Sistemas de Operação (2018/2019)
Ficha 4

Q1. 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/stat.h>
#include <sys/types.h>
#include <sys/uio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>

#define BUFFER_SIZE 1024

int next_block_size(int count, int buffer_size) {
  return (count >= buffer_size)? buffer_size: count % buffer_size;
}

int main(int argc, char* argv[]) {
  /* check if exactly one argument is present */
  if (argc != 2) {
    printf("usage: cat filename\n");
    return EXIT_FAILURE;
  }
  /* check if file can be opened and is readable */
  int fd = open(argv[1], O_RDONLY);
  if (fd == -1) {
    printf("error: cannot open %s\n", argv[1]);
    return EXIT_FAILURE;
  }
  /* get the file size */
  struct stat info;
  int ret = lstat(argv[1], &info);
  if (ret == -1) {
    printf("error: cannot stat %s\n", argv[1]);
    return EXIT_FAILURE;
  }
  /* print the contents in blocks */
  int count = info.st_size;
```

```
  char buffer[BUFFER_SIZE];
  while (count != 0) {
    int bytesin = read(fd, buffer, next_block_size(count, BUFFER_SIZE));
    count -= bytesin;
    write(STDOUT_FILENO, buffer, bytesin);
  }
  /* close file */
  close(fd);
  return 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.

Q2. Considere a seguinte implementação de um comando `mychmod` semelhante ao comando `chmod` da shell Bash (veja a respectiva página do manual).

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

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

int main(int argc, char* argv[]) {

  if (argc != 3 ) {
    (void)fprintf(stderr, "usage: %s perms file\n", argv[0]);
    return 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:
  (void)fprintf(stderr, "%s: illegal permission value\n", argv[0]);
  /* ... */
}

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:
  /* ... */
  return 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:
  (void)fprintf(stderr, "%s: illegal permission value\n", argv[0]);
  return EXIT_FAILURE;
}

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

```
  /* ... */
}
```

Leia atentamente o código e pesquise nas páginas de manual do sistema as funções que não reconhecer, em particular `man 2 chmod`. Complete as linhas de código em falta nos pontos com comentários `/* ... */` e execute o programa com um ficheiro, e.g.:

```
$ 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
```

Q3. 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"`lstat`.

```
#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]);
    return EXIT_FAILURE;
  }

  if (lstat(argv[1], &info) == -1) {
    fprintf(stderr, "fsize: Can't stat %s\n", argv[1]);
    return EXIT_FAILURE;
  }

  printf("%s size: %d bytes, disk_blocks: %d\n",
         argv[1], (int)info.st_size, (int)info.st_blocks);
  return 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.

Q4. Implemente um comando `mytouch` semelhante ao comando `touch` da shell Bash. O comando recebe o nome de um ficheiro como argumento e:

- cria um novo ficheiro vazio com permissões `644` se não existir um ficheiro com o nome dado;
- altera a data de última alteração do ficheiro para a data actual, caso o ficheiro já exista.

Sugestão: baseie-se no exercício anterior e use ainda as "system calls" `open`, `close` e `umask`.

Q5. Usando a "system call" `getwd` escreva um programa que implemente o equivalente ao comando da shell Bash `pwd` que imprime o directório corrente. Faça `man getwd` para saber como usar a função.

Q6. Com base nos dois exemplos anteriores implemente um comando `myls` que dado um ficheiro escreva como output a mesma informação que o comando de shell Bash `ls -l`.

Generalize o programa de tal forma que aceite ficheiros comuns e directórios como argumento, mantendo um comportamento semelhante ao do `ls -l`. Note que, no segundo caso, o comando lista o conteúdo do directório, uma linha para cada ficheiro ou subdirectório encontrado. Veja o seguinte exemplo em que se exemplifica como deve consultar o conteúdo de um directório. Como sabe se o nome que é dado como argumento é um ficheiro simples ou um directório?

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

int main (int argc, char** argv) {
  int len;
  struct dirent *p;
  DIR *q;

  if (argc != 2) {
    fprintf (stderr, "usage: %s dirname\n", argv[0]);
```

```
    return EXIT_FAILURE;
  }
  q = opendir (argv[1]);
  if (q == NULL) {
    fprintf (stderr, "%s: Cannot open directory '%s'\n",
             argv[0], argv[1]);
    return EXIT_FAILURE;
  }
  printf ("%s/\n", argv[1]);
  p = readdir(q);
  while (p != NULL) {
    printf ("\t%s\n", p->d_name);
    p = readdir(q);
  }
  closedir (q);
  return EXIT_SUCCESS;
}
```