EGC 301P Operating Systems Lab

List of Operating Systems Lab Exercises

I File Management

1. **Creating Different File Types:** Utilize both shell commands and system calls to create the following types of files:
   a. soft link (using the symlink system call)
   b. hard link (using the link system call)
   c. FIFO (using either the mkfifo Library Function or the mknod system call)
2. **Background Process Exploration:** Develop a simple program to execute indefinitely in the background. Traverse the /proc directory and extract relevant process information from the corresponding proc directories.
3. **File Creation and Descriptor Printing:** Write a program that creates a file and prints its file descriptor value using the creat() system call.
4. **Opening Existing File:** Write a program to open an existing file in read-write mode. Experiment with the O_EXCL flag.
5. **Continuous File Creation:** Create a program that generates five new files in an infinite loop. Execute the program in the background and inspect the file descriptor table at /proc/pid/fd.
6. **Input and Output Using System Calls:** Craft a program to take input from STDIN and display it on STDOUT using only read/write system calls.
7. **File Copying:** Develop a program to copy the contents of file1 into file2, emulating the behavior of the **$cp file1 file2** command.

8. **Read-Only File Reading:** Write a program to open a file in read-only mode, read line by line, and display each line as it is read. Close the file upon reaching the end of the file.
9. **File Information Extraction:** Create a program to print various details about a given file, including:
   a. Inode
   b. Number of hard links
   c. UID
   d. GID
   e. Size
   f. Block size
   g. Number of blocks
   h. Time of last access
   i. Time of last modification
   j. Time of last change
10. **File Write and Seek:** Implement a program to open a file in read-write mode, write 10 bytes, move the file pointer by 10 bytes using lseek, and then write another 10 bytes.
    a. Check the return value of lseek.
    b. Open the file with od command and examine the empty spaces between the data.
11. **File Descriptor Duplication and Appending:** Write a program to open a file, duplicate the file descriptor, append the file with both descriptors, and verify whether the file is updated correctly.
    a. Use **dup**
    b. Use **dup2**
    c. Use **fcntl**

12. **Determining Opening Mode of a File:** Create a program to find out the opening mode of a file using the **fcntl** system call.
13. **Waiting for STDIN Using Select:** Develop a program to wait for input from STDIN for 10 seconds using **select**. Include proper print statements to verify data availability within the specified time.
14. **File Type Identification:** Write a program that takes input from the command line and identifies the type of file. Ensure the program can recognize various file types.
15. **Displaying User Environmental Variables:** Create a program to display the environmental variables of the user, utilizing the **environ** variable.
16. **Mandatory Locking Implementation:** Write a program to perform mandatory locking with the following implementations:
    a. Implement a write lock.
    b. Implement a read lock.
17. **Online Ticket Reservation Simulation:** Develop a program to simulate online ticket reservation with the implementation of a write lock. Write one program to open a file, store a ticket number, and exit. Write a separate program to open the file, implement a write lock, read the ticket number, increment the number, print the new ticket number, and then close the file.
18. **Record Locking Implementation:** Write a program to perform record locking with the following implementations:
    a. Implement a write lock.
    b. Implement a read lock.

Create three records in a file. Whenever you access a particular record, first lock it, then modify/access it to avoid race conditions.

II Process Management

19. **Process States:** Develop a program to initiate a process in distinct states:
    a. running
    b. sleeping
    c. stopped

    Confirm the current state of the process using the relevant commands.

20. **Printing Parent and Child Process IDs:** Write a program that calls fork and prints both the parent and child process IDs.
21. **File Writing by Parent and Child Processes:** Develop a program that opens a file, calls fork, and allows both the child and parent processes to write to the file. Examine the output of the file.
22. **Creating a Zombie State:** Write a program to create a zombie state in the running program.
23. **Creating an Orphan Process:** Develop a program to create an orphan process.
24. **Creating and Waiting for Child Processes:** Write a program to create three child processes. The parent process should wait for a specific child process using the waitpid system call.
25. Executing an Executable Program:
    a. Execute a program using the exec system call.
    b. Pass input to an executable program (e.g., execute an executable as **$./a.out name**).
26. **Executing ls -Rl Using Various exec System Calls:** Write a program to execute **ls -Rl** using the following system calls:
    a. execl
    b. execlp

c. execle

d. execv

e. execvp

27. **Getting Maximum and Minimum Real-Time Priority:** Develop a program to retrieve the maximum and minimum real-time priority.
28. **Determining and Modifying Program Priority:** Find out the priority of your running program and modify it using the nice command.
29. **Getting and Modifying Scheduling Policy:** Write a program to obtain the scheduling policy and modify it (e.g., SCHED_FIFO, SCHED_RR).
30. **Running a Script at a Specific Time with a Daemon Process:** Create a program to execute a task at a specific time using a Daemon process.

III System V IPC Mechanisms

31. **Pipe Creation and Communication:** Write a simple program to create a pipe, write to the pipe, read from the pipe, and display the content on the monitor.
32. **Data Transmission from Parent to Child:** Develop a simple program to send data from the parent process to the child process.
33. **Two-Way Communication:** Write a program to send and receive data between the parent and child processes using two-way communication.
34. **Executing ls -l | wc:** Write a program to execute **ls -l | wc** using:
    a. dup
    b. dup2
    c. fcntl

35. **Counting Directories with dup2:** Write a program to find the total number of directories in the present working directory using **ls -l | grep ^d | wc**. Implement the solution using only **dup2**.
36. **FIFO File Creation:** Create a FIFO file using:
    a. **mknod** command
    b. **mkfifo** command
    c. Use **strace** command to determine which command (**mknod** or **mkfifo**) is more efficient.
    d. **mknod** system call
    e. **mkfifo** library function
37. **FIFO Communication (One-Way):** Write two programs to enable communication through FIFO using one-way communication.
38. **FIFO Communication (Two-Way):** Write two programs to establish communication through FIFO using two-way communication.
39. **Waiting for Data in FIFO:** Write a program to wait for data to be written into a FIFO within 10 seconds, utilizing the **select** system call with the FIFO.
40. **Process File Limit and Pipe Size:** Write a program to print the maximum number of files that can be opened within a process and the size of a pipe (circular buffer).
41. **Message Queue Creation:** Write a program to create a message queue, and print the key and message queue ID.
42. **Message Queue Information:** Write a program to print information about a message queue using **msqid_ds** and **ipc_perm** structures:
    a. Access permission
    b. UID, GID
    c. Time of last message sent and received
    d. Time of last change in the message queue

e. Size of the queue
f. Number of messages in the queue
g. Maximum number of bytes allowed
h. PID of the **msgsnd** and **msgrcv**

43. **Sending Messages to Message Queue:** Write a program to send messages to the message queue. Check using **$ipcs -q**.
44. **Receiving Messages from Message Queue:** Write a program to receive messages from the message queue:
    a. With 0 as a flag
    b. With **IPC_NOWAIT** as a flag
45. **Changing Message Queue Permissions:** Write a program to change the existing message queue permissions using the **msqid_ds** structure.
46. **Removing Message Queue:** Write a program to remove the message queue.
47. **Shared Memory Operations:** Write a program to create shared memory and perform the following operations:
    a. Write some data to the shared memory
    b. Attach with **O_RDONLY** and check whether you are able to overwrite
    c. Detach the shared memory
    d. Remove the shared memory
48. **Semaphore Creation and Initialization:** Write a program to create a semaphore and initialize its value:
    a. Create a binary semaphore
    b. Create a counting semaphore
49. **Semaphore Implementation:** Write a program to implement a semaphore to protect any critical section:
    a. Rewrite the ticket number creation program using a semaphore

b. Protect shared memory from concurrent write access
c. Protect multiple pseudo resources (maybe two) using a counting semaphore
d. Remove the created semaphore

50. **Deadlock:** Develop a program that intentionally induces a deadlock scenario using semaphores.
51. **Inter-Machine Communication using Socket:** Write a program to communicate between two machines using sockets.
52. **Concurrent Server Creation:** Write a program to create a concurrent server:
    a. Use **fork**
    b. Use **pthread_create**

IV Timers, Resource Limits, Multithreading and Signals

53. **Interval Timer Programming:** Write separate programs for each time domain to set an interval timer for 10 seconds and 10 microseconds.
    a. Use ITIMER_REAL
    b. Use ITIMER_VIRTUAL
    c. Use ITIMER_PROF
54. **System Resource Limits:** Write a program to print system resource limits using the getrlimit system call.
55. **Setting System Resource Limit:** Write a program to set a system resource limit using the setrlimit system call.
56. **Execution Time Measurement:** Write a program to measure the time taken to execute 100 **getppid()** system calls using a time stamp counter.
57. **System Limitation Exploration:** Write a program to print system limitations for:
    a. Maximum length of arguments in the exec family of functions.
    b. Maximum number of simultaneous processes per user ID.

c. Number of clock ticks (jiffies) per second.
d. Maximum number of open files.
e. Size of a page.
f. Total number of pages in physical memory.
g. Number of currently available pages in physical memory.

58. **Multithreading Exploration:** Write a simple program to create three threads and print the IDs of the created threads.
59. **Signal Handling:** Write separate programs using the signal system call to catch the following signals:
    a. SIGSEGV
    b. SIGINT
    c. SIGFPE
    d. SIGALRM (using the alarm system call)
    e. SIGALRM (using the setitimer system call)
    f. SIGVTALRM (using the setitimer system call)
    g. SIGPROF (using the setitimer system call)
60. **Ignoring and Resetting Signals:** Write a program to ignore a SIGINT signal and then reset it to the default action using the signal system call.
61. **Signal Handling with sigaction:** Write separate programs using the sigaction system call to catch the following signals:
    a. SIGSEGV
    b. SIGINT
    c. SIGFPE
62. **Signal Handling with sigaction:** Write a program to ignore a SIGINT signal and then reset it to the default action using the sigaction system call.

63. **Creating an Orphan Process:** Write a program to create an orphan process. Utilize the kill system call to send a SIGKILL signal from the child process to the parent process.
64. **Signal SIGSTOP Handling:** Create two programs: the first program awaits the SIGSTOP signal, while the second program sends the signal using the kill system call. Determine whether the first program successfully catches the signal or not.

****