File Processing
(using the kernel API)

---

1. Consider the following implementation of a command `mycat` (similar to `cat` in Bash) using the kernel API directly (system calls) rather than functions implemented in the Standard C Library (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);
}
```

Read the code carefully and search the manual pages for information regarding functions that you are not familiar with. Compile and execute the program. Afterwards, change the program so that it works for multiple input files, just like the Bash `cat` command.

2. The following code implements a command that receives the name of a file in the command line and returns its size in bytes using the 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 the program so that it can receive a variable number of filenames in the command line and calculate also the total size of all files in bytes as well as the total number of disk blocks they occupy. Change again the program so that it prints, for each file, the date of the last change and the UID of the owner of the file.
Suggestion: check the manual page for the system call `stat` and see the fields of the `struct stat` defined therein.
Suggestion: dates in Unix systems are kept as the number of seconds elapsed since 00:00 Jan 1st, 1970. To print a date that can be read easily by humans check function `ctime`.

3. Implement a command `mytouch`, similar to the Bash command `touch`. The command receives the name of a file in the command line. If that file does not exist, it must be created anew and empty with permissions `644` (or `rw-r--r--`). Otherwise, if it already exists, `mytouch` should adjust the date of the last modification to the current date.

Suggestion: check the manual pages for the system calls `stat`, `open`, `close`, `umask` and `utimes`. For the `open` function, check the flags `O_CREAT` and `O_EXCL`.

4. The following program shows how the contents of a directory, whose name is passed in the command line, may be read and listed.

```
#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);
}
```

Based on this code, write a command `myls` that works in just like Bash's `ls -l` for files and directories. In case the argument is the name of a directory the command should list its contents with a line for each file or subdirectory found. How can you tell whether the argument provided is a simple file or a directory?