L:EIC / SO2122:
Processing Files
(using the Kernel API)

Q1. Consider the following implementation of a command `mycat` (similar to `cat` from the Bash shell) using the kernel API (system calls).

```
#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;
}
```

Read the code with care and search the manual pages for information on the functions you do not recognize. Compile and execute the program. Next, rewrite it so that it can handle multiple files as input as the usual `cat` does.

Q2. Consider the following implementation of the command `mychmod` similar to the `chmod` command from the Bash shell (check the manual page). Add the missing code in the comments `/* ... */`.

```
#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;
  }

  /* ... */
}
```

Pay attention to the code and search the manual pages for the functions you do not recognize, in particular function `chmod()`. Compile and execute the program with an input file, 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. Look at the following code for a command that gets a file name as a command line argument and returns its size in bytes using the 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 the program programa so that it can receive a variable number of command line arguments and computes the total number of bytes and the total number of disk blocks that they ocupy. Still with the same program, change it so that it prints, for each file, the date of last change and the owner of the file (hint: check the information returned in the data structure `struct stat` returned filled by `lstat` nas páginas de manual.

Q4. Implement a command `mytouch` similar to the command `touch` from the Bash shell. The command gets a file name as an argument and:

- creates a new file with permissions `644` if a file with that name does not exist in the current directory;
- changes the date of last modification to the current date, if the file already exist.

(hint: check the previous exercise and use the system calls `open`, `close` and `umask`.

Q5. Using the system call `getcwd` write a program that implements the equivalent to the Bash shell command `pwd` that prints the absolute path of the current directory. Faça `man getcwd` to understand how to use the system call.

Q6. Based on the previous two questions, implement a command `myls` that given a file writes the same output as the Bash shell command `ls -l`. Generalize the program so that it accepts common files as well as directories as command line arguments. The behavior for files and directories should be similar to that of `ls -l`. Note that, in the second case, the command lists the content of the directory, one line for each file or sub-directory found. Start by studying the following example of a program that exemplifies how to open a directory and find the information within. How do you know if an argument is the name of a file or of a directory?

```
#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;
}
```