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. 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 utilizador dono do ficheiro. Sugestão: veja com atenção a estrutura de dados `struct stat` utilizada pela "system call" `stat` nas páginas de manual.

3. 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. Use as "system calls" `stat`, `open`, `close` e `umask`. Leia atentamente as páginas de manual das ditas.

4. 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?