1 SYSTEMS PROGRAMMING LABORATORY IX - Linux System Calls & Inline Assembly Code

Examples&Exercises:

- Complete the following codes if necessary, then compile and run the code.
- Analyze the code and output.

1. access: Testing File Permissions; check-access.c

   - first execute with a non-existing file,
   - then create a file and change permission bits to observe the behavior of the program,
   - see the system calls by

     ```
     $ strace check-access
     ```

     study the output in detail.

2. fcntl: Locks and Other File Operations; lock-file.c

   - execute without supplying a filename,
   - execute in two different windows as

     ```
     $ ./lock-file supplyafile
     ```

3. fsync: Flushing Disk Buffers ; write_journal_entry.c

   - Complete the code,
   - say you have a endless loop to produce the entries in the main function,
   - study the cases with and without *fsync*,
   - observe the size changes in the journal in another window,
   - can you estimate the buffer size for the without *fsync* case?

4. getrlimit and setrlimit: Resource Limits; limit-cpu.c

   - see what are other possible resource limits by

```
$ man getrlimit
$ man setrlimit
```

- modify the code to print out these resource limits supplied as defaults,
- interpret the output.

5. getrusage: Process Statistics; print-cpu-times.c
    - Complete the code,
    - see what are other possible process statistics by (see struct rusage)

      ```
      $ man getrusage
      ```

    - modify the code to print out these process statistics supplied as defaults,
    - interpret the output.

6. mprotect: Setting Memory Permissions; mprotect.c
    - it is given for *PROT_NONE* for no memory access,
    - try the other memory protection flags *PROT_READ*, *PROT_WRITE*, and *PROT_EXEC* for read, write, and execute permission, respectively.

7. readlink: Reading Symbolic Links; print-symlink.c
    - study the cases;
        - without a file,
        - with an ordinary file (not a symbolic file),
        - create a link by

          ```
          $ ln -s arealfile supplyaname
          $./print-symlink supplyaname
          ```

8. sysinfo: Obtaining System Statistics; sysinfo.c
    - see what are other possible system statistics by (see struct sysinfo)

      ```
      $ man sysinfo
      ```

    - modify the code to print out these system statistics,
    - interpret the output.

9. Inline Assembly Code (Example); bit-pos-asm.c, bit-pos-loop.c

- compile and execute as the followings

  ```
  $ gcc -O2 -o bit-pos-loop bit-pos-loop.c
  $ gcc -O2 -o bit-pos-asm bit-pos-asm.c
  $ time ./bit-pos-loop 250000000
  $ time ./bit-pos-asm 250000000
  ```

- why the optimization level 2 is used?
- try the other levels and observe the execution *time*s,
- analyze the results; which optimization level should be used and why?

10. TO BE GRADED; modify the code lock-file.c such that

    - we have two processes (either two threads or a forked child),
    - these two processes have an access to same file to read and write,
    - your program should investigate the following cases;
      - one is locked the file by *fcntl* and the other tries to read and write,
      - one is locked the file by *fcntl* and the other also tries to lock the file by *fcntl* then attempts to read and write.
    - what are the possible outcomes and issues?

11. TO BE GRADED; modify the completed code; print-cpu-times.c such that

    - we have a parent and one child,
    - get process statistics for both the parent and the child,
    - interpret the output.