Operating Systems

Homework Assignment #6

Due 2.7.2015, 23:59

Part 1

In this part of the assignment you will implement a HTTP client.

Write a program `http_client.o`. The program will:

- Receive 3 arguments
  - Host name (e.g., "www.cs.tau.ac.il")
  - Path on the host – URL (e.g., "/~moshesulamy/os.htm").
  - Port number – **optional** argument. Use default HTTP port if not provided.
- Open a TCP connection to the host
  - Needs formatted network address (`sin_addr` field).
  - Read about `gethostbyname` function (`man 3 gethostbyname`) to retrieve it.
- Send a minimal HTTP 1.1 GET request to the server.
  - According to the program's parameters.
  - No need for special headers, except to prevent keep-alive.
- Parse the response.
  - Verify you got a valid response (i.e., HTTP 200), otherwise output error message & code.
  - For any response (including invalid responses), output the body (**only!**) to the standard output.

Guidelines

- Use only the system calls learned in class. Follow instructions **exactly!**
- Do not forget to check the return values of all functions. Comment out auxiliary code!
- Do not slave over parsing strings, assume all data is of valid format (address, HTTP request/response, etc.)
- **Submit:** `http_client.c`.

Part 2

In this part of the assignment you will implement a simple character device module.

- Modify the code we used in the recitation (`chardev.c`)
  - Initialize the Message buffer so it contains a string with your full name.
  - Modify the read code so it returns characters from the buffer
    - Every read continues where the previous read ended
    - If reading more than left in the buffer, read what is available, then reset the count - so the next read will resume reading from the start
  - Modify the write code so it updates the buffer with whatever data the user writes
    - The data is written to the start of the buffer
    - The data should not exceed 100 bytes – if it does, write until 79th byte and ignore rest
    - Make sure to properly terminate the string (i.e., 80-bytes buffer, place '\0' at the end)
    - Any future reads (after each write) should start from the beginning of the buffer, i.e., the write resets the read count
    - Write should fail (with no effect on the buffer!) if the user-provided write buffer is corrupt somehow (use a temp buffer for that)
  - Test your code by creating and accessing one character device
- Next test your code by creating two character devices of the relevant type. Try reading from both devices one after the other using "`head`" utility (`man 1 head`).
  - What are you seeing? Why?
- In `chardev.c` there is a simple mechanism to prevent concurrent access to the same device
  - What else does it prevent?
  - Write a (short) program `test.c` which demonstrates this (no need for error checking)

Useful Utilities

- read 1 byte from file (device files too): `head -c 1 /dev/simple_char_dev`
- read last 15 lines from kernel log: `dmesg | tail -15`

Guidelines

- Clearly document your code (also use comments to separate workers, main, and queue implementations).
- Use only the system calls learned in class. Follow instructions **exactly!**
- Use **strtol** (man 3 **strtol**) to convert a string to integer. You may assume input is valid and in correct range.
- Do not forget to check the return values of all functions. Comment out auxiliary code!
- Do not slave over parsing strings, assume all data is valid (address, HTTP request, etc.)
- **Submit:** `test.c`, modified `chardev.c` **including relevant Makefile!**