manual do sistema as funções que não reconhecer. Compile e execute o programa. Em seguida modifique-o para que funcione com múltiplos ficheiros de input, tal como o comando `cat` habitual.

2. Adapte o programa anterior para que imprima o conteúdo do ficheiro - os caracteres individuais - por ordem inversa. O seu programa deve funcionar da seguinte forma:

```
$ gcc -Wall backwards.c -o backwards
$ cat > test.txt
to live is the opposite of being dead
^D
$ ./backwards test.txt
daed gnieb fo etioppo eht si evil ot
```

Sugestão: use a "system call" `seek`.

3. Veja o código seguinte para um comando que recebe o nome de um ficheiro como argumento e retorna o seu tamanho em bytes usando a "system call" `stat`.

```
#include <sys/stat.h>
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char* argv[]) {
  struct stat info;
  if (argc != 2) {
    fprintf(stderr, "usage: %s file\n", argv[0]);
    exit(EXIT_FAILURE);
  }
  int retv = stat(argv[1], &info);
  if (retv == -1) {
    fprintf(stderr, "fsize: Can't stat %s\n", argv[1]);
    exit(EXIT_FAILURE);
  }
  printf("%s size: %d bytes, disk_blocks: %d\n",
         argv[1], (int)info.st_size, (int)info.st_blocks);
  exit(EXIT_SUCCESS);
}
```

Generalize o programa para que receba um número variável de argumentos e calcule o tamanho total em bytes que eles representam, bem como o número total de blocos em disco que ocupam. Altere ainda o programa para que indique para cada ficheiro a data de última alteração e o UID do dono do ficheiro. As datas em sistemas Unix são guardadas sob a forma do número de segundos que decorreram desde as 00:00 de 1 Janeiro 1970.

Sugestão: consulte a página do manual da "system call" `stat` e veja com atenção os campos da `struct stat` aí descrita.
Sugestão: para imprimir uma data num formato mais legível para um humano use por exemplo a função `ctime`.

4. Considere o seguinte código (incompleto) de um comando `mychmod`, semelhante ao `chmod` da shell Bash, usado para alterar as permissões de acesso de ficheiros.

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

int main(int argc, char* argv[]) {
  if (argc != 3 ) {
    fprintf(stderr, "usage: %s perms file\n", argv[0]);
    exit(EXIT_FAILURE);
  }

  int perms  = atoi(argv[1]);
  int operms = perms % 10;
  perms = perms / 10;
  int gperms = perms % 10;
  perms = perms / 10;
  int uperms = perms;
  mode_t newperms = (mode_t)0;

  switch (uperms) {
  case 0: break;
  case 1: /* ... */
  case 2: /* ... */
  case 3: /* ... */
  case 4: newperms |= S_IRUSR; break;
  case 5: newperms |= S_IRUSR | S_IXUSR; break;
  case 6: newperms |= S_IRUSR | S_IWUSR; break;
  case 7: /* ... */
  default:
    fprintf(stderr, "%s: illegal permission value\n", argv[0]);
    exit(EXIT_FAILURE);
  }

  switch (gperms) {
  case 0: /* ... */
```

```
  case 1: newperms |= S_IXGRP; break;
  case 2: newperms |= S_IWGRP; break;
  case 3: newperms |= S_IWGRP | S_IXGRP; break;
  case 4: /* ... */
  case 5: /* ... */
  case 6: newperms |= S_IRGRP | S_IWGRP; break;
  case 7: newperms |= S_IRGRP | S_IWGRP | S_IXGRP; break;
  default:
    fprintf(stderr, "%s: illegal permission value\n", argv[0]);
    exit(EXIT_FAILURE);
  }

  switch (operms) {
  case 0: break;
  case 1: newperms |= S_IXOTH; break;
  case 2: newperms |= S_IWOTH; break;
  case 3: /* ... */
  case 4: newperms |= S_IROTH; break;
  case 5: newperms |= S_IROTH | S_IXOTH; break;
  case 6: /* ... */
  case 7: /* ... */
  default:
    fprintf(stderr, "%s: illegal permission value\n", argv[0]);
    exit(EXIT_FAILURE);
  }

  if (chmod(argv[2], newperms) == -1) {
    fprintf(stderr, "%s: cannot chmod %s\n", argv[0], argv[2]);
    exit(EXIT_FAILURE);
  }

  exit(EXIT_SUCCESS);
}
```

Leia o código com cuidade e consulte as páginas de manual das funções com que não esteja familiarizado, e.g., `chmod`. Complete o programa, compile-o e execute-o da seguinte forma:

```
$ gcc mychmod.c -o mychmod
$ touch testfile
$ ls -l testfile
-rw-r--r-- 1 lblopes  staff  0 Feb 27 13:31 testfile
$ ./mychmod 755 testfile
$ ls -l testfile
```

```
-rwxr-xr-x 1 lblopes  staff  0 Feb 27 13:31 testfile
$ ./mychmod 799 testfile
$ mychmod: illegal permission value
```

5. Implemente um comando `mytouch` semelhante ao comando `touch` da shell Bash. O comando recebe o nome de um ficheiro como argumento. Se não existir um ficheiro com o nome dado então deve criar um novo ficheiro vazio com permissões `644` (ou `rw-r--r--`). Caso contrário, deve actualizar apenas a data da última modificação do ficheiro para a data actual. Comece pelo código seguinte que usa a chamada ao sistema `access` para verificar se o ficheiro dado como argumento existe ou não. Certifique-se que compreende o código. Não se esqueça de incluir os ficheiros cabeçalho necessários.

```
int main(int argc, char* argv[]) {
    if (access(argv[1], F_OK) == 0) {
       /* file exists - insert code to change last access date */
    } else {
        /* file does not exist - create it with given access permissions */
        mode_t perms = ...;
        int fd = open(argv[1], O_CREAT|O_WRONLY, perms);
        close(fd);
    }
    exit(EXIT_SUCCESS);
}
```

Sugestão: consulte as páginas de manual das chamadas ao sistema `access`, `open` e `chmod`. Veja também a página de manual da função `utimes` da Standard C Library.

6. O seguinte programa exemplifica como pode ser lido o conteúdo de um directório cujo nome é passado na linha de comando.

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

int main(int argc, char** argv) {
  int len;
  if (argc != 2) {
    fprintf (stderr, "usage: %s dirname\n", argv[0]);
    exit(EXIT_FAILURE);
  }
  DIR *q = opendir(argv[1]);
  if (q == NULL) {
    fprintf (stderr, "cannot open directory: %s\n", argv[1]);
```

```
    exit(EXIT_FAILURE);
  }
  printf ("%s/\n", argv[1]);
  struct dirent *p = readdir(q);
  while (p != NULL) {
    printf ("\t%s\n", p->d_name);
    p = readdir(q);
  }
  closedir(q);
  exit(EXIT_SUCCESS);
}
```

Com base neste código e no das duas alíneas anteriores, escreva um comando `myls` que funcione de forma semelhante ao comando da shell Bash `ls -l`. Note que o programa deve aceitar ficheiros comuns e directórios como argumentos. No caso do argumento ser o nome de um directório, o comando lista o conteúdo do mesmo, com uma linha para cada ficheiro ou subdirectório encontrado. Como consegue saber se o nome que é dado como argumento é de um ficheiro simples ou de um directório?
Sugestão: use a "system call"`stat` em cada entrada do directório.