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. Afterward, change the program so that it works for multiple input files, just like the Bash `cat` command.

2. Adapt the previous program to print the file contents - as individual characters - backwards. Your program should work as follows:

```
$ 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
```

Suggestion: use the system call `seek`.

3. 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, "%s: Can't stat %s\n", argv[0], 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. Note that dates in Unix systems are kept as the number of seconds elapsed since 00:00 Jan 1st, 1970.

Suggestion: check the manual page for the system call `stat` and see the fields of the `struct stat` defined therein.
Suggestion: to print a date that can be read easily by humans check function `ctime`.

4. Consider the following implementation of the command `mychmod`, similar to `chmod` from Bash, used to change the access permissions of a file.

```
#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);
}
```

Read the code carefully and consult the manual pages for the functions you are unfamiliar with, e.g., `chmod`. Complete the missing code and compile and run the program as follows:

```
$ 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. 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. Start with the following code that uses the system call `access` to check whether the file exists or not. Make sure you understand how it works. Don't forget to add the required header files.

```
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);
}
```

Suggestion: check the manual pages for the system calls `access`, `open` and `chmod`. See also function `utimes` from the Standard C Library.

6. 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?
Suggestion: use the `stat` system call on every entry in the directory.