Jawaharlal Nehru National College of Engineering

NAVULE (SAVALANGA Road), Shivamogga – 577204. Karnataka

Department of Computer Science and Engineering

Unix System Programming
and
Compiler Design
(10CSL68),VI Sem CSE

LAB MANUAL

Prepared by

Mr. Chetan K R
Asst. Professor,
Dept. of CSE,
JNNCE Shivamogga

Mr. Manohar Nelli V
Asst. Professor,
Dept. of CSE,
JNNCE Shivamogga

Mr. Chakrapani D S
Asst. Professor,
Dept. of CSE,
JNNCE Shivamogga

Syllabus

PART-A

1. Write a C/C++ POSIX compliant program to check the following limits:

   (i) No. of clock ticks

   (ii) Max. no. of child processes

   (iii) Max. path length

   (iv) Max. no. of characters in a file name

   (v) Max. no. of open files/ process

2. Write a C/C++ POSIX compliant program that prints the POSIX defined configuration options supported on any given system using feature test macros.

3. Consider the last 100 bytes as a region. Write a C/C++ program to check whether the region is locked or not. If the region is locked, print pid of the process which has locked. If the region is not locked, lock the region with an exclusive lock, read the last 50 bytes and unlock the region.

4 Write a C/C++ program which demonstrates interprocess communication between a reader process and a writer process. Use mkfifo, open, read, write and close APIs in your program.

5 a) Write a C/C++ program that outputs the contents of its Environment list

b) Write a C / C++ program to emulate the unix ln command

6 Write a C/C++ program to illustrate the race condition.

7 Write a C/C++ program that creates a zombie and then calls system to execute the ps command to verify that the process is zombie.

8 Write a C/C++ program to avoid zombie process by forking twice.

9 Write a C/C++ program to implement the system function.

10 Write a C/C++ program to set up a real-time clock interval timer using the alarm API.

PART-B

11. Write a C program to implement the syntax-directed definition of “if E then S1” and “if E then S1 else S2”. (Refer Fig. 8.23 in the text book prescribed for 06CS62 Compiler Design, Alfred V Aho, Ravi Sethi, and Jeffrey D Ullman: Compilers- Principles, Techniques and Tools, 2nd Edition, Pearson Education, 2007).

12. Write a yacc program that accepts a regular expression as input and produce its parse tree as output.

Note: *In the examination each student picks one question from the lot of all 12 questions*

Content Sheet

1. Write a C/C++ POSIX compliant program to check the following limits:

1. **No. of clock ticks**
2. **Max. no. of child processes**
3. **Max. path length**
4. **Max. no. of characters in a file name**
5. **Max. no. of open files/ process**

Explanation:

Limits:

There are three types of limits. They are,

1. Compile-time limits (headers).
2. Runtime limits that are not associated with a file or directory (the sysconf function).
3. Runtime limits that are associated with a file or directory (the pathconf and fpathconf functions).

sysconf, pathconf, and fpathconf Functions:

The runtime limits are obtained by calling one of the following three functions.

```
#include <unistd.h>

long sysconf(int name);

long pathconf(const char *pathname, int name);


Long fpathconf(int filedes, int name);
```

All three return: corresponding value if OK, 1 on error

The difference between the last two functions is that one takes a pathname as its argument and the other takes a file descriptor argument.

Table 1 lists the *name* arguments that sysconf uses to identify system limits. Constants beginning with _SC_ are used as arguments to sysconf to identify the runtime limit. Table 2 lists the *name* arguments that are used by pathconf and fpathconf to identify system limits. Constants beginning with _PC_ are used as arguments to pathconf and fpathconf to identify the runtime limit

Name of limit	Description	*name* argument
CHILD_MAX	maximum number of processes per real user ID	_SC_CHILD_MAX
clock ticks/second	number of clock ticks per second	_SC_CLK_TCK
OPEN_MAX	maximum number of open files per process	_SC_OPEN_MAX
SAVED_IDS	The _POSIX_SAVED_IDS value	_SC_SAVED_IDS

Table 1: Limits and *name* arguments to `sysconf`

Name of limit	Description	*name* argument
CHOWN_RESTRICTED	The _POSIX_CHOWN_RESTRICTED value	_PC_CHOWN_RESTRICTED
NAME_MAX	maximum number of bytes in a filename (does not include a null at end)	_PC_NAME_MAX
PATH_MAX	maximum number of bytes in a relative pathname, including the terminating null	_PC_PATH_MAX

Algorithm

Step 1: Start
Step 2: Query the clock tick limits
Step 3: If yes, print the number of clock ticks. Go to Step 5
Step 4: If no, print the feature is not supported
Step 5: Query the limit on max. no. of child process supported
Step 6: If yes, print the max. no. of child process supported Go to Step 8
Step 7: If no, print the feature is not supported
Step 8: Query the limit on max. no. of characters in the pathname
Step 9: If yes, print the max. no. of characters in the pathname. Go to Step 11
Step 10: If no, print the feature is not supported
Step 11: Query the limit on max. no. of characters in the filename
Step 12: If yes, print the max. no. of characters in the filename. Go to Step 14
Step 13: If no, print the feature is not supported
Step 14: Query the limit on max. no. of files that can be opened
Step 15: If yes, print the max. no. of files that can be opened. Go to Step 17
Step 16: If no, print the feature is not supported
Step 17: End

Program

```
#define _POSIX_SOURCE
#define _POSIX_C_SOURCE 199309L
#include<unistd.h>
#include<iostream.h>
int main()
{
  int res;
  if((res=sysconf(_SC_CLK_TCK))==-1)
    cout << "clock ticks not supported" << endl;
```

```
 else
    cout << "No of clock ticks is " <<res <<endl;

  if((res=sysconf(_SC_CHILD_MAX))==-1)
    cout << "child max not supported" << endl;
 else
    cout << "No of child processes that can be created
simulataneously " <<res <<"Min value" << _POSIX_CHILD_MAX<<endl;


  if((res=pathconf("/",_PC_PATH_MAX))==-1)
    cout << "path max not supported" << endl;
 else
    cout << "Max path length is " <<res <<endl;

  if((res=pathconf("/",_PC_NAME_MAX))==-1)
    cout << "name max not supported" << endl;
 else
    cout << "Max size of file name " <<res <<endl;

  if((res=sysconf(_SC_OPEN_MAX))==-1)
    cout << "open max not supported" << endl;
 else
    cout << "No of files/processes that can be opened simultaneously
is " <<res <<endl;

}
```

Output

```
[root@localhost new2]# c++ -Wno-deprecated s1.cpp
[root@localhost new2]# ./a.out
No of clock ticks is 100
No of child processes that can be created simulataneously 999Min value25
Max path length is 4096
Max size of file name 255
No of files/processes that can be opened simultaneously is 1024
[root@localhost new2]#
```

2. Write a C/C++ POSIX compliant program that prints the POSIX defined configuration options supported on any given system using feature test macros

Explanation:

POSIX.1 defines a set of feature test macro‟s which if defined on a system, means that the system has implemented the corresponding features. All these test macros are defined in **<unistd.h>** header.

Feature test macro	Effects if defined on as system
_POSIX_JOB_CONTROL	The system supports the BSD style job control.
_POSIX_SAVED_IDS	Each process running on the system keeps the saved set UID and the set-GID, so that they can change its effective user-ID and group-ID to those values via seteuid and setegid API's respectively.
_POSIX_CHOWN_RESTRICTED	If the defined value is -1, users may change ownership of files owned by them, otherwise only users with special privilege may change ownership of any file on the system. If this constant is undefined in <unistd.h> header, user must use the pathconf or fpathconf function to check the permission for changing ownership on a per-file basis.
_POSIX_NO_TRUNC	If the defined value is -1, any long pathname passed to an API is silently truncated to NAME_MAX bytes, otherwise error is generated. If this constant is undefined in <unistd.h> header, user must use the pathconf or fpathconf function to check the path name truncation option on a per-directory basis.
_POSIX_VDISABLE	If defined value is -1, there is no disabling character for special characters for all terminal device files. Otherwise the value is the disabling character value. If this constant is undefined in <unistd.h> header, user must use the pathconf or fpathconf function to check the disabling character option on a per-terminal device file basis.

Algorithm:

Step 1: Start
Step2: Check whether Job control is supported.
Step 3: If yes, print the message "job control is supported". Go to Step 5
Step 4: Print the message "Job control is not supported".
Step 5: Check whether Saved IDs is supported.
Step 6: If yes, print the message "Saved IDs is supported". Go to Step 7
Step 7: Print the message "Saved IDs is not supported".
Step 8: Check whether Change of Ownership is supported.
Step 9: If yes, print the message "Change of ownership is supported" and also print its value. Go to Step 11
Step 10: Print the message "Change of ownership is not supported".
Step 11: Check whether No Truncation option is supported.

Step 12: If yes, print the message "No Truncation is supported" and also print its value. Go to Step 14
Step 13: Print the message "No Truncation option is not supported".
Step 14: Check whether Vdisabling is supported.
Step 15: If yes, print the message "VDisabling is supported" and also print its value. Go to Step 11
Step 16: Print the message "VDisabling is not supported".
Step 17: End

Program (Basic version)

```
#define _POSIX_SOURCE
#define _POSIX_C_SOURCE 199309L
#include<unistd.h>
#include<iostream.h>
int main()
{

  #ifdef _POSIX_JOB_CONTROL
    cout << "Job control is supported" <<endl;
 #else
  cout << "Job control is not supported" << endl;
 #endif

  #ifdef _POSIX_SAVED_IDS
    cout << "Saved Ids is supported" <<endl;
 #else
  cout << "Saved Ids is not supported" << endl;
 #endif

  #ifdef _POSIX_CHOWN_RESTRICTED
    cout << "Change of ownership is supported and value is "<<
_POSIX_CHOWN_RESTRICTED<<endl;
 #else
  cout << "Change of ownership is not supported"<<endl;
 #endif

  #ifdef _POSIX_NO_TRUNC
    cout << "No truncation is supported and value is "<<
_POSIX_NO_TRUNC<<endl;
 #else
  cout << "No truncation is not supported"<<endl;
 #endif

  #ifdef _POSIX_VDISABLE
    cout << "Disabling is supported and disabling char is"<<
_POSIX_VDISABLE<<endl;
 #else
  cout << "Disabling is not supported"<<endl;
 #endif

}
```

Output

```
[root@localhost usp2]# c++ -Wno-deprecated bs2.cpp
[root@localhost usp2]# ./a.out
Job control is supported
Saved Ids is supported
Change of ownership is supported and value is 1
No truncation is supported and value is 1
Disabling is supported and disabling char is
[root@localhost usp2]#
```

Program (with colors and delay)

```
#define _POSIX_SOURCE
#define _POSIX_C_SOURCE 199309L
#include<unistd.h>
#include<iostream.h>
int main()
{
   cout << "\033[1;31m" ;
  #ifdef _POSIX_JOB_CONTROL
    cout << "Job control is supported" <<endl;
 #else
  cout << "Job control is not supported" << endl;
 #endif
  sleep(5);
   cout << "\033[1;32m" ;
  #ifdef _POSIX_SAVED_IDS
    cout << "Saved Ids is supported" <<endl;
 #else
  cout << "Saved Ids is not supported" << endl;
 #endif
sleep(5);
   cout << "\033[1;33m";
  #ifdef _POSIX_CHOWN_RESTRICTED
    cout << "Change of ownership is supported and value is "<<
_POSIX_CHOWN_RESTRICTED<<endl;
 #else
  cout << "Change of ownership is not supported"<<endl;
 #endif

sleep(5);
  cout << "\033[1;34m";
  #ifdef _POSIX_NO_TRUNC
    cout << "No truncation is supported and value is "<<
_POSIX_NO_TRUNC<<endl;
 #else
  cout << "No truncation is not supported"<<endl;
 #endif

sleep(5);
 cout << "\033[1;35m";
  #ifdef _POSIX_VDISABLE
    cout << "Disabling is supported and disabling char is"<<
```

```
_POSIX_VDISABLE<<endl;
 #else
  cout << "Disabling is not supported"<<endl;
 #endif
cout << "\033[0m";
}
```

Output

```
[root@localhost usp2]# c++ -Wno-deprecated s2.cpp
[root@localhost usp2]# ./a.out
Job control is supported
Saved Ids is supported
Change of ownership is supported and value is 1
No truncation is supported and value is 1
Disabling is supported and disabling char is
[root@localhost usp2]#
```

3. Consider the last 100 bytes as a region. Write a C/C++ program to check whether the region is locked or not. If the region is locked, print pid of the process which has locked. If the region is not locked, lock the region with an exclusive lock, read the last 50 bytes and unlock the region.

Explanation

- Multiple processes performs read and write operation on the same file concurrently.
- This provides a means for data sharing among processes, but it also renders difficulty for any process in determining when the other process can override data in a file.
- So, in order to overcome this drawback UNIX and POSIX standard support file locking mechanism.
- File locking is applicable for regular files.

UNIX systems provide fcntl function to support file locking. By using fcntl it is possible to impose read or write locks on either a region or an entire file

The prototype of fcntl is:

```
#include<fcntl.h>
int fcntl(int fdesc, int cmd_flag, ....);
```

- The first argument specifies the file descriptor for a file to be processed.
- The second argument cmd_flag specifies what operation has to be performed.
- The possible *cmd_flag* values are defined in the <fcntl.h> header. The specific values for file locking and their uses are:

cmd_flag	**Use**
F_SETLK	sets a file lock, do not block if this cannot succeed immediately.
F_SETLKW	sets a file lock and blocks the process until the lock is acquired.
F_GETLK	queries as to which process locked a specified region of file.

- For file locking purpose, the third argument to fctnl is an address of a *struct flock* type variable.

- This variable specifies a region of a file where lock is to be set, unset or queried. The *struct flock* is declared in the <fcntl.h> as:

```
struct flock
{
        short l_type;       /* what lock to be set or to unlock file */
        short l_whence;     /* Reference address for the next field */
        off_t l_start ;     /*offset from the l_whence reference addr*/
        off_t l_len ;       /*how many bytes in the locked region */
        pid_t l_pid ;       /*pid of a process which has locked the file */
};
```

- The l_type field specifies the lock type to be set or unset.
- The possible values, which are defined in the <fcntl.h> header, and their uses are

***l_type* value**	**Use**
F_RDLCK	Set a read lock on a specified region
F_WRLCK	Set a write lock on a specified region
F_UNLCK	Unlock a specified region

- The l_whence, l_start & l_len define a region of a file to be locked or unlocked.
- The l_whence field defines a reference address to which the l_start byte offset value is added. The possible values of *l_whence* and their uses are:

***l_whence* value**	**Use**
SEEK_CUR	The *l_start* value is added to current file pointer address
SEEK_SET	The *l_start* value is added to byte 0 of the file
SEEK_END	The *l_start* value is added to the end of the file

Algorithm

Step 1: Start
Step 2: Open a file for reading and writing
Step 3: Try to obtain an exclusive lock for the last 100 bytes of the file
Step 4: Check, if some process has put a read lock
Step 5: If yes, print the message "process.. has put a read lock", wait until process unlocks
Step 6: If no, check if some process has put a write lock
Step 7: If yes, print the message "process .. has put a write lock", wait until process unlocks
Step 8: Put an exclusive lock with waiting option for last 100 bytes
Step 9: Simulate some 10 sec delay for processing
Step 10: Read last 50 bytes of data from the file into a memory buffer
Step 11: Display the buffer
Step 12: Unlock the file
Step 13: End

Program

```
#include<fcntl.h>
#include<iostream.h>
#include<stdlib.h>
int main()
{
    char buf[50];
   int fd=open("b.txt",O_RDWR);
   struct flock f1;
   f1.l_type=F_WRLCK;
   f1.l_whence=SEEK_END;
   f1.l_start=0;
  f1.l_len=100;

   fcntl(fd,F_GETLK,&f1);
   if(f1.l_type==F_RDLCK)
   {
      cout  << f1.l_pid  << " has put read lock" << endl;
```

```
      // exit(0);
  }

   if(f1.l_type==F_WRLCK)
   {
      cout  << f1.l_pid  << " has put write lock" << endl;
      //exit(0);
  }
   f1.l_type=F_WRLCK;
   f1.l_pid=getpid();
    fcntl(fd,F_SETLKW,&f1);
   sleep(10);
   lseek(fd,-50,SEEK_END);
   read(fd,buf,50);
  cout << buf;
   f1.l_type=F_UNLCK;
   fcntl(fd,F_SETLK,&f1);

}
```

Output

```
[root@localhost usp2]# c++ -Wno-deprecated s3.cpp
[root@localhost usp2]# ./a.out &
[1] 3748
[root@localhost usp2]# ./a.out
3748 has put write lock
khkhkhkhkhkh
asfasdasdasdasdasdasdasdadasasdas
khkhkhkhkhkh
asfasdasdasdasdasdasdasdadasasdas
[1]+  Done                    ./a.out
[root@localhost usp2]#
```

4. Write a C/C++ program which demonstrates interprocess communication between a reader process and a writer process. Use mkfifo, open, read, write and close APIs in your program.

Explanation:

It is a special pipe device file which provides a temporary buffer for two or more processes to communicate by writing data to and reading data from the buffer. The size of the buffer is fixed to PIPE_BUF. Data in the buffer is accessed in a first-in-first-out manner. The buffer is allocated when the first process opens the FIFO file for read or write. The buffer is discarded when all processes close their references (stream pointers) to the FIFO file. Data stored in a FIFO buffer is temporary.

The prototype of mkfifo is:

```
#include<sys/types.h>
#include<sys/stat.h>
#include<unistd.h>
int mkfifo(const char *path_name, mode_t mode);
```

The first argument pathname is the pathname(filename) of a FIFO file to be created. The second argument mode specifies the access permission for user, group and others and as well as the S_IFIFO flag to indicate that it is a FIFO file. On success it returns 0 and on failure it returns –1.

open:

This is used to establish a connection between a process and a file i.e. it is used to open an existing file for data transfer function or else it may be also be used to create a new file. The returned value of the open system call is the file descriptor (row number of the file table), which contains the inode information.
The prototype of open function is,

```
#include<sys/types.h>
#include<sys/fcntl.h>
int open(const char *pathname, int accessmode, mode_t permission);
```

read:

The read function fetches a fixed size of block data from a file referenced by a given file descriptot
The prototype of read function is:

```
#include<sys/types.h>
#include<unistd.h>
size_t read(int fdesc, void *buf, size_t nbyte);
```

write:

The write system call is used to write data into a file. The write function puts data to a file in the form of fixed block size referred by a given file descriptor.

Algorithm:

Step 1: Start
Step 2: Create a FIFO file
Step 3: Check whether task is for reader or writer
Step 4: If reader , else go to Step 5
    Step 4.1: Open FIFO file in read mode
    Step 4.2: Load FIFO file contents into buffer
    Step 4.3: Display the buffer. Go to Step 6
Step 5: If writer
    Step 5.1: Open FIFO file in write mode
    Step 5.2: Write user data into FIFO file
Step 6: End

Program

```
#include<sys/types.h>
#include<sys/stat.h>
#include<fcntl.h>
#include<iostream.h>
int main(int argc,char *argv[])
{
    char buf[100];
   mkfifo(argv[1],S_IFIFO|0777);
  if(argc==3)
  {
      int fd=open(argv[1],O_WRONLY);
      write(fd,argv[2],strlen(argv[2]));
      close(fd);
  }
  if(argc==2)
  {
     int fd=open(argv[1],O_RDONLY);
     int n=read(fd,buf,sizeof(buf));
     buf[n]='\0';
     cout << buf << endl;
    close(fd);

 }

}
```

Output : (Case -1: Writer background, Reader foreground
Case-2: Reader background, Writer foreground)

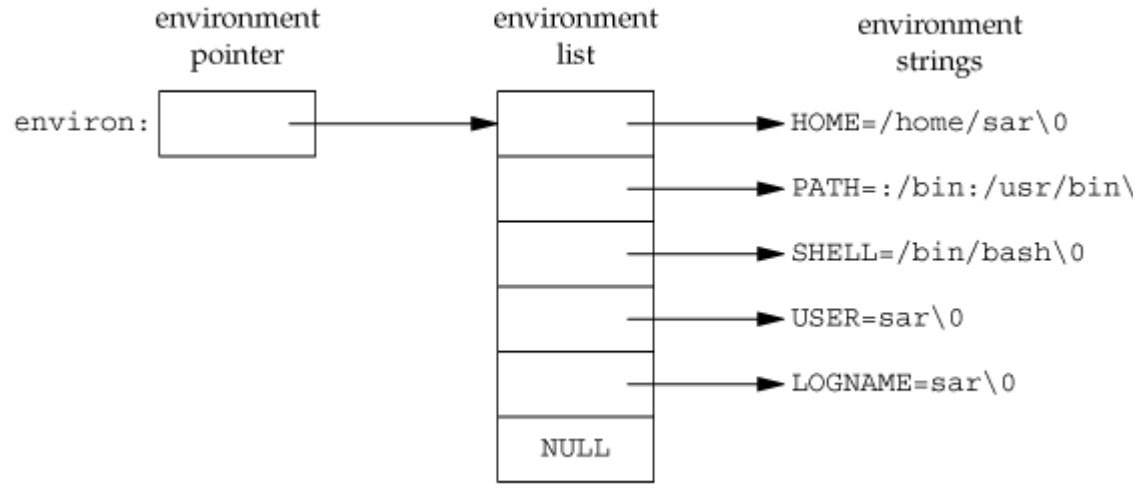
```
[root@localhost usp2]# c++ -Wno-deprecated s4.cpp
[root@localhost usp2]# ./a.out fifo1 "hi how r u" &
[1] 3777
[root@localhost usp2]# ./a.out fifo1
hi how r u
[1]+  Done                    ./a.out fifo1 "hi how r u"
[root@localhost usp2]# ./a.out fifo1 &
[1] 3779
[root@localhost usp2]# ./a.out fifo1 "hi how r u"
hi how r u
[root@localhost usp2]#
```

5. a) Write a C/C++ program that outputs the contents of its Environment list.

Explanation:

Each program is also passed an environment list. Like the argument list, the environment list is an array of character pointers, with each pointer containing the address of a null-terminated C string. The address of the array of pointers is contained in the global variable environ:

extern char **environ;



By convention the environment consists of *name=value* strings as shown in above figure. Historically, most Unix systems have provided a third argument to the main function that is the address of the environment list:

int main(int argc, char * argv[ ], char *envp[ ]);

Since ANSI C specifies that the main function be written with two arguments, and since this third argument provides no benefit over the global variable environ, POSIX.1 specifies that environ should be used instead of the third argument. Access to specific environment variables is normally through the getenv and putenv functions instead of through the environ variable.

The environment strings are usually of the form: ***name=value.*** The UNIX kernel never looks at these strings; their interpretation is up to the various applications. The shells, for example, use numerous environment variables. Some, such as HOME and USER, are set automatically at login, and others are for us to set.

Algorithm: Main

Step 1: Start
Step 2: Create an environment list (name=value format)
Step 3: Execute a program that can display all environment variables
Step 4: End

Algorithm: Environment variables display

Step 1: Start
Step 2: Loop over global variable "environ" or third argument in main function
Step 3: Display each environment variable name=value format
Step 4: End

Program:Main

```
#include<unistd.h>
#include<sys/types.h>
#include<sys/wait.h>
int main()
{
   pid_t pid;
   pid=fork();

  char *e[]={"name=chetan","key=jnnce",NULL};
  if(pid==0)
  {
     execve("/root/usp/s51",NULL,e);
  }
  else
  {
     wait(NULL);
  }
}
```

Program: Display environment list

```
#include<iostream.h>
extern char **environ;
int main(int argc,char *argv[])
{
   int i=0;
  cout.flush();

  i=0;
  while(environ[i]!=NULL)
  {
    cout << environ[i] << ",";
    i++;
  }
}
```

Output:

```
[root@localhost usp]# c++ -Wno-deprecated s51.cpp -o s51
[root@localhost usp]# c++ -Wno-deprecated s51m.cpp
[root@localhost usp]# ./a.out
name=chetan,key=jnnce,
[root@localhost usp]# ./s51
```

```
NCTUNSHOME=/usr/local/nctuns,SSH_AGENT_PID=3190,HOSTNAME=localhost.localc
ESKTOP_STARTUP_ID=,SHELL=/bin/bash,TERM=xterm,HISTSIZE=1000,GTK_RC_FILES=
k/gtkrc:/root/.gtkrc-1.2-gnome2,WINDOWID=35651665,QTDIR=/usr/lib/qt-3.3,L
t,LD_LIBRARY_PATH=/usr/local/nctuns/lib,LS_COLORS=no=00:fi=00:di=00;34:lr
pi=40;33:so=00;35:bd=40;33;01:cd=40;33;01:or=01;05;37;41:mi=01;05;37;41:e
:*.cmd=00;32:*.exe=00;32:*.com=00;32:*.btm=00;32:*.bat=00;32:*.sh=00;32:*
;32:*.tar=00;31:*.tgz=00;31:*.arj=00;31:*.taz=00;31:*.lzh=00;31:*.zip=00;
00;31:*.Z=00;31:*.gz=00;31:*.bz2=00;31:*.bz=00;31:*.tz=00;31:*.rpm=00;31:
00;31:*.jpg=00;35:*.gif=00;35:*.bmp=00;35:*.xbm=00;35:*.xpm=00;35:*.png=0
tif=00;35:,GNOME_KEYRING_SOCKET=/tmp/keyring-9TFTIo/socket,SSH_AUTH_SOCK=
h-GjIEtH3189/agent.3189,KDEDIR=/usr,SESSION_MANAGER=local/localhost.local
/tmp/.ICE-unix/3162,MAIL=/var/spool/mail/root,DESKTOP_SESSION=default,PAT
kerberos/sbin:/usr/kerberos/bin:/usr/local/sbin:/usr/local/bin:/sbin:/bir
bin:/usr/bin:/usr/X11R6/bin:/root/bin,INPUTRC=/etc/inputrc,PWD=/root/usp,
_US.UTF-8,GDMSESSION=default,SSH_ASKPASS=/usr/libexec/openssh/gnome-ssh-a
HOME=/root,SHLVL=2,GNOME_DESKTOP_SESSION_ID=Default,LOGNAME=root,DBUS_SES
S_ADDRESS=unix:abstract=/tmp/dbus-bIuokugDwH,LESSOPEN=|/usr/bin/lesspipe.
ISPLAY=:0.0,G_BROKEN_FILENAMES=1,COLORTERM=gnome-terminal,XAUTHORITY=/roc
hority,_=./s51,OLDPWD=/root,[root@localhost usp]#
```

5b) Write a C / C++ program to emulate the unix ln command.

Explanation:

Hard and Symbolic Links

link:

- The link function creates a new hard link for the existing file.

The prototype of the link function is

```
#include <unistd.h>
int link(const char *cur_link, const char *new_link);
```

- If successful, the link function returns 0. If unsuccessful, link returns –1.
- The first argument cur_link, is the pathname of existing file.
- The second argument new_link is a new pathname to be assigned to the same file.
- If this call succeeds, the hard link count will be increased by 1.
- The UNIX ln command is implemented using the link API.

Symlink:

- A symbolic link is created with the symlink function.

The prototype of the symlink function is

```
#include <unistd.h>
int symlink(const char *actualpath , const char *sympath);
```

- If successful, the link function returns 0. If unsuccessful, link returns –1.
- A new directory entry, sympathy, is created that points to actualpath.
- It is not required that actualpath exists when the symboliclink is created.
- Actualpath and sympath need not reside in the same file system.

Algorithm

Step 1: Start
Step 2: Check if user has asked for symbolic link file
Step 3: If yes,
Step 3.1 validate source and destination files
Step 3.2 : Create symbolic link file. Go to Step 5
Step 4: If no,
Step 4.1 validate source and destination files
Step 4.2 : Create hard link file.
Step 5: End

Program:

```
#include<unistd.h>
#include<iostream.h>
#include<string.h>
int main(int argc,char *argv[])
{
   if(argc<3 || argc>4)
   {
       cout << "Error in usage"<<endl;
      return(-1);
  }
  if(argc==4 && strcmp(argv[1],"-s")!=0)
  {
      cout << "for symbolic link use -s option" << endl;
      return(-1);
  }

  if(argc==4 && access(argv[2],F_OK)==-1)
  {
      cout << "source file does not exist" << endl;
      return(-1);
  }


  if(argc==3  && access(argv[1],F_OK)==-1)
  {
      cout << "source file does not exist" << endl;
      return(-1);
  }

  if(argc==4  && access(argv[3],F_OK)!=-1)
  {
      cout << "destination file already exists" << endl;
      return(-1);
  }

  if(argc==3  && access(argv[2],F_OK)!=-1)
  {
      cout << "destination file already exists" << endl;
      return(-1);
  }
  if(argc==4)
  {
     symlink(argv[2],argv[3]);
      return(0);
   }
  if(argc==3)
  {
      link(argv[1],argv[2]);
      return(0);
  }
}
```

Output:

```
[root@localhost usp]# c++ -Wno-deprecated s5b.cpp
[root@localhost usp]# ./a.out
Error in usage
[root@localhost usp]# ./a.out file1 file2
source file does not exist
[root@localhost usp]# cat >file1
jnnce
[root@localhost usp]# ./a.out file1 file2
[root@localhost usp]# ls -l file1 file2
-rw-r--r--  2 root root 6 Apr 27 05:09 file1
-rw-r--r--  2 root root 6 Apr 27 05:09 file2
[root@localhost usp]# ./a.out -s file1 file2
destination file already exists
[root@localhost usp]# ./a.out -s file1 file3
[root@localhost usp]# ls -l file1 file3
-rw-r--r--  2 root root 6 Apr 27 05:09 file1
lrwxrwxrwx  1 root root 5 Apr 27 05:10 file3 -> file1
[root@localhost usp]#
```

6 Write a C/C++ program to illustrate the race condition.

Explanation:

A race condition occurs when multiple processes are trying to do something with shared data and the final outcome depends on the order in which the processes run. It is also defined as; an execution ordering of concurrent flows that results in undesired behavior is called a race condition-a software defect and frequent source of vulnerabilities.

Race condition is possible in runtime environments, including operating systems that must control access to shared resources, especially through process scheduling.

Algorithm:

Step 1: Start
Step 2: Initialize two messages, one each for parent and child process
Step 3: Create a process
Step 4: If child, print child message character by character. Go to Step 6
Step 5: If parent, print parent message character by character.
Step 6: End

Program:

```
#include<unistd.h>
#include<sys/types.h>
#include<iostream.h>
void cat(char*);
int main()
{
   char *p="jnnce shimoga vtu future of excellence qqqqqqqq";
   char *c="computer science and engineering- bringing up excellence
rrrrrrrr";

   pid_t pid;
   pid=fork();
  if(pid==0)
  {
       cat(c);
      exit(0);
   }
   else
  {

     cat(p);
  }
}
void cat(char *str)
{
   cout.flush();
   for(int i=0;i<strlen(str);i++)
    {
        cout.flush();
```

```
        cout << str[i];
    }
}
```

Output:

```
[root@localhost new2]# c++ -Wno-deprecated s6.cpp
[root@localhost new2]# ./a.out
computer science and engineering- bringing up excellence rrrrrrrrjnnce sh
tu future of excellence qqqqqqqq[root@localhost new2]# ./a.out
computer science and engineering- bringing up excejnnce shimoga vtu futur
cellence qqqqqqqq[root@localhost new2]# llence rrrrrrrr
```

7. Write a C/C++ program that creates a zombie and then calls system to execute the ps command to verify that the process is zombie.

Explanation:

A zombie process or defunct process is a process that has completed execution but still has an entry in the process table. This entry is still needed to allow the parent process to read its child's exit status. The term zombie process derives from the common definition of zombie - an undead person.

Algorithm:

Step 1: Start
Step 2: Create a process
Step 3: If process is child, exit and go to Step 6
Step 4: If process is parent, display process status
Step 5: Illustrate zombie state
Step 6: End

Program:

```
#include<unistd.h>
#include<sys/types.h>
#include<iostream.h>

int main()
{

   pid_t pid;
   pid=fork();
  if(pid==0)
 {
    cout << "child" <<endl;
   exit(0);
 }
 else
 {
   char str[100],b[10];

   //system("ps -o stat,pid,ppid,command");
   strcpy(str,"ps -o stat,pid,ppid,command --pid ");
   sprintf(b,"%d",pid);
   strcat(str,b);
   system(str);
   sleep(10);
  cout << "parent";

}
}
```

Output:

```
[root@localhost usp2]# c++ -Wno-deprecated s7.cpp
[root@localhost usp2]# ./a.out
child
STAT   PID  PPID COMMAND
Z+    3546  3545 [a.out] <defunct>
parent[root@localhost usp2]#
```

8. Write a C/C++ program to avoid zombie process by forking twice.

Explanation

When a process terminates, either normally or abnormally, the kernel notifies the parent by sending the SIGCHLD signal to the parent. Because the termination of a child is an asynchronous event - it can happen at any time while the parent is running - this signal is the asynchronous notification from the kernel to the parent. The parent can choose to ignore this signal, or it can provide a function that is called when the signal occurs: a signal handler. A process that calls wait or waitpid can:

- Block, if all of its children are still running
- Return immediately with the termination status of a child, if a child has terminated and is waiting for its termination status to be fetched

Return immediately with an error, if it doesn't have any child processes.

Algorithm:

Step 1: Start
Step 2: Create a process
Step 3: If child, create another process Go to Step 5
Step 4: If parent, Wait for the death of the child. Go to Step 8
Step 5: If parent, exit and go to Step 8
Step 6: If child, Sleep for 20 secs
Step 7: print the process status (zombie not created)
Step 8: End

Program:

```
#include<unistd.h>
#include<sys/types.h>
#include<iostream.h>
#include<sys/wait.h>
int main()
{
   pid_t cid,gcid;
   cid=fork();
   if(cid==0)
   {
     cout << "child id " <<getpid()<<endl;
     gcid=fork();
     if(gcid==0)
     {
       cout << "grandchild " <<getpid()<<endl;
        sleep(20);
       system("ps -o stat,pid,ppid,command");
     }
     else
     {
       exit(0);
    }
   }
  else
  {
```

```
    cout << "parent " <<getpid()<<endl;
    wait(NULL);
  }

}
```

Output:

```
[root@localhost usp2]# c++ -Wno-deprecated s8.cpp
[root@localhost usp2]# ./a.out
child id 3576
grandchild 3577
parent 3575
[root@localhost usp2]# STAT   PID  PPID COMMAND
Ss+   3313  3311 bash
S     3577     1 ./a.out
R     3578  3577 ps -o stat,pid,ppid,command

[root@localhost usp2]#
```

9. Write a C/C++ program to implement the system function.

Explanation:

When a process calls one of the exec functions, that process is completely replaced by the new program, and the new program starts executing at its main function. The process ID does not change across an exec, because a new process is not created; exec merely replaces the current process - its text, data, heap, and stack segments - with a brand new program from disk.

There are six exec functions:

```
#include <unistd.h>
int execl(const char *pathname, const char *arg0,... /* (char *)0 */ );
int execv(const char *pathname, char *const argv [ ]);
int execle(const char *pathname, const char *arg0,... /*(char *)0, char *const envp */ );
int execve(const char *pathname, char *const argv[ ], char *const envp[]);
int execlp(const char *filename, const char *arg0, ... /* (char *)0 */ );
int execvp(const char *filename, char *const argv [ ]);
```

All six return: -1 on error, no return on success.

System() function:

It is convenient to execute a command string from within a program. ANSI C defines the system function, but its operation is strongly system dependent. The system function is not defined by POSIX.1 because it is not an interface to the operating system, but really an interface to a shell.

The prototype of system function is:

```
#include <stdlib.h>
int system(const char *cmdstring);
```

It executes a command specified in string by calling /bin/sh -c string, and returns after the command has been completed. During execution of the command, SIGCHLD will be blocked, and SIGINT and SIGQUIT will be ignored.

If cmdstring is a null pointer, system returns nonzero only if a command processor is available. This feature determines whether the system function is supported on a given operating system. Under the UNIX System, system is always available. Because system is implemented by calling fork, exec, and waitpid, there are three types of return values.

- If either the fork fails or waitpid returns an error other than EINTR, system returns 1 with errno set to indicate the error.
- If the exec fails, implying that the shell can't be executed, the return value is as if the shell had executed exit(127).
- Otherwise, all three functions fork, exec, and waitpid succeed, and the return

value from system is the termination status of the shell, in the format specified for waitpid.

Algorithm:

Step 1: Start
Step 2: Register to ignore Interrupt and Terminate signals
Step 3: Create a process
Step 4: If parent, wait for the death of the child Go to Step 7
Step 5: If child, execute a shell program with command line options
Step 6: return exit status
Step 7: Print the exit status
Step 8: End

Program:

```
#include<unistd.h>
#include<sys/types.h>
#include<sys/wait.h>
#include<iostream.h>
#include<signal.h>

int system(const char*cmd);
void pr_status(int);
int main(int argc,char *argv[])
{
   int s;
   s=system(argv[1]);
  pr_status(s);
}
void pr_status(int s)
{
   if(WIFEXITED(s))
    cout << "Normal termination and status=" << WEXITSTATUS(s);
   if(WIFSIGNALED(s))
     cout  << "Abnormal termination and signal is " << WTERMSIG(s);
}
int system(const char*cmd)
{
   pid_t pid;
   int status=0;
   signal(SIGINT,SIG_IGN);
   signal(SIGTERM,SIG_IGN);

   pid=fork();
   if(pid<0)
   {
      status=-1;
   }
   if(pid==0)
   {
       execl("/bin/sh","sh","-c",cmd,(char*)0);
```

```
        _exit(127);
    }
    else
    {
          waitpid(pid,&status,0);
    }
  return(status);
}
```

Output: Case -1: Simple command without options

```
[root@localhost usp2]# c++ -Wno-deprecated s9.cpp
[root@localhost usp2]# ./a.out date
Sun Apr 24 21:18:37 IST 2016
Normal termination and status=0[root@localhost usp2]#
```

Case 2: Command with options

```
[root@localhost usp2]# ./a.out "ls -l"
-rw-r--r--  1 root root  190 Apr 24 20:58 s51.cpp
-rw-r--r--  1 root root  244 Apr 24 20:59 s51m.cpp
-rwxr-xr-x  1 root root  398 Apr 24 21:07 s7.cpp
-rwxr-xr-x  1 root root  475 Apr 24 21:10 s8.cpp
-rwxr-xr-x  1 root root  771 Apr 24 21:17 s9.cpp
-rwxr-xr-x  1 root root  693 Apr 24 21:16 system2.cpp
-rwxr-xr-x  1 root root  771 Apr 24 21:17 system3.cpp
-rwxr-xr-x  1 root root  534 Mar 23 11:29 wait1.cpp
-rwxr-xr-x  1 root root  534 Mar 23 11:29 wait2.cpo
-rwxr-xr-x  1 root root  550 Mar 23 11:38 wait2.cpp
-rwxr-xr-x  1 root root  554 Mar 23 11:55 wait3.cpp
Normal termination and status=0[root@localhost usp2]#
```

Case 3: Invalid command

```
[root@localhost usp2]# ./a.out "aaa"
sh: aaa: command not found
Normal termination and status=127[root@localhost usp2]#
```

10. Write a C/C++ program to set up a real-time clock interval timer using the alarm API.

Explanation:

Alarm() function:

The alarm API can be called by a process to request the kernel to send the SIGALRM signal after a certain number of real clock seconds. The function prototype of the API is:

```
#include<signal.h>
unsigned int alarm(unsigned int time_interval);
```

Returns: 0 or number of seconds until previously set alarm

pause() function: It waits for signal.

```
#include <unistd.h>
int pause(void);
```

The pause() library function causes the invoking process (or thread) to sleep until a signal is received that either terminates it or causes it to call a signal-catching function. The pause() function only returns when a signal was caught and the signal-catching function returned. In this case pause() returns -1, and *errno* is set to EINTR.

Sigaction() function:

The sigaction API blocks the signal it is catching allowing a process to specify additional signals to be blocked when the API is handling a signal. The sigaction API prototype is:

```
#include<signal.h>
int sigaction(int signal_num, struct sigaction* action, struct sigaction* old_action);
```

Returns: 0 if OK, 1 on error The struct sigaction data type is defined in the <signal.h> header as:

```
struct sigaction
   {
     void (*sa_handler)(int);
    sigset_t sa_mask;
     int sa_flag;
}
```

Algorithm:

Step 1: Start
Step 2: Accept starting time and interval value in seconds from user
Step 3: Register for alarm signal
Step 4: Set up alarm with starting time value
Step 5: When alarm expires, reset alarm with a diagnostic message
Step 6: End

Program:

```
#include<signal.h>
#include<iostream.h>
int m,n;
void g1(int signo)
{
   cout << "wake up" <<endl;
   alarm(n);
}
int main(int argc,char *argv[])
{

  sscanf(argv[1],"%d",&m);
 sscanf(argv[2],"%d",&n);
  struct sigaction s;
  sigemptyset(&s.sa_mask);
   s.sa_flags=0;
   s.sa_handler=g1;
   sigaction(SIGALRM,&s,0);
    alarm(m);
   while(1);
}
```

Output:

```
[root@localhost new2]#
[root@localhost new2]#
[root@localhost new2]#
[root@localhost new2]# c++  -Wno-deprecated s10.cpp
[root@localhost new2]# ./a.out 5 2
wake up
wake up
wake up
```

11. Write a C program to implement the syntax-directed definition of "if E then S1" and "if E then S1 else S2". (Refer Fig. 8.23 in the text book prescribed for 06CS62 Compiler Design, Alfred V Aho, Ravi Sethi, and Jeffrey D Ullman: Compilers- Principles, Techniques and Tools, 2nd Edition, Pearson Education, 2007).

Explanation:

Syntax Directed Definition:

A Syntax directed definition is a generalization of a context free grammar in which each grammar symbol has an associated set of attributes, partitioned into two subsets called the synthesized and inherited attributes of that grammar symbol.

An attribute can represent anything we choose: a string, a number, a type, a memory location, or whatever. The value of an attribute at a parse tree node is defined by a semantic rule associated with a production used at that node. The value of a synthesized attribute at a node is computed from the values of attributes at the children of that node in the parse tree; the value of an inherited attribute is computed from the values of attributes at the siblings and parent of that node.

Semantic rules set up dependencies between attributes that will be represented by a graph. From the dependency graph, we derive an evaluation order for the semantic rules. Evaluation of the semantic rules defines the values of the attributes at the nodes in parse tree for the input string.

A parse tree showing the values of attributes at each node is called an annotated parse tree. The process of computing the attributes at the nodes is called annotating or decorating the parse tree.

SDD for simple if and if-else

Simple if

S → if E then S1
E.true := new_label()
E.false :=S.next
S1.next :=S.next
S.code :=E.code | | gen_code(E.true „: „) | | S1.code

If-else
S → if E then S1 else S2
E.true := new_label()
E.false := new_label()
S1.next :=S.next
S2.next :=S.next
S.code :=E.code | | gen_code(E.true „: „) |S1.code | gen_code("goto",S.nex)

|| gen_code(E.false,":") || S2.code

Program (Basic Version)

```
#include<stdio.h>
#include<string.h>
int main()
{
   char str[100],expr[100],tstmt[100],fstmt[100];
   printf("enter the if statement\n");
```

```
   scanf("%[^\n]s",str);
   if(strstr(str,"if")!=NULL  && strstr(str,"else")==NULL)
   {
      sscanf(str,"if %s then %s",expr,tstmt);
      printf("100: if %s then goto 102\n",expr);
      printf("101:goto 103\n");
      printf("102:%s\n",tstmt);
      printf("103:....\n");

  }
  else if(strstr(str,"if")!=NULL && strstr(str,"else")!=NULL)
  {
      sscanf(str,"if %s then %s else %s",expr,tstmt,fstmt);
      printf("100: if %s then goto 102\n",expr);
      printf("101:%s, goto 103\n",fstmt);
      printf("102:%s\n",tstmt);
      printf("103:....\n");
  }
}
```

Output:

```
[root@localhost new2]# c++ -Wno-deprecated s111.c
[root@localhost new2]# ./a.out
enter the if statement
if E then S1
100: if E then goto 102
101:goto 103
102:S1
103:....
[root@localhost new2]# ./a.out
enter the if statement
if E then S1 else S2
100: if E then goto 102
101:S2, goto 103
102:S1
103:....
[root@localhost new2]# ./a.out
enter the if statement
if ( E ) then S1
100: if ( then goto 102
101:goto 103
102:
103:....
[root@localhost new2]#
```

Program: Complete version

```
#include<stdio.h>
#include<string.h>
int position_if(char *str);
int position_then(char *str);
int position_else(char *str);
```

```
void str_bet(char *,int,int,char *);

int main()
{
   char str[100],expr[100],tstmt[100],fstmt[100];
   printf("enter the if statement\n");
   scanf("%[^\n]s",str);
   if(strstr(str,"if")!=NULL  && strstr(str,"else")==NULL)
   {
      int p1=position_if(str);
      int p2=position_then(str);
      str_bet(str,p1,p2-5,expr);
      str_bet(str,p2,strlen(str)-1,tstmt);
      printf("100: if %s then goto 102\n",expr);
      printf("101:goto 103\n");
      printf("102:%s\n",tstmt);
      printf("103:....\n");

  }
  else if(strstr(str,"if")!=NULL && strstr(str,"else")!=NULL)
  {
      int p1=position_if(str);
      int p2=position_then(str);
      int p3=position_else(str);
      str_bet(str,p1,p2-5,expr);
      str_bet(str,p2,p3-5,tstmt);
      str_bet(str,p3,strlen(str)-1,fstmt);
      printf("100: if %s then goto 102\n",expr);
      printf("101:%s, goto 103\n",fstmt);
      printf("102:%s\n",tstmt);
      printf("103:....\n");
  }
}
int position_if(char *str)
{
  int i=0;
  for(i=0;i<strlen(str);i++)
  {
      if(str[i]=='i' && str[i+1]=='f')
          return(i+2);

  }

  return(-1);
}
```

```
int position_then(char *str)
{
  int i=0;
  for(i=0;i<strlen(str);i++)
  {
      if(str[i]=='t' && str[i+1]=='h' && str[i+2]=='e' &&
str[i+3]=='n')
          return(i+4);
```

```
  }

  return(-1);
}


int position_else(char *str)
{
  int i=0;
  for(i=0;i<strlen(str);i++)
  {
      if(str[i]=='e' && str[i+1]=='l' && str[i+2]=='s' &&
str[i+3]=='e')
          return(i+4);

  }

  return(-1);
}

void str_bet(char *str,int p1,int p2,char *temp)
{
    int j=0,i;
    for(i=p1;i<=p2;i++)
    {
        temp[j++]=str[i];
    }
   temp[j]='\0';
}
```

Output:

```
[root@localhost new2]# c++ -Wno-deprecated s112.c
[root@localhost new2]# ./a.out
enter the if statement
ifEthenS1elseS2
100: if E then goto 102
101:S2, goto 103
102:S1
103:....
[root@localhost new2]# ./a.out
enter the if statement
if ( a > b ) then a = a+ 1; else a = a - 1;
100: if  ( a > b )  then goto 102
101: a = a - 1;, goto 103
102: a = a+ 1;
103:....
[root@localhost new2]#
```

12. Write a yacc program that accepts a regular expression as input and produce its parse tree as output.

Explanation:

Regular Expression:

A regular expression is a specific pattern that provides concise and flexible means to "match" (specify and recognize) strings of text, such as particular characters, words, or patterns of characters. A regular expression provides a grammar for a formal language; this specification can be interpreted by a regular expression processor, which is a program that either serves as a parser generator or examines text and identifies substrings that are members of the specified (again, formal) language.

Derivations:

To check whether a sequence of tokens is legal or not, we start with a nonterminal called the start symbol. We apply productions, rewriting nonterminals, until only terminals remain. A derivation replaces a nonterminal on LHS of a production with RHS.

Leftmost and Rightmost Derivations:

When deriving a sequence of tokens. More than one nonterminal may be present and can be expanded. A leftmost derivation chooses the leftmost nonterminal to expand. A rightmost derivation chooses the rightmost nonterminal to expand.

Parse Tree:

Is a graphical representation for a derivation. It filters out choice regarding replacement order. It is rooted by the start symbol S, interior nodes represent nonterminals in N, leaf nodes are terminals in T or node A can have children X1, X2,...Xn if a rule A->X1, X2... Xn exists.

Algorithm:

```
Step 1: Start
Step 2: Scan a regular expression character by character
Step 3: For each character, return token ALPHABET if it is alphabet,
else return value
Step 4: Write a grammar that recursively defines regular
expressions:
        re-> re.re |re|re | re+ | re* | ALPHABET
Step 5: For each rule, store corresponding pattern in a production
array
Step 6: Start traversing production from end
Step 7: Print last element as it is
Step 8: For each remaining elements, replace rightmost regular
expression in previous value printed by current value in production
array
Step 9: End
```

Program : (RMD)

```
%{
   #include<stdio.h>
   #include<string.h>
    char prod[10][200],temp[200],temp1[100];
   int cnt=0,i,j;
%}
%token ALPHABET
%left '.'
%left '|'
%nonassoc '+' '*'
%%
S:re '\n'{
            for(i=cnt-1;i>=0;i--)
            {
               if(i==cnt-1)
               {
                     printf("%s\n",prod[i]);
                     strcpy(temp,prod[i]);
               }
               else
               {
                  j=rmo(temp);
                  temp[j]='\0';
                  sprintf(temp1,"%s%s%s",temp,prod[i],temp+j+2);
                  printf("%s\n",temp1);
                  strcpy(temp,temp1);
               }
            }
            exit(0);
         }
re:re'|'re {strcpy(prod[cnt++],"re|re");}
  |re'.'re {strcpy(prod[cnt++],"re.re");}
  |re'*' {strcpy(prod[cnt++],"re*");}
  |re'+' {strcpy(prod[cnt++],"re+");}
  |'('re')' {strcpy(prod[cnt++],"(re)");}
  | ALPHABET {prod[cnt][0]=yylval;prod[cnt++][1]='\0';}
%%
main()
{
  yyparse();
}

int yylex()
{
  int ch;
  ch=getchar();
  yylval=ch;
  if(isalpha(ch))
     return ALPHABET;
  return ch;
}

int yyerror()
```

```
{
   printf("invalid expr");

}
int rmo(char *str)
{
  int i;
  for(i=strlen(str)-1;i>=0;i--)
  {
    if(str[i]=='e' && str[i-1]=='r')
      return(i-1);
  }
}
```

Output:

```
[root@localhost usp2]# yacc -d s12
s121.y  s122.y
[root@localhost usp2]# yacc -d s121.y
[root@localhost usp2]# cc y.tab.c
[root@localhost usp2]# ./a.out
a+.b*
re.re
re.re*
re.b*
re+.b*
a+.b*
[root@localhost usp2]# ./a.out
aa
invalid expr[root@localhost usp2]# ./a.out
(p+)
(re)
(re+)
(p+)
[root@localhost usp2]#
```

Program (LMD – extra)

```
%{
   #include<stdio.h>
   #include<string.h>
    char prod[10][200],temp[200],temp1[100],buf[100],bufr[100];
   int cnt=0,i,j,cnti=0;
%}
%token ALPHABET
%left '.'
%left '|'
%nonassoc '+' '*'
%%
S:re '\n'{
```

```
            for(i=cnt-1;i>=0;i--)
            {
               if(i==cnt-1)
               {
                     printf("%s\n",prod[i]);
                     strcpy(temp,prod[i]);
               }
               else
               {
                  j=lmo(temp);
                  temp[j]='\0';
                  sprintf(temp1,"%s%s%s",temp,prod[i],temp+j+2);
                  printf("%s\n",temp1);
                  strcpy(temp,temp1);
               }
            }
         }
re:re'|'re {strcpy(prod[cnt++],"re|re");}
  |re'.'re {strcpy(prod[cnt++],"re.re");}
  |'*'re {strcpy(prod[cnt++],"re*");}
  |'+'re {strcpy(prod[cnt++],"re+");}
  |')'re'(' {strcpy(prod[cnt++],"(re)");}
  | ALPHABET {prod[cnt][0]=yylval;prod[cnt++][1]='\0';}
%%
main()
{
  scanf("%[^\n]s",buf);
  for(i=strlen(buf)-1,j=0;i>=0;i--,j++)
     bufr[j]=buf[i];
  bufr[j]='\n';
   yyparse();
}

int yylex()
{
  int ch;
  ch=bufr[cnti++];
  //printf("a=%c",ch);
  yylval=ch;
  if(isalpha(ch))
     return ALPHABET;
  return ch;
}

int yyerror()
{
   printf("Invalid expr");
}
int lmo(char *str)
{
  int i;
  for(i=0;i<=strlen(str)-1;i++)
  {
    if(str[i]=='r' && str[i+1]=='e')
      return(i);
  }
```

```
}
```

Output:

```
[root@localhost usp2]# yacc -d s122.y
[root@localhost usp2]# cc y.tab.c
[root@localhost usp2]# ./a.out
a+.b*
re.re
re+.re
a+.re
a+.re*
a+.b*
[root@localhost usp2]# ./a.out
aa
Invalid expr[root@localhost usp2]# ./a.out
(a+)
(re)
(re+)
(a+)
[root@localhost usp2]#
```

Extra programs: (Other than syllabus)

1. Program to illustrate CPP macros

```
#include<iostream.h>
int main()
{
   #if (__STDC__==0)
       cout << "Cpp macros not supported";
  #else
     cout << __FILE__ << endl;
     cout << __DATE__ << endl;
     cout << __TIME__ << endl;
     cout << __LINE__ << endl;
  #endif
}
```

2. Program to illustrate POSIX version

```
#define  _POSIX_SOURCE
#define _POSIX_C_SOURCE 199309L
#include<unistd.h>
#include<iostream.h>
int main()
{
  #ifdef _POSIX_VERSION
    cout << "POSIX version is " << _POSIX_VERSION << endl;
 #else
    cout << "POSIX not defined " ;
  #endif
}
```

3. Program to illustrate open flags of file handling

```
#include<sys/types.h>
#include<fcntl.h>
#include<iostream.h>
int main()
{
  int
fd=open("b.txt",O_CREAT|O_EXCL|O_WRONLY,S_IRWXU|S_IRGRP|S_IROTH);
  int n= write(fd,"cse-dept",8);
  cout << fd;

}
```

4. Program to illustrate read-write operation of file

```
#include<sys/types.h>
#include<fcntl.h>
#include<iostream.h>
int main()
{
  char *buf;
  buf=(char*)malloc(sizeof(buf));
  int fd=open("a.txt",O_WRONLY);
  int n= write(fd,"cse-dept",8);
```

```
  close(fd);
  fd=open("a.txt",O_RDONLY);
  int m=read(fd,buf,n);
   close(fd);
  cout << buf;

}
```

5. Program to demonstrate file APIs with structures

```
#include<sys/types.h>
#include<fcntl.h>
#include<iostream.h>
struct book
{
   char name[30];
   int cost;
};
int main()
{
  struct book b1,b2;
  strcpy(b1.name,"chachi");
  b1.cost=420;

  int fd=open("a.txt",O_WRONLY);
  int n= write(fd,&b1,sizeof(b1));
  close(fd);
  fd=open("a.txt",O_RDONLY);
  int m=read(fd,&b2,n);
   close(fd);
  cout << b2.name << b2.cost;

}
```

6. Program to illustrate lseek operations

```
#include<sys/types.h>
#include<fcntl.h>
#include<iostream.h>
int main()
{
  int fd=open("a.txt",O_WRONLY);
  lseek(fd,-3,SEEK_END);
  write(fd,"cas",3);
  lseek(fd,0,SEEK_SET);
 int n=lseek(fd,0,SEEK_END);
  cout << n;
  close(fd);
}
```

7. Program to delete a file

```
#include<sys/types.h>
#include<fcntl.h>
#include<iostream.h>
int main(int argc,char *argv[])
```

```
{
   if(argc!=2)
      cout << "error in usage";
   else
    unlink(argv[1]);
}
```

8. Program to change access permissions

```
#include<sys/stat.h>
#include<sys/types.h>
#include<unistd.h>

int main()
{
   struct stat s1;
   stat("a.c",&s1);
   int rperm=S_IRWXG|S_IXOTH;
   int aperm=s1.st_mode &~ rperm;
   chmod("a.c",aperm);

}
```

9. Program to count number of files in a directory

```
#include<sys/types.h>
#include<sys/stat.h>
#include<unistd.h>
#include<dirent.h>
#include<iostream.h>
int main(int argc,char *argv[])
{
  DIR *dp;
  struct dirent *dr;
  int cnt=0;
    if(!(dp=opendir(argv[1])))
    {
        mkdir(argv[1],0777);
    }
    else
    {
        while(dr=readdir(dp))
        {
               cout << dr->d_name << endl;
            cnt++;
       }
      cout << "No of files is " << (cnt-2);
    }

   }
```

10. Program to illustrate fork() API

```
#include<unistd.h>
#include<sys/types.h>
```

```
#include<iostream.h>
int g=2;
int main()
{
   int s=3;
   pid_t pid;
   pid=fork();
  if(pid==0)
 {
    g++;
    s++;
 }
 cout << "g="<<g <<",s="<<s<<endl;
}
```

11. Program to illustrate vfork API

```
#include<unistd.h>
#include<sys/types.h>
#include<iostream.h>
int g=2;
int main()
{
   int s=3;
   pid_t pid;
   pid=vfork();
  if(pid==0)
 {
    g++;
    s++;
   exit(0);
 }
 cout << "g="<<g <<",s="<<s<<endl;
}
```

12. Program to print wait status

```
#include<unistd.h>
#include<sys/types.h>
#include<iostream.h>
#include<sys/wait.h>
void print_status(int);
int main()
{
   pid_t cid,gcid;
   cid=fork();
   if(cid==0)
   {
     cout << "child id " <<getpid()<<endl;
     gcid=fork();
     if(gcid==0)
     {
       cout << "grandchild " <<getpid()<<endl;
        sleep(6);
       system("ps -o stat,pid,ppid,command");
     }
     else
     {
```

```
        //exit(0);
        abort();
     }
    }
  else
  {
    cout << "parent " <<getpid()<<endl;
    int s;
    wait(&s);
    print_status(s);
  }

}
void print_status(int status)
{
   if(WIFEXITED(status))
     cout << "normal termination" <<endl;
  else if(WIFSIGNALED(status))
   cout << "abnormal termination" <<endl;
  else if(WIFSTOPPED(status))
  cout << "stopped " <<endl;
}
```

13. Program to illustrate wait API

```
#include<unistd.h>
#include<sys/types.h>
#include<iostream.h>
#include<sys/wait.h>
int main()
{
   pid_t pid1,pid2,pid3;
   pid1=fork();
   if(pid1==0)
   {
     sleep(9);
     exit(0);
   }
  pid2=fork();
  if(pid2==0)
  {
    sleep(7);
    exit(0);
  }
  pid3=fork();
  if(pid3==0)
  {
     sleep(10);
     exit(0);
  }
  else
  {
     cout << "parent id =" << getpid()<<endl;
    cout <<
"child="<<pid1<<",child2="<<pid2<<",child3="<<pid3<<endl;
   pid_t pid= wait(NULL);
```

```
    cout << "the child died is " <<pid <<endl;

  }
```

14. Program to illustrate the use of waitpid API

```
#include<unistd.h>
#include<sys/types.h>
#include<iostream.h>
#include<sys/wait.h>
int main()
{
   pid_t pid1,pid2,pid3;
   pid1=fork();
   if(pid1==0)
   {
     sleep(9);
     exit(0);
   }
  pid2=fork();
  if(pid2==0)
  {
    sleep(7);
    exit(0);
  }
  pid3=fork();
  if(pid3==0)
  {
     sleep(10);
     exit(0);
  }
  else
  {
     cout << "parent id =" << getpid()<<endl;
    cout <<
"child="<<pid1<<",child2="<<pid2<<",child3="<<pid3<<endl;
   pid_t pid= waitpid(pid1,NULL,WNOHANG);
    cout << "the child died is " <<pid <<endl;

  }
}
```

15. Program to demonstrate the use of waitid API

```
#include<unistd.h>
#include<sys/types.h>
#include<iostream.h>
#include<sys/wait.h>
int main()
{
   pid_t pid1,pid2,pid3;
   pid1=fork();
   if(pid1==0)
   {
     sleep(9);
     exit(0);
   }
  pid2=fork();
```

```
  if(pid2==0)
  {
    sleep(7);
    exit(0);
  }
  pid3=fork();
  if(pid3==0)
  {
     sleep(10);
     exit(0);
  }
  else
  {
     cout << "parent id =" << getpid()<<endl;
    cout <<
"child="<<pid1<<",child2="<<pid2<<",child3="<<pid3<<endl;
   pid_t pid= waitid(P_PID,pid1,NULL,WEXITED);
   cout << "the child died is " <<pid <<endl;

  }

}
```

16. Program to demonstrate the use of itimer APIs

```
#include<signal.h>
#include<iostream.h>
#include<sys/time.h>
int m,n;
void g1(int signo)
{
   cout << "wake up" <<endl;
   alarm(n);
}
int main(int argc,char *argv[])
{

  sscanf(argv[1],"%d",&m);
 sscanf(argv[2],"%d",&n);
  struct itimerval i;
  i.it_interval.tv_sec=n;
  i.it_interval.tv_usec=0;
  i.it_value.tv_sec=m;
  i.it_value.tv_usec=0;

  struct sigaction s;
  sigemptyset(&s.sa_mask);
   s.sa_flags=0;
   s.sa_handler=g1;
   sigaction(SIGALRM,&s,0);
   setitimer(ITIMER_REAL,&i,0);

   while(1);
}
```

17. Program to avoid race condition by scheduling parent first, child next

```
#include<unistd.h>
#include<sys/types.h>
#include<iostream.h>
#include<sys/wait.h>
void cat(char*);
static void  WAIT_PARENT();
static void TELL_CHILD();
static int s=1;
int main()
{
   char *p="jnnce future of excellence";
   char *c="computer science - bringing up excellence";

   pid_t pid;
   pid=fork();
  if(pid==0)
  {
   WAIT_PARENT();
      cat(c);

   }
   else
  {
       cat(p);
       TELL_CHILD();

  }
}
void cat(char *str)
{
   for(int i=0;i<strlen(str);i++)
    {
        cout.flush();
        cout << str[i];
   }
}

 void WAIT_PARENT()
{
    while(getppid()!=1);
}
void TELL_CHILD()
{
    exit(0);
}
```

18. Program to avoid race condition by scheduling child first and then parent

```
#include<unistd.h>
#include<sys/types.h>
#include<iostream.h>
#include<sys/wait.h>
void cat(char*);
int main()
{
```

```
   char *p="jnnce future of excellence";
   char *c="computer science - bringing up excellence";

   pid_t pid;
   pid=fork();
  if(pid==0)
  {
     wait(NULL);
       cat(c);
      exit(0);
   }
   else
  {

     cat(p);
  }
}
void cat(char *str)
{
   for(int i=0;i<strlen(str);i++)
    {
        cout.flush();
        cout << str[i];
   }
}
```

19. Program to emulate ls -l

```
     #include<iostream.h>
#include<sys/stat.h>
#include<sys/types.h>
#include<unistd.h>
#include<grp.h>
#include<pwd.h>

void display_file_type(struct stat);
void display_file_perm(struct stat );
void display_ug(struct stat);
int main(int argc,char *argv[])
{
   struct stat s;
   stat(argv[1],&s);
   display_file_type(s);
   display_file_perm(s);
   cout << "\t" << s.st_nlink;
   display_ug(s);
   cout << " " << s.st_size;
    cout << " " << ctime(&s.st_mtime);
   cout << "  "  <<argv[1];
}
void display_ug(struct stat s)
{
   struct group *g;
   struct passwd *p;
```

```
   p=getpwuid(s.st_uid);
  cout << "\t" << p->pw_name;
  g=getgrgid(s.st_gid);
  cout << "\t" << g->gr_name;

}
void display_file_perm(struct stat s)
{
   char dperm[]="rwxrwxrwx";
  char aperm[10];
   for(int i=0,j=(1<<8);i<9;i++,j=j>>1)
   {
     aperm[i]=(s.st_mode&j)?dperm[i]:'-';
   }
 if(s.st_mode&S_ISUID)
    aperm[2]=(aperm[2]=='x')?'s':'S';

 if(s.st_mode&S_ISGID)
    aperm[5]=(aperm[5]=='x')?'s':'S';

 if(s.st_mode&S_ISVTX)
    aperm[8]=(aperm[8]=='x')?'t':'T';

  aperm[9]='\0';
  cout <<aperm;
}
void display_file_type(struct stat s)
{
    switch(s.st_mode&S_IFMT)
    {
         case S_IFREG:cout << "-"; break;
         case S_IFDIR:cout <<"d"; break;
         case S_IFBLK:cout <<"b"; break;
         case S_IFCHR:cout <<"c";break;
         case S_IFIFO:cout << "p";break;
         case S_IFLNK:cout<<"l";break;
    }
}
```