InterBit
Consulting & Training ltd.
ייעוץ, הדרכה, הטמעה, ביצוע פרויקטים

Unix / Linux Programming Exercises

Basic I/O exercise.

1. In directory /proc resides directories that their names are process Ids. The ownership of each directory is the owner of the process. In each directory there is a sub-directory named “fd” that represents the file descriptor table. The content of /proc/xxx/fd are symbolic link files the process has now open.
   1. Write a program that scans /proc and finds processes that belong to you.
   2. Check which files each of your processes uses.
   3. add a switch –r that will only display open files. Verify by running tail –f /var/log/messages

Signals exercise.

Write an application that uses temporary files in /tmp and simulates database operations.

1 Stamp your temporary files with your PID extension.

2 Block critical sections of database update from signals.

3 Create a signal handler that will perform cleanup on exit.

    1 Finish database update and close.

    2 Remove temporary files.

4. ignore ctrl-c input, only exit when kill is sent and then exit nicely.

Processes exercise

1. Create a watchdog daemon that will start two other daemons that will fake a service.
   1. Each child daemon will fake a service.
   2. Each child process will send life sigh signal (use SIGUSR1/2).
   3. The watchdog service will make sure it's children are running by expecting a life sign on a time frame window.
   4. Once any of the child processes dies or not responding, the watchdog process will restart it.

Pipes exercise

1. Improve the processes exercise but now one child process will send a message to the other one regarding the “service”.

Message Queue exercise

1. Implement a “Mirs” network. Each “subscriber” will have an ID which will be equivalent to a queue ID. Each subscriber will read from the queue of his own ID and send messages to other subscribers queue ID.
2. Error messages will be sent to an “Operator” on a designated queue like 166.
3. Operator will be able to read from all queues and empty them.
4. Try to identify resource full situations automatically and perform step 3 when they happened.

Semaphores exercise

1. Implement a simulation to the philosophers riddle having 5 philosophers around a table.
2. To initialize the simulation, write a “waiter” program that “sets up” the table.

Shared Memory exercise

1. Implement IRC network using shared memory. Every “subscriber” should have a piece of the shared memory assigned to him.
2. Every shared memory piece assigned to a subscriber should have some space reserved to pointers (3 could be enough).
3. Make sure that reader will not read information while modified. (not necessarily by using semaphores or locks)

Socket exercise.

1. In BroadcastClient.c add the following features.
    1. Display the server that replied.

2. improve your IRC software by using sockets to connect.

Threads exercise

1. In the example program simplethread.c there is a sleep(1) function call just before main() exits.
    1. What will happen if sleep() is removed?
    2. Can you replace the sleep() function to anything better?

2. Improve your IRC software to use threads.

Advanced I/O exercise.

1. Write a service that simulates printing. The service will receive print requests. The user will execute a client application to submit print request “mylpr”. Each client activation will write to a common “Index” file residing in a common directory it's print request.
    1. Use fixed width record containing a job ID – numeric and a file name null terminated string up to MAXPATHLEN.
    2. Read the index file to find the last print request numeric id. Make sure no one can touch the file at that moment.
    3. Write the new request to the file having an id greater than the last one found.
2. Write the server side of that print simulation. Focus only on the index file. Do not implement actual printing.
    1. Scan the index file and find a job to execute.
    2. Mark the job selected as “Printed” a simple method can be simply by setting the job ID to 0.
3. BroadcastClient.c, In the current implementation, after the last server replies, the client is blocked on the accept() system call. Improve the client so it will be able to send more requests.

4. further improve your IRC software to incorporate everything you’ve learned in the course.