C/C++ Programming under Linux - Part 1

A GNU Linux Program which will help you to understand the Compilation and the Linking Process.

```
//////////////////////
//first.cpp
// A hello world program using GNU C/C++ under Linux
// g++ first.cpp => a.out
// if you want to give another name (say test.exe ) for
// the executable

// g++ -otest.exe first.cpp
// ./test.exe
// test.exe will give you error , to avoid the error
// reset the enviornment variable
//  export PATH=.:$PATH
// test.exe ( will print hello world on to the console )
// Console output
// -------------------
//home/praseed/gnu_linux$ g++ -otest.exe first.cpp
//home/praseed/gnu_linux$ test.exe
//Hello world
#include <stdio.h>
//---- user entry point
int main( int argc , char **argv ){
            printf("Hello world \n" );
}
```

C/C++ Programming under Linux - Part 2

```
//////////////////////////////////////////
// second.cpp
// Command Line argument spitter
// g++ -ocmdline.exe second.cpp
// ./cmdline.exe one two three
#include <stdio.h>
int main( int argc , char **argv ){
  if ( argc == 1 ) {
    printf("No command Line Argument\n");
    // argc will be at least 1 in the case of C/C++ Programs
    // argv[0] will contain the executable name
    return 0;
  }
  //---------------- Print the Executable name
  printf("%s\n", argv[0] );
  //------------- Spit the rest of the arguments
  for( int j=1; j< argc ; ++j )
    puts(argv[j]);
}
```

C/C++ Programming under Linux - Part 3

The following program will spit the environment variables inherited when invoked.

```
////////////////////////////////
// third.cpp
// A C/C++ program to spit the environmental variables
//
// g++ -oenvspitter.exe third.cpp
// ./envspitter.exe
#include <stdio.h>
int  main( int argc , char **argv , char **envp ){
  //  env1\0\0env2\0\0env3\0\0\0\0
  char **temp = envp;
  while (*(temp+1) != 0 ) {
        puts(*temp);
        temp++;
   }
   return 0;
}
```

C/C++ Programming under Linux - Part 4

Let us see how one can compile and link a C/C++ program with multiple source files.

Here is our main

```
//fourth.cpp
// g++ -c -ofourth.o fourth.cpp
// g++ -ofourth.exe fourth.o fourth_a.cpp
//---------------------------------
////home/praseed/gnu_linux$ g++ -o fourth.exe fourth.o fourth_a.cpp
////home/praseed/gnu_linux$ ./fourth.exe
//The absolute value of -1 is 1
////home/praseed/gnu_linux$ g++ -c -o fourth_a.o fourth_a.cpp
////home/praseed/gnu_linux$ g++ -o fourth.exe fourth.o fourth_a.o
////home/praseed/gnu_linux$ ./fourth.exe
//The absolute value of -1 is 1
#include <stdio.h>
long myabs(long a );   // implementation in fourth_a.cpp

int main( int argc , char **argv ){
  printf("The absolute value of %d is %ld\n",-1,myabs(-1));
}
// EOF fourth.cpp
```

Here is the list for the Fourth_a.cpp file

```
// fourth_a.cpp
// g++ -c -o fourth_a.o fourth_a.cpp
// link with fourth.cpp ( see the heading for how to do it)
long myabs( long a ) {
   if ( a < 0 ) { a = -a; }
```

```
   return a;
}
// eof fourth_a.cpp
```

C/C++ Programming under Linux - Part 5

Let us see how we can create a Shared Library (aka DLL [in windows parlance])

a) The main file fifth.cpp

```
//fifth.cpp
// g++ -c -o fifth.o fifth.cpp (will produce fifth.o )
// g++ -ofifth.exe fifth.o ./libFiftha.so
#include <stdio.h>
// ------------ The Add function will be packaged in  a shared library (.so ) by the name "libFiftha.so"
extern "C" int Add( int x , int y );
int main(){
  int r = Add(3,4);
  printf("%d\n",r);
  return 0;
}
```

b) the source file which will be compiled and linked to form Shared library

```
//
// fifth_a.cpp
// -fpic => pic stands for position independent code
// g++ -fpic fifth_a.cpp -shared -o libFiftha.so
#include <stdio.h>

extern "C" int Add( int x , int y ) { return x + y;}
```

The output produced is

```
//home/praseed/gnu_linux$ g++ -fpic fifth_a.cpp -shared -olibFifa.so
//home/praseed/gnu_linux$ g++ fifth.cpp ./libFiftha.so
//home/praseed/gnu_linux$ ./a.out
7
```

C/C++ Programming under Linux - Part 6

In the part 5 , we learned about Dynamic Linking (or shared library). The Shared library was linked by the LD linker with the main executable.

This time , we will learn how the application can load a DLL manually and call a method through Function Pointers.

In other words , it is a classic example of Dynamic Linking of a Shared Library (Library is linked at the run time)

a) Key in The Shared library code (sixth_a.cpp)

```
// sixth_a.cpp
// g++ -fpic sixtha_a.cpp -shared -o libSixtha.so
#include <stdio.h>
extern "C" int Add( int x , int y ) { return x + y;}
// Eof sixth_a.cpp
```

b) Compile the file Add.cpp into a shared library

```
g++ -fpic sixth_a.cpp -shared -o libSixtha.so
```

c) Key in the main file (sixth.cpp)

```
// sixth.cpp
//
// g++ -o test.exe sixth.cpp -ldl
#include <stdio.h>
#include <dlfcn.h>    // necessary for dynamic loading of shared libraries

//--- typedef for a function pointer which represent int Add(int , int )
typedef int (*BinaryFunction)( int x , int y );
//--------------- user entry point
int main(){

   void * lib  = dlopen("./libSixtha.so", RTLD_LAZY);

   if ( lib == 0 ) {
        printf("Failed to load the shared library \n");
        return -1;
   }

   BinaryFunction bf = (BinaryFunction)dlsym(lib,"Add");

   if ( bf == 0 ) {
         printf("Failed to retrieve function pointer \n");
         return -1;
   }

   int r = (*bf)(3,4);
   dlclose(lib);

   printf("%d\n",r);

   return 0;
}

// Eof main.cpp
```

The Output window is given below

```
//home/praseed/gnu_linux$ g++ -o test.exe sixth.cpp
/tmp/ccFr5Iit.o: In function `main':
sixth.cpp:(.text+0x15): undefined reference to `dlopen'
sixth.cpp:(.text+0x46): undefined reference to `dlsym'
sixth.cpp:(.text+0x83): undefined reference to `dlclose'
```

```
collect2: error: ld returned 1 exit status
//home/praseed/gnu_linux$ g++ -o test.exe sixth.cpp -ldl
//home/praseed/gnu_linux$ ./test.exe
```

C/C++ Programming under Linux - Part 7

In Part 5 , when we linked a shared library created by us , we were forced to give ./ prefix before the library name (./libFiftha.so)

By exporting LD_LIBRARY_PATH with the current directory we can eliminate that

export LD_LIBRARY_PATH=.;$LD_LIBRARY_PATH

After this it is easy to link the shared library

```
//home/praseed/gnu_linux$ g++ -fpic fifth_a.cpp -shared -o libFiftha.so
//home/praseed/gnu_linux$ g++ -o test.exe fifth.cpp libFiftha.so l
//home/praseed/gnu_linux$ ./test.exe
7
```

if we give the current directory in the path through PATH environment variable , there is no need to give ./ while executing our executable

export PATH=.:$PATH

```
//home/praseed/gnu_linux$ export PATH=.:$PATH
//home/praseed/gnu_linux$ test.exe
7
```

C/C++ Programming under Linux - Part 8

We will write a file Copy Program using getchar() and putchar(). The Program will teach you I/O redirection as well. The Program is also available from the book, “The C Programming Language” by Kernighan and Ritchie (aka K&R)

```
// eighth.cpp
// g++ -oeighth.exe eighth.cpp
// ./eighth.exe < eighth.cpp > eight.dup
// cat eight.dup
#include <stdio.h>
#include <ctype.h>
int main(){
 int c;
 while ( ( c = getchar() ) != EOF )
        putchar(c);
 return 0;
}
```

The Console output is given as below

```
//home/praseed/gnu_linux$ g++ -o eighth.exe eighth.cpp
//home/praseed/gnu_linux$ ./eighth.exe < eighth.cpp > eight.dup
//home/praseed/gnu_linux$ cat eight.dup
// eighth.cpp
// g++ -oeighth.exe eighth.cpp
// ./eighth.exe < eighth.cpp > eight.dup
// cat eight.dup
#include <stdio.h>
#include <ctype.h>
int main(){
 int c;
 while ( ( c = getchar() ) != EOF )
         putchar(c);
 return 0;
}
```

C/C++ Programming under Linux - Part 9

In the part 8, we learned about file copying with the support of I/O redirection Operators.This time , we will write a filter which converts a file to upper case.With Linux Operating System's ability to string together commands , this can become very powerful.

```
// ninth.cpp
// g++ -oninth.exe ninth.cpp
// ./ninth.exe < ninth.cpp > ninth.dup
// cat eight.dup
#include <stdio.h>
#include <ctype.h>
int main(){
 int c;
 while ( ( c = getchar() ) != EOF )
         putchar(toupper(c));
 return 0;
}
```

The Console output is as given below

```
//home/praseed/gnu_linux$ g++ -oninth.exe ninth.cpp
//home/praseed/gnu_linux$ ./ninth.exe < ninth.cpp > ninth.dup
//home/praseed/gnu_linux$ cat ninth.dup
// NINTH.CPP
// G++ -ONINTH.EXE NINTH.CPP
// ./NINTH.EXE < NINTH.CPP > NINTH.DUP
// CAT EIGHT.DUP
#INCLUDE <STDIO.H>
#INCLUDE <CTYPE.H>
INT MAIN(){
 INT C;
 WHILE ( ( C = GETCHAR() ) != EOF )
```

```
        PUTCHAR(TOUPPER(C));
RETURN 0;
```

C/C++ Programming under Linux - Part 10

This time i will introduce a handy utility called ObjDump. using this i will demonstrate a technique called Self Modifying Code. This material is bit advanced. I think any one with some familiarity of x86 Assembly language can master it.

Let us get into the action

```
//tenth_a.cpp
//This function add two integers ( to keep things simple )
//
//
//g++ -c -o tenth_a.o tenth_a.cpp   // -c compile only
//

int Add( int a , int b ) { return a + b; }

// Eof tenth_a.cpp
```

Invoke ObjDump utility as follows (objdump -d -S tenth_a.o)

```
//home/praseed/gnu_linux$ objdump -d -S tenth_a.o

tenth_a.o:     file format elf64-x86-64


Disassembly of section .text:

0000000000000000 <_Z3Addii>:
   0:   55                    push   %rbp
   1:   48 89 e5              mov    %rsp,%rbp
   4:   89 7d fc              mov    %edi,-0x4(%rbp)
   7:   89 75 f8              mov    %esi,-0x8(%rbp)
   a:   8b 55 fc              mov    -0x4(%rbp),%edx
   d:   8b 45 f8              mov    -0x8(%rbp),%eax
  10:   01 d0                 add    %edx,%eax
  12:   5d                    pop    %rbp
  13:   c3                    retq
```

Now Convert the Instruction stream into an array

```
Invoke the Objdump utility as follows => objdump -d -S tenth_a.o
```

```
tenth_a.o:     file format elf64-x86-64
Disassembly of section .text:
0000000000000000 <_Z3Addii>:
   0:   55                      push   %rbp
   1:   48 89 e5                mov    %rsp,%rbp
   4:   89 7d fc                mov    %edi,-0x4(%rbp)
   7:   89 75 f8                mov    %esi,-0x8(%rbp)
   a:   8b 55 fc                mov    -0x4(%rbp),%edx
   d:   8b 45 f8                mov    -0x8(%rbp),%eax
  10:   01 d0                   add    %edx,%eax
  12:   5d                      pop    %rbp
  13:   c3                      retq
================================================
// Convert array into addfunc
char addfunc[] =
"\x55\x48\x89\xe5\x89\x7d\xfc\x89\x75\xf8\x8b\x55\xfc\x8b\x45\xf8\x01\xd0\x5d\xc3";
```

using Memory Mapped File , we can execute the code

```
///
/// tenth.cpp
//  g++ -o temp.exe tenth.cpp
// ./temp.exe ( will execute the program )

#include <stdio.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/mman.h>
#include <errno.h>
#include <string.h>

/******************
  Generated using ObjDump utility

   0:   55                      push   %rbp
   1:   48 89 e5                mov    %rsp,%rbp
   4:   89 7d fc                mov    %edi,-0x4(%rbp)
   7:   89 75 f8                mov    %esi,-0x8(%rbp)
   a:   8b 55 fc                mov    -0x4(%rbp),%edx
   d:   8b 45 f8                mov    -0x8(%rbp),%eax
  10:   01 d0                   add    %edx,%eax
  12:   5d                      pop    %rbp
  13:   c3                      retq
************************************************/


// --- The Above code snippet was converted to a string ....
// Convert array into addfunc
```

```
char addfunc[] =
"\x55\x48\x89\xe5\x89\x7d\xfc\x89\x75\xf8\x8b\x55\xfc\x8b\x45\xf8\x01\xd0\x5d\xc3";



int main( int argc , char **argv )
{

 // Allocate memory from Heap

 char *Code = (char *)malloc(100);

 // Compute the Page #
 unsigned long page = ((unsigned long)Code) & ~( getpagesize() - 1 );


 //---------- Set the Executable Bit

 if( mprotect( (char*)page, getpagesize(), PROT_READ | PROT_WRITE | PROT_EXEC ) )
  {
   perror( "mprotect failed" );
   exit( errno );
  }

 //---------- Copy the Code from addfunc to Code
 memcpy(Code,addfunc,sizeof(addfunc));

 // Cast the Code into a Function Pointer
 int (*AddFunc)(int , int ) = (int (*)(int,int))((void *)Code);

 // invoke the method
 int retval = AddFunc(4,5);

 printf("%d\n",retval);

 // ---------- Remove the executable bit

 if( mprotect( (char*)page, getpagesize(), PROT_READ | PROT_WRITE ) )
  {
   perror( "mprotect failed" );
   exit( errno );
  }

  //----------Free the heap allocated memory

  free(Code);


}

/// EOF tenth.cpp
```

The Console Output is given below

```
//home/praseed/gnu_linux$ g++ -o temp.exe tenth.cpp
//home/praseed/gnu_linux$ ./temp.exe
9
//home/praseed/gnu_linux$
```

C/C++ Programming under Linux - Part 11

This part will show how one can redirect the output of a system command to a program using Pipes. In the program given below , we use popen function to open a pipe for read access. All the program does is go through the pipe stream one line at a time and emit the line at the console.

```
// eleventh.cpp
//
//
//g++ -o test.exe eleventh.cpp
//./test.exe
//
#include <stdio.h>
#include <unistd.h>
#include <string.h>

int main(){
  FILE *fp=0;
  //--------- open a pipe to the ls command
  //--------- second parameter (r) indicates read
  if ( ( fp = popen("ls -l","r") ) == 0 ) { return -1;}
  char bfr[8192];
  //--------- Read line at a time and spit to the console
  while ( !feof(fp) ){
      fgets(bfr,8192,fp);
      printf("%s",bfr);
  }
  //-----------close the file
  fclose(fp);
}
//EOF eleventh.cpp
```

C/C++ Programming under Linux - Part 12

This part will use system function from stdlib.h to invoke a system command (Remember the Linux maxim .... Every Command is a Program and Every Program is a Command)

```
// twelth.cpp
// g++ -o test.exe twelth.cpp
// ./test.exe
#include <stdio.h>
#include <stdlib.h>
int main( int argc , char **argv ){
```

```
    int rc =  system("ls -l");
    return rc;
}
// eof twelth.cpp
```

C/C++ Programming under Linux - Part 13

In this part , we will learn how we can use pipes to sort a list of names. We are going to use sort command to achieve our objective.

```
// thirteenth.cpp
// g++ -o psp.exe thirteenth.cpp
// ./psp.exe
#include <stdio.h>
#include <unistd.h>
int main( int argc , char **argv ){
  FILE *fp = 0;
  //---------- open a pipe for writing
  if ( ( fp = popen("sort","w")) == 0 ) { return -1; }
  //------------ write to the pipe
  fprintf(fp,"Ziyad\n");
  fprintf(fp,"Anand\n");
  fprintf(fp,"Arjun\n");
  fprintf(fp,"Ram\n");
  fprintf(fp,"Hooper\n");
  pclose(fp);  // when you close the pipe sorting will happen..
  return 0;
}
// EOF thirteenth.cpp
```

The Console Output is as follows

```
//home/praseed/gnu_linux$ g++ -o psp.exe thirteenth.cpp
//home/praseed/gnu_linux$ ./psp.exe
Anand
Arjun
Hooper
Ram
Ziyad
//home/praseed/gnu_linux$
```

C/C++ Programming under Linux - Part 14

In this part we will learn about C standard file I/O to dump the contents of a text file... (binary file I/O will be covered in a future post).This method is portable across platforms (Win32 and Linux for sure)

```
//  fourteenth.cpp
```

```
//      Demonstrates C standard I/O which is portable across platforms
//  g++ -o txtfiledump fourteenth.cpp
//  ./txtfiledump fourteenth.cpp
#include <stdio.h>
int main( int argc , char **argv ){
   //----------- the program expects one argument
   //----------- if there are more or less..it is considered as error
   if ( argc != 2 ) {
      fprintf(stdout,"Usage: txtfiledump  &lt;filename> \n");
      return -1;
   }
   //--------- argv[1] contains the first argument
   //----------argv[0] contains the executable file name
   FILE *fp = fopen(argv[1],"rt");
   //--------- Failed to  Open the file
   if ( fp == 0 ){
       fprintf(stdout,"Error Opening File %s\n",argv[1] );
       return -2;
   }
   char bfr[4096];
   //----------- While not the end of file
   //----------- take one line at a time and spit
   while ( !feof(fp) ){
      fgets(bfr,4096,fp);
      fprintf(stdout,"%s",bfr);
   }
   //--------- close the file ...
   fclose(fp);
}
// EOF fourteenth.cpp
```

C/C++ Programming under Linux - Part 15

In this part , we learn about Binary I/O through C run time library calls. The Buffered I/O calls fread and fwrite is used to write and read from binary files.

```
// fifteenth.cpp
// Program To Demonstrate Binary I/O using C run time library function
// fopen/fread/fwrite/fclose family of functions
// g++ -o mycp fifteenth.cpp
//
//  ./mycp fifteenth.cpp mycp.tmp
//
#include <stdio.h>
int main( int argc , char **argv ){
  if ( argc != 3 ) {
       fprintf( stdout,"usage: mycp <srcfile> <destfile> \n");
       return -1;
  }
  FILE *inp , *outp;
```

```
  inp = outp = 0;
  if ( ( inp = fopen(argv[1],"rb") ) == 0 ){
       fprintf(stdout,"Failed to Open src file\n");
       return -2;
  }
  if ( ( outp = fopen(argv[2],"wb") ) == 0 ){
       fclose(inp);
       fprintf(stdout,"Failed to Open desitnation file\n");
       return -2;
  }
  char bfr[8192];
  int num = -1;
  while ( ( num  = fread(bfr,1,8192,inp) ) == 8192 ){
      fwrite(bfr,1,8192,outp);
  }
  if ( num > 0 )
    fwrite(bfr,1,num,outp); // write the remaining bytes
  fclose(inp);
  fclose(outp);
}
//EOF: fifteenth.cpp
```

C/C++ Programming under Linux - Part 16

In this part , we will learn about open/read/write/close model of FILE I/O. These are system calls (interrupt 80 calls) and are exposed to the programmer as C callable subroutines.

```
// sixteenth.cpp
// A program to copy file to another using Linux System Calls
// These are conveniently packaged as C called routine by glibc
// g++ -o sysfile sixteenth.cpp
// ./sysfile sixteenth.cpp sysfile.out
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <string.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <unistd.h>
int main( int argc , char **argv ){
  if ( argc != 3 ) {
    fprintf( stdout,"usage: sysfile <srcfile> <destfile> \n");
    return -1;
  }
  int inp_fd; // input file descriptor
  int out_fd; // output file descriptor
  // use Linux System calls to open file...
  // O_RDONLY => Read only access
```

```
  if ( ( inp_fd = open(argv[1],O_RDONLY) ) == -1 ){
    fprintf(stdout,"Failed to Open src file\n");
    return -2;
  }
  // This file should not exist when the call is made
  if ( ( out_fd = open(argv[2],O_WRONLY | O_CREAT | O_EXCL) ) == -1 ){
    close(inp_fd);
    fprintf(stdout,"Failed to Open desitnation file\n");
    return -2;
  }
  char bfr[8192];
  int num = -1;
  while ( ( num = read(inp_fd,bfr,8192) ) == 8192 ){
    write(out_fd,bfr,8192);
  }
  if ( num > 0 )
    write(out_fd,bfr,num); // write the remaining bytes
  close(inp_fd);
  close(out_fd);
}
//EOF: Sixteenth.cpp
```

C/C++ Programming under Linux - Part 17

In this part , we will learn about Forking a child process. This was the way multi tasking was achieved at one point of time in Unix. Now a days , people use Pthreads to achieve the same objective.

insert a call to fork system call

```
int main()
{
   //------------ do some logic
  if ( fork() == 0 ){
      // This Part is child process Exit once we finished the action
  }
  //---------- continue with the parent process logic
}
```

Here Is a simple program which demonstrates the idea of forking a process

```
////////////////////////////
//seventeenth.cpp
// A Simple Program to Demonstrate Forking of a new process....
// g++ -o forktest.exe seventeenth.cpp
// ./forktest.exe
#include <stdio.h>
#include <sys/types.h>
#include <stdlib.h>
#include <unistd.h>
int main( int argc , char **argv ){
  //------------ Every Process has a Process id
```

```
  printf("PARENT Process ID = %d\n",getpid() );
  int child_pd;
  // Here we make the fork system call to create a child process
  if (  ( child_pd = fork() )  == 0 ) {

    //---------- We check the return value of Fork call
    //---------  if it is zero , this part is child process
    printf("CHILD Process ID = %d\n",getpid() );
    exit(0);
  }
  // continue with the parent process
  // sleep for 3 seconds
  sleep(3);
  printf("PARENT Process ID = %d\n",getpid() );
}
//EOF:seventeenth.cpp
```

Here is the console output

```
//home/praseed/gnu_linux$ ./forktest.exe
PARENT Process ID = 2410
CHILD Process ID = 2411
PARENT Process ID = 2410
//home/praseed/gnu_linux$
```

C/C++ Programming under Linux - Part 18

In this part , we will enhance the program which we wrote in the part 17. We will be using waitpid (wait-process-id call) to query the exit status of the child program.

```
////////////////////////////
//eighteenth.cpp
// A Demonstrate Forking of a new process....
// The Program waits for the status of the child process to continue
// g++ -o forktest2.exe eighteenth.cpp
// ./forktest2.exe
#include <stdio.h>
#include <sys/types.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <sys/wait.h>
int main( int argc , char **argv ){
  //------------ Every Process has a Process id
  printf("PARENT Process ID = %d\n",getpid() );
  int child_pd;
  int ps_status;
  // Here we make the fork system call to create a child process
  if (  ( child_pd = fork() )  == 0 ) {
    //---------- We check the return value of Fork call
```

```
    //---------  if it is zero , this part is child process
    printf("CHILD Process ID = %d\n",getpid() );
    exit(3);
  }
  // continue with the parent process
  // if the waitpid call succeded , we will get the
  // child process id as return value
  if ( waitpid(child_pd,&ps_status,0) != child_pd ){
    // if there is no error , we should not reach here
    printf("Error In execution of Child Process\n");
  }
  if ( WIFEXITED(ps_status) ) {
    printf("The child process terminated properly with status %d\n",
    WEXITSTATUS(ps_status));
  }
  printf("PARENT Process ID = %d\n",getpid() );
}
//EOF: eighteenth.cpp
```

The Console Output of the above program is given below

```
//home/praseed/gnu_linux$ g++ eighteenth.cpp
//home/praseed/gnu_linux$ ./a.out
PARENT Process ID = 2422
CHILD Process ID = 2423
The child process terminated properly with status 3
PARENT Process ID = 2422
//home/praseed/gnu_linux$
```

C/C++ Programming under Linux - Part 19

In this Part , we will start fiddling with Directory Access functions in Linux.The opendir/readdir/closedir/ model is very similar to open/read/close programming model for accessing files. The only difference is addition of dir suffix with every call.

The Algorithm for dumping the contents of a Directory is as follows

a) Open Dir (opendir)

b) As long as there is entry , Read an entry (readdir)

c) Check the Directory Bit , if true , it is a directory

d) Otherwise , print file name , loop back to [b]

e) When finished , Close the Directory. (closedir)

The following program given below puts the algorithm into practice...

```
//nineteenth.cpp
// This Program demonstrates simple DirectoryTraversal
```

```
// g++ -o SimpleDir.exe nineteenth.cpp
// ./SimpleDir.exe
#include <stdio.h>
#include <unistd.h>
#include <dirent.h>
#include <stdlib.h>
#include <sys/stat.h>
#include <string.h>
////////////////////////////////////
//
// Emits the content of Directory ...
//
void DumpDirectory( char *DirName ){
  DIR *dir = 0;
  struct stat status;
  struct dirent *direntry;
  char TempDir[8192];
  if (( dir = opendir(DirName) ) == 0 ) {
    fprintf(stdout,"Failed to Get into %s\n",DirName);
    return;
  }
  //----------------- Read one entry at a Time
  //----------------- Check the status ,
  //------------ if Directory give appropriate message
  while ( ( direntry = readdir(dir) ) != 0 )
  {
    strcpy(TempDir,DirName);
    strcat(TempDir,direntry->d_name);
    lstat(TempDir,&status);
    //----------- Check the Directory Bit
    if ( S_ISDIR( status.st_mode ) ){
      //---------- ignore . , ..
      if ( direntry->d_name[0] == '.')
        continue;
      fprintf(stdout,"Directory @ %s\n",direntry->d_name );
    }
    else {
      fprintf(stdout,"File @ %s\n",direntry->d_name );
    }
  }
  //------------- Finished Read the Directory...
  closedir(dir);
  return;
}
// Driver Program
int main( int argc , char **argv ){
  const char *dirname = "/usr/lib/";
  DumpDirectory((char *)dirname);
}

// Eof nineteenth.cpp
```

C/C++ Programming under Linux - Part 20

In this part , we will extend the program we wrote in the step 19 to support sub directory processing. This is a well known algorithm to most programmers.

```
////////////////////////////////
//twentieth.cpp
// This Program demonstrates Recursive Directory Traversal
// g++ -o Directory.exe twentieth.cpp
// ./Directory.exe
#include <stdio.h>
#include <unistd.h>
#include <dirent.h>
#include <stdlib.h>
#include <sys/stat.h>
#include <string.h>


// A Recursive Directory Traversal Routine
//
void TraverseDirectory( char *DirName , int level ){
  DIR *dir = 0;
  struct stat status;
  struct dirent *direntry;
  //---------------- open the directory
  if (( dir = opendir(DirName) ) == 0 ) {
    fprintf(stdout,"Failed to Get into %s\n",
      DirName);
    return;
  }
  //---------- change the directory
  chdir(DirName);
  //------------ Try to read the contents of the
  // current directory
  while ( ( direntry = readdir(dir) ) != 0 )
  {
    lstat(direntry->d_name,&status);

    if ( S_ISDIR( status.st_mode ) )
    {
      if ( direntry->d_name[0] == '.')
        continue;
      fprintf(stdout,"%*s%s/\n",level ,"",
      direntry->d_name );

      //---------- Recurse to Process the Sub directory
      TraverseDirectory(direntry->d_name,level + 4 );
    }

    else {
```

```
      fprintf(stdout,"%*s%s/\n",level ,
      "",direntry->d_name );

    }

  }
  //------------ move to the previous directory
  chdir("..");
  //------------ close the directory
  closedir(dir);
  return;
}
//-------------------entry point
int main( int argc , char **argv ){
  char directory_buffer[8192];
  strcpy(directory_buffer,"/usr/lib/");
  TraverseDirectory(directory_buffer,0);
}
//EOF: twentieth.cpp
```

C/C++ Programming under Linux - Part 21

In this previous part , we changed (to) the directory to get information in each directory. This is dangerous, if you are running as a root and program terminated prematurely.

In this part , we will modify the program to avoid this pitfall.

```
////////////////////////////////
//twentyfirst.cpp
// This Program demonstrates Recursive Directory Traversal
// g++ -o Directory2.exe twnetyfirst.cpp
// ./Directory2.exe


#include <stdio.h>
#include <unistd.h>
#include <dirent.h>
#include <stdlib.h>
#include <sys/stat.h>
#include <string.h>

//////////////////////
//
// This Routine does not change directories to traverse the
// directory for it's entries.
//

void TraverseDirectory( char *DirName , int level )
{
```

```
  DIR *dir = 0;
  struct stat status;
  struct dirent *direntry;
  char TempDir[8192];

  strcpy(TempDir,DirName);
  if (( dir = opendir(TempDir) ) == 0 ) {
   fprintf(stdout,"Failed to Get into %s\n",DirName);
   return;
  }

  while ( ( direntry = readdir(dir) ) != 0 )
  {
   //------------- Copy the Directory name to Temp Dir
   //------------- Add the entry (can be directory or file )
   strcpy(TempDir,DirName);
   strcat(TempDir,direntry->d_name);

   //--------- We need to give the fully qualified
   //--------- path name to retrieve the status
   lstat(TempDir,&status);


   if ( S_ISDIR( status.st_mode ) )
   {
    if ( direntry->d_name[0] == '.')
     continue;
    fprintf(stdout,"%*s%s/\n",level ,
    "", direntry->d_name );
    //--------- TempDir , contains fully qualified file name
    TraverseDirectory(TempDir,level + 4 );
   }

   else {
    fprintf(stdout,"%*s%s/\n",level ,
     "",direntry->d_name );

   }
  }
  closedir(dir);
  return;
}
```

C/C++ Programming under Linux - Part 22

In this part, we will learn about signal handling in Linux. While a program is running , we can use CTRL-C to terminate the program. We will make it mandatory that a user has to press CTRL-C three times to terminate this program.

```
//////////////////
```

```
//twentysecond.cpp
// Press CTRL-C three times to terminate this program
// g++ -o Signal.exe twentysecond.cpp
// ./Signal.exe
#include <signal.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <stdlib.h>
void interrupt_handler( int cause ){
  static int test = 0;
  test++;
  #if 0 // make this 1 to compile #if part
    if ( test == 3 )
      exit(0);
    else
      printf("Press CTRL-C %d times\n",
        3-test );
  #else

    if ( test == 2 ) {
      //--------- Reset to the defualt behaviour
      signal(SIGINT,SIG_DFL);
    }
    else
      printf("Press CTRL-C %d times\n", 3-test );
  #endif

}


int main( int argc , char **argv )
{
  signal( SIGINT , interrupt_handler);
  while (1)
  {
    //---------- an infinite loop to demonstrate signal
    //------------ handling
  }
  return 0;
}
//EOF:twentysecond.cpp
```

C/C++ Programming under Linux - Part 23

In this part , we will see how we can use memory mapped files. Memory Mapped Files are a handy mechanism by which we can treat a file as a memory based array. Whatever we write to the mapped array will go to the file.One nice example is open a file , map it as a array of characters and use C/C++ string functions to search a file. Microsoft Windows uses Memory mapped files for loading it's executables.The following program writes values from [0-19] to a file.

The file is once again opened and mapped as an array of int. We double the value in the array. (*2)

When we dump the contents of the file, we will see values in the range [0-38]

```
////////////////////
// twentythird.cpp
// A Program to demonstrate Memory Mapped files
// g++ -o mmaptest.exe twentythird
//
#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>
#include <stdlib.h>
#include <sys/mman.h>
#include <string.h>

////////////////////////////
//
// This Function Writes values to a binary file.
// This function will write values from 0-19
// Total file size will be 80 bytes
// 20*sizeof(int)
//
void PopulateValues()
{
  unlink("TEST.tmp");
  int fd = open("TEST.tmp",O_RDWR | O_CREAT );

  if ( fd == -1 )
   {
     fprintf(stdout,"P9:Failed to open TEST.bin %d\n",fd);
     exit(0);
   }

   int val;

   for( val = 0; val < 20; ++val )
     write(fd,&val,4 );

  close(fd);
}

////////////////////////////
//
//
//
int main()
{
  PopulateValues();

  int fd;
  if ( ( fd = open("TEST.tmp",O_RDWR) ) == -1 )
```

```
  {
     fprintf(stdout,"Failed to open TEST.tmp %d\n",fd);
     exit(-1);
  }

  ////////////////////////////////////////
  //  Map the entire contents of a file to a pointer...
  //
  int *mapped_values = (int *)
          mmap(0,
              20*sizeof(int),
              PROT_READ | PROT_WRITE,
              MAP_SHARED,fd,0 );

  ////////////////////////////////////////////
  //
  //  Now we can treat the whole file as an array of integer
  //
  int val;
  for(val=0; val<20; ++val )
       mapped_values[val] *= 2; // double the value...

  msync((void *)mapped_values,20*sizeof(int),MS_ASYNC);
  munmap((void *)mapped_values,20*sizeof(int));

  ////////////////////////////////////////////////////
  // Now reset the file pointer and dump the values...
  //
  lseek(fd,0,0);

  for( int i=0; i<20; ++i )
  {
     read(fd,&val,4);
     printf("%d\n",val);
  }
  close(fd);
  unlink("TEST.tmp");
}
//EOF: twentythird.cpp
```

The Console Output is

```
//home/praseed/gnu_linux$ g++ twentythird.cpp
//home/praseed/gnu_linux$ ./a.out
0
2
4
6
8
10
12
```

```
14
16
18
20
22
24
26
28
30
32
34
36
38
```

C/C++ Programming under Linux - Part 24

In a Previous Part, we learned about the signal API under Linux. In this part , we will learn about POSIX compliant signal API.

```
////////////////////////////////////
// twentyfourth.cpp
// POSIX Signaling API demo
// g++ twentyfourth.cpp
// ./a.out
#include <sys/types.h>
#include <sys/wait.h>
#include <signal.h>
#include <unistd.h>
#include <stdio.h>
#include <errno.h>
#include <string.h>
#include <stdlib.h>
// CTRL-C handler
//
void SigIntHandler( int sig , siginfo_t *siginfo , void *ign ){
       printf("Pressed CTRL-C to exit \n");
       exit(0);
}
//---- Entry Point
int main(){
  struct sigaction action;
  memset(&action,0,sizeof(action));
  action.sa_flags = SA_SIGINFO;
  action.sa_sigaction = SigIntHandler;
  sigaction(SIGINT,&action,0);
  while ( 1 )
    ;
}
```

The Console Output is as follows

```
//home/praseed/gnu_linux$ g++ twentyfourth.cpp
//home/praseed/gnu_linux$ ./a.out
^CPressed CTRL-C to exit
//home/praseed/gnu_linux$
```

C/C++ Programming under Linux - Part 25

In this part , we will learn about Named Pipes. Named Pipe is one of the inter process communication mechanism available in Linux. Microsoft SQL server and client talks to each other using Named Pipes.

The Following Program spits to the console whatever recieved through the named pipe.

```
////////////////////////////////////////
//twentyfifth.cpp
// Program to demonstrate Named Pipes
// g++ -o np_server.exe twentyfifth.cpp
// ./np_server.exe  &  ( Run in the back ground )
#include <stdio.h>
#include <unistd.h>
#include <string.h>
#include <stdlib.h>
#include <fcntl.h>
#include <sys/types.h>
#include <sys/stat.h>
//-------------- Entrypoint
int main( int argc , char ** ){
  //-------------- if named pipe does not exist
  //-------------- create one
  if ( access("named_pipe@786",F_OK ) == -1 ){
    int stat =  mkfifo("named_pipe@786",0777);

    if ( stat != 0 )
    {
       fprintf(stdout,"Failed to create named pipe\n");
       exit(-1);
    }

  }

 //----------- open the fifo queue in read only mode ...
 int pipe_handle = open("named_pipe@786",O_RDONLY  );

 if ( pipe_handle == -1 )
 {
    fprintf(stdout,"Failed to open the named pipe\n");
    exit(-2);
 }

 int num_read;
 char buffer[128];
```

```
 memset(buffer,0,128);

 while (( num_read = read( pipe_handle,buffer,127)) != -1 )
 {
      //------ nothing has been read from the client
      if ( num_read == 0 ) {
          sleep(5);
      }

      if ( strcmp(buffer,"QUIT\n") == 0 )
            exit(0);

      if ( buffer[0] == 0 )
           continue;
      printf("%s \n",buffer);
      memset(buffer,0,128);
 }

 close(pipe_handle);

}
//twentyfifth.cpp
```

The Console Output is given below

```
//home/praseed/gnu_linux$ g++ -o np_server.exe twentyfifth.cpp
//home/praseed/gnu_linux$ ./np_server.exe &
[2] 2747
//home/praseed/gnu_linux$ echo "Hello" > named_pipe@786
Hello

//home/praseed/gnu_linux$ echo "Hello Word" > named_pipe@786
//home/praseed/gnu_linux$ Hello Word

echo "QUIT" > named_pipe@786
//home/praseed/gnu_linux$
```

C/C++ Programming under Linux - Part 26

Today , we will learn about POSIX Threads. When a Program starts , it has got a single thread (path) of execution. In GNU Linux , we can create additional threads (paths) which can run asynchronously with the main thread.

Read about topics like MultiThreading , POSIX threads on the Wiki to know more about this. The program given below is a multi threaded version of hello world program.

```
//  twentysixth.cpp
// A Simple POSIX thread program
// g++ -o thread_first.exe twentysixth.cpp -lpthread
// ./thread_first.exe
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <pthread.h>
// Thread Entry Point
void *PrintFunction( void *temp ) {
    printf("Hello World from Worker Thread\n");
    return 0;
}
// User Entry Point for the Application
int main( int argc , char **argv ){
      //  Thread Handle
      pthread_t th;
      // ---------- Create a Thread and start the execution
      int re_val = pthread_create( &th, NULL, PrintFunction, 0);
      printf("Hello World from the main Thread\n");
      // ---------- Wait for Worker thread to finish
      pthread_join( th, NULL);
      return 0;
}
```

The Output of the Console is as follows

```
//home/praseed/gnu_linux$ g++ -o first_thread.exe twentysixth.cpp
/tmp/ccdHXjP1.o: In function `main':
twentysixth.cpp:(.text+0x56): undefined reference to `pthread_create'
twentysixth.cpp:(.text+0x76): undefined reference to `pthread_join'
collect2: error: ld returned 1 exit status
//home/praseed/gnu_linux$ g++ -o first_thread.exe twentysixth.cpp -lpthread
//home/praseed/gnu_linux$ ./first_thread.exe
Hello World from the main Thread
Hello World from Worker Thread
```

```
//home/praseed/gnu_linux$
```

C/C++ Programming under Linux - Part 27

In this part , we will learn how to pass parameters to a thread function. we need to make only a slight modification to the earlier program to achieve this.

```
////////////////////////////////////////
// twentyseventh.cpp
// A Simple POSIX thread program to demonstrate
// passing of parameters to the thread entry point
// g++ -o thread_second.exe twentyseventh.cpp -lpthread
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <pthread.h>
// Thread Entry Point
void *PrintFunction( void *temp ){
    int s = *((int *)temp)  ;
    printf("Hello World from Worker Thread %d\n",s);
    return 0;
}
// User Entry Point for the Application
int main( int argc , char **argv ){
  //  Thread Handle
  pthread_t th;
  // Create a Thread and start the execution
  int param = 100;
  int re_val = pthread_create( &th, NULL, PrintFunction,
     (void *)&param  // Thread parameter
   );
   printf("Hello World from the main Thread\n");
  // Wait for Worker thread to finish
  pthread_join( th, NULL);
  return 0;
}
//EOF:twentyseventh.cpp
```

The Console Output is as follows

```
//home/praseed/gnu_linux$ g++ -o thread_second.exe twentyseventh.cpp -lpthread
//home/praseed/gnu_linux$ ./thread_second.exe
Hello World from the main Thread
Hello World from Worker Thread 100
//home/praseed/gnu_linux$
```

C/C++ Programming under Linux - Part 28

In this part , we learn about POSIX mutex. Mutex stands for Mutual Exclusion. Only one thread will be able to enter a region guarded by a mutex (co-operating threads should look for same variable).

```
// twentyeighth.cpp
// A Simple POSIX thread program to demonstrate
// mutex for guarding modification to shared variables.
// g++ -o thread_third.exe thread_third.cpp -lpthread
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <pthread.h>
// some global variables;
int Counter=0;
pthread_mutex_t m1 = PTHREAD_MUTEX_INITIALIZER;
// Thread Entry Point
void *PrintFunction( void *temp ){
    pthread_mutex_lock( &m1 ); //guard area
    Counter++; // we can safely modify variable now
    pthread_mutex_unlock(&m1); //remove guard
    int s = *((int *)temp)  ;
    printf("Hello World from Worker Thread %d\n",s);
    return 0;
}
// User Entry Point for the Application
int main( int argc , char **argv ){
  //  Thread Handle
  pthread_t th;
  pthread_t th1;
  // Create two Threads and start the execution
  int param = 100;
  int re_val = pthread_create( &th, NULL, PrintFunction, (void *)&param  // Thread parameter
  );
  int param1=200;
  int re_val1 = pthread_create( &th1, NULL, PrintFunction,(void *)&param1  // Thread parameter
  );
  printf("Hello World from the main Thread\n");
  // Wait for Worker threads to finish
  pthread_join( th, NULL);
  pthread_join(th1,NULL);
  return 0;
}
//EOF:twentyeighth.cpp
```

The Output of the Console is as follows

```
//home/praseed/gnu_linux$ g++ -o thread_third.exe twentyeighth.cpp -lpthread
//home/praseed/gnu_linux$ ./thread_third.exe
Hello World from Worker Thread 100
Hello World from the main Thread
Hello World from Worker Thread 200
praseed@LAPTOP-QIG6F4HD://home/praseed/gnu_linux$
```