Learning Report
Linux OS and programming

GLOBAL ENGINEERING ACADEMY

Genesis

L&T Technology Services

Details

Name	PS Number	Email ID
ASHA N	99002646	asha.n@ltts.com

Contents

Learning Objectives of the Module

- Knowledge on features of shell and commands to handle files and directories in Linux
- Apply the knowledge of GNU tools to build, debug and analyze C/C++ programs
- Apply and analyze concepts of Process management and IPC to develop system programs
- Apply the knowledge of commands to analyze, signal the running processes from CLI.

Activity 1 – Individual and Collaborative Learning

- Understanding each component in Learning Outcomes
- Share Good resources that would be beneficial for all.
- Posting questions in the communication channels.
- Help peers with their queries.
- Reviewing the assignment of peers.

https://lnttsgroup.sharepoint.com/:f:/s/GEA/Global%20Engineering%20Academy/GEA%20Insights/Genesis/EmLStNH5ff1HsO5mpMHwlPEBNCWyrPP1nVzIXfzzR67yMA?e=TVJaQE

Activity 2 – Linux Commands

- Several resources were provided for the pre-read (flipped classroom).

  Linux Commands for Beginners

- Various Linux commands execution was demonstrated during the session along with hands-on. Few of the basic commands are:

1.date	5. Uname	9. Wc	13.ls
2.cal	6. clear	10.cat	
3. echo	7. seq	11.tac	
4. bc	8.rev	12.date	

- Commands related to file locations, file type, file permissions and disk management. Few of them are:

1.locate	5.chgrp	9.unmount
2.type	6.df	
3.chamod	7.du	
4.chown	8. Mount	

- Filter commands, pattern matching commands were also included in the leaning module. Few of the commands included are:
  1.grep 2.awk 3.sed
  0 - #stdin 1 - #stdout 2 - #stderr

- Standard input output files and their redirection to other log files

- Several examples along with practice questions were provided as assignments.

- The assignments were submitted in the SharePoint platform whose links are provided below.

Task Performed is available at:
https://lnttsgroup.sharepoint.com/:f:/s/GEA/Global%20Engineering%20Academy/GEA%20Insights/Genesis/EuusR8qpKMFEpaJWgVLPXigBHaHq16_-otI8nheX7X5wCA?e=dfdRfT

Activity 3 – Development tools

Understanding the Tool chain:
Set of Software development tools, linked (or chained) together by specific stages
• Preprocessor, Compiler, Assembler, Linker
• Debugger, Symbol Table checker, Object Core dump, header analysis, Size analysis
Native Tool chain
• Translates Program for same Hardware
• It is used to make programs for same hardware& OS it is installed on
• It is dependent on Hardware & OS
• It can generate executable file like exe or elf
Cross Tool chain
• Translates Program for different hardware
• It is used to make programs for other hardware like AVR/ARM
• It is Independent of Hardware & OS

The development tools include all the GNU tools, multiple file programming libraries and Makefiles.

1. GNU Compiler Collection includes compilers for C, C++, Objective C, Ada, FORTRAN, Go and java.

   gcc : C compiler in GCC
   g++ : C++ compiler in GCC

   To check the gcc Version
   − gcc -v
   − man gcc # More info about gcc

   C Program Build Process
   1) Pre-processor : gcc -E filename.c
      cpp hello.c -o hello.i
   2) Compilation: gcc -S filename.c
      gcc -S hello.i
   3) Assembler: gcc -c filename.c
      as -o hello.o hello.s
   4) Linker: gcc filename.c
      ld -o hello.out hello.o ...libraries...
   5) Build executable: gcc file.c # Creates a.out as executable file
      gcc file.c -o output

2. Libraries
   Collection of pre-compiled object files that can be linked into your programs via the linker.
   Example: system functions such as printf () and sqrt ().

   1) Static Library:
      • The machine code of external functions used in program is copied into the executable.
      • Has file extension of. a (archive file) in Unixes or .lib (library) in Windows.
      • A static library can be created via the archive program ar.Pros of Static Library linking
      • No need to load additional files before running the executable.

- Runtime is faster. Cons of Static Library linking
- Larger file size of executable
- Updating library code requires rebuild of whole code.

ex: Static Library:

```
gcc sum.c -c
gcc sqr.c -c
ar rc libsimple.a sum.o sqr.o
gcc test.c -c
gcc -L. test.o -o s1.out -lsimple
gcc -L. test.o -o s2.out -lsimple -static
```

2) Dynamic/Shared Library:

- File extension is .so (shared objects) in Unixes or .dll(dynamic link library) in Windows.
- Only a small table is created in the executable.
- When the function is called, on demand OS loadsmachine code of external functions
- A process known as dynamic linking.
- Executable file is smaller and saves disk space
- Most operatingsystems allows one copy of a shared library in memory to be used by all running programs
- Shared library code can be upgraded without the need to recompile your program.

Because of the advantage of dynamic linking, GCC linksto the shared library by default if it is available.

ex: Dynamic Library:

```
gcc sum.c -c
gcc sqr.c -c
gcc -shared -o libsample.so sum.o sqr.o
gcc test.c -c
gcc -L. test.o -o d1.out -lsample
LD_LIBRARY_PATH=. ./d1.out
```

3. Makefile

```
syntax : target :dependencies
        <tab> command
        all:all.out
    create makefile -> go to terminal ->enter make
    $@:Target, $<:first dependency ,$^:all dependencies.
    all.out : test.c sum.c sqr.c
    gcc test.c sum.c sqr.c -o all.out
    run:all.out
    ./all.out
    clean:
    rm all.out
```

Exercise Performed: Makefile creation for a c program using two directories include and source.

4. Static and Dynamic Code analysis
    Static Analysis Tool: cppCheck
    Usage: cppcheck path_to_src
    Dynamic Code Analysis tool: Valgrind
    Usage: valgrind ./a.out

Task Performed is available at:
https://lnttsgroup.sharepoint.com/:f:/s/GEA/Global%20Engineering%20Academy/GEA%20Insights/Genesis/Et6U4RMjJF1ArR0QdAMURNQBlhuOetz3190UUqiSWshYtg?e=AjxDIW

Activity 4 – Shell Script

- Micro project

  Brief description about the Micro Project:
  Developed a Micro project using Shell Scripting to automate the process of cloning multiple git repositories, executing the Makefile, Cppcheck and Valgrind.

  Step1: Created a shell script file (.sh) which includes following operations:
  Step 2: From the input file(.csv) extracted the username, usermailid and git repository link stored in variables and also username, usermaild are written to the output file(.csv)
  Step 3: Cloned the git repository to local machine and the git clone status is noted down to the output .csv file (3rd column).
  Step 4: Executed Makefile present in the cloned repository and its status of Build or Not build noted down to output .csv file (4th column)
  Step 5: Evaluated Cppcheck to all the files in repositories and the number of error status is noted down to the output .csv file (5th column)
  Step 6: Also evaluated the valgrind to check the memory leakage or Heap status for the executable file and the heap status is noted down to the output .csv file (6th column)
  Step 7: The above Step-2 to Step-6 is repeated for all the repositories link present in the Input file (.csv)

Task Performed is available at:
https://lnttsgroup.sharepoint.com/:f:/s/GEA/Global%20Engineering%20Academy/GEA%20Insights/Genesis/EnDHF-MTaipIrMmp7N1RZzwBDAFjvl5N6qAM_A5bAXEsig?e=AhxUbY

Activity 5 – Process, Threads and IPC

- Concepts about the process, threads and Inter process communication were also thought both in live sessions and flipped mode.

- Concepts of Thread were explained which has the following advantages
- Concurrent execution and faster response, less time for context switch
- Effective use of multiprocessor system.
- Explanation about the operating system including the scheduling algorithms pro's and cons of each and efficient process management were discussed.
- Resource sharing: code, global data, files can be shared among threads • PC, Stack and Registers is separate for each thread

- **Types of threads**

  • User threads
  • Threads used by application programmers, are above kernel and without kernel support
  • Kernel threads
  • Supported within kernel, perform multiple simultaneous tasks to serve multiple kernel system calls

- **Thread Models**

  • Used to map user threads to kernel threads
  • Many to One model : Many user-level threads are mapped to single kernel thread, thread management is handled by thread library in user space
  • One to One model : Separate kernel thread is created to handle each and every thread, limitation is the count of threads that can be created
  • Many to Many: Many user-level threads are mapped to multiple kernel level threads
  Restricted Circulation
  |Commands
  • ps –e –L –o pid,ppid,lwp,nlwp,stat,cmd
  • ps –eLf
  • To create threads, POSIX thread library is used
  • pthread_create
  • pthread_join
  • pthread_self
  • pthread_equal
  • pthread_yield
  • pthread_cancel
  • gcc psample.c –lpthread

Task Performed is available at:

https://lnttsgroup.sharepoint.com/:f:/s/GEA/Global%20Engineering%20Academy/GEA%20Insights/Genesis/ElErSgR2HANGvZzgOTqfXQMB5DzS3EjYWA6f1gsCMPB7HA?e=Dgzd8s

- **IPC**

  • Data exchange
  • Synchronization
  • Dependency / Sequencing
  • Mutual Exclusion shared memory, message queues, FIFOs/pipes◊
  • Data exchange semaphore, mutex, spinlocks◊
  • Mutual exclusion semaphores, condition variables / event flags◊
  • Dependency Process that writes/updates data is PRODUCER and process that reads is CONSUMER

- **MUTEX**

  • Mutual Exclusion
  • Only locked Process(es)/Threads can unlock the resources
  • Any other Process/Threads trying to unlock is referred as "unauthorized operation"
  • Unlocking twice or unlocking before locking is not allowed
  • Strictly lock & unlock in the same thread only
  • Mutex will have "ownership" as compared to semaphore

- **MUTEX API'S**

  • #include <pthread.h>
  • pthread_mutex_t m1=PTHREAD_MUTEX_INITIALIZER (declare & initialize)
  • pthread_mutex_init(&m1)
  • pthread_mutex_lock(&m1) (lock)
  • pthread_mutex_unlock(&m1) (unlock)

- **SEMAPHORE**

  • Sequencing, signaling mechanism, used for process/thread synchronization
  • Manage and protect access to shared resources
  • Kernel level data structure Types of usage
  • Binary Semaphore
  • Value of semaphore ranges between 0 & 1
  • Mutual Exclusion / Access to a single resource
  • Counting Semaphore
  • Value of semaphore can be 0 (zero) & any positive value
  • Accessing/sharing multiple similar resources Two (2) varieties of semaphores
  • Traditional System V semaphores
  • POSIX semaphores.

Two (2) types of POSIX semaphores
• Named
• Unnamed

Task Performed is available at:

https://lnttsgroup.sharepoint.com/:f:/s/GEA/Global%20Engineering%20Academy/GEA%20Insights/Genesis/EmIp69YJgnBMkYrUOhDZyScBHPQpxsIGtOin5bysUy07SQ?e=zaln90

- **FIFO/PIPES**

• Pipe is one-way communication only
• If a process tries to read before something is written to the pipe, the process is suspended until something is written.
• For two-way communication using pipes, two pipes should be used.
• Process-1 writes to Pipe-1 & reads from Pipe-2
• Process-2 reads from Pipe-1 & writes to Pipe-2 Named Pipe/ FIFO
• Connection between two unrelated processes int mkfifo(const char *pathname, mode_t mode)
• mkfifo mypipe, tail -f mypipe.

- **SHARED MEMORY**

• Memory Segment is created by the kernel and mapped to the data segment of the address space of a requesting process Can be used like a global variable in address space.
• int shm_open (const char *name, int oflag, mode_t mode); Create, or gain access to, a shared memory object.
• void *mmap (void *addr, size_t length, int prot, int flags, int fd, off_t offset); Map a shared memory object into its address space. Do operations on shared memory (read, write, update).
• int munmap (void *addr, size_t length); Delete mappings of the shared memory object.
• int shm_unlink (const char *name); Destroy a shared memory object when no references to it remain open.

- **Pre Read for the above mentioned topics**

- https://www.youtube.com/watch?v=exlaEOVRWQM
- https://www.youtube.com/watch?v=_NlmflJQDI4
- https://www.youtube.com/watch?v=aKjDqOguxjA
- https://www.youtube.com/watch?v=vF3KKMI3_1s
- https://www.youtube.com/watch?v=scfDOof9pww

- **Study materials and resources provided by faculty**
.

- https://www.softprayog.in/programming/program-process-threads
- https://www.softprayog.in/tutorials/ps-command-usage-examples-in-linux
- https://www.softprayog.in/programming/creating-processes-with-fork-and-exec-in-linux
- https://www.softprayog.in/programming/signals-in-linux
- https://www.softprayog.in/programming/posix-threads-programming-in-c
- https://www.softprayog.in/programming/posix-threads-synchronization-in-c
- https://www.softprayog.in/programming/semaphore-basics
- https://www.softprayog.in/programming/posix-semaphores
- https://www.softprayog.in/programming/system-v-semaphores
- https://www.softprayog.in/programming/interprocess-communication-using-posix-message-queues-in-linux
- https://www.softprayog.in/programming/interprocess-communication-using-posix-shared-memory-in-linux
- https://www.softprayog.in/programming/interprocess-communication-using-system-v-message-queues-in-linux
- https://www.softprayog.in/programming/interprocess-communication-using-system-v-shared-memory-in-linux
- https://www.softprayog.in/programming/interprocess-communication-using-fifos-in-linux
- https://www.softprayog.in/programming/interprocess-communication-using-pipes-in-linux

Activity 6 – Mini project

- Topic Checklist - Details
- Implementation Guidelines - Details
- Mini-Project Title – Matrix multiplication
-

The above mentioned mini project includes the concepts of threads, file handling and multithread. T project takes input from file in the system for the matrix multiplication and the threads are assignesd assigned with the respective activities parallely.

Mini project on linux : Matrix multiplication

https://lnttsgroup.sharepoint.com/:f:/s/GEA/Global%20Engineering%20Academy/GEA%20Insights/Genesis/EuAPKu-1gK5JoUFicn0cj0QBG5ySlSE22B-N853xcjFDdA?e=65saN4