**DAYANANDA SAGAR COLLEGE OF ENGINEERING**

**SHAVIGE MALLESHWARA HILLS, K.S. LAYOUT, BANGALORE -560078.**

**DEPARTMENT**

**OF**

**COMPUTER SCIENCE AND ENGINEERING**



**System Software & Operating System Laboratory**

**Subject Code:**

**Semester: VI**

**Vision and Mission of the Department**

**Vision**

To provide a vibrant learning environment in computer science and engineering with focus on industry needs and research, for the students to be successful global professionals contributing to the society.

**Mission**

* To adopt a contemporary teaching learning process with emphasis on hands on and collaborative learning.
* To facilitate skill development through additional training and encourage student forums for enhanced learning.
* To collaborate with industry partners and professional societies and make the students industry ready.
* To encourage innovation through multidisciplinary research and development activities.
* To inculcate human values and ethics in students and groom them to be responsible citizens.

**COMPUTER PROGRAMMING LABORATORY**

**Subject Code: 15CPL16/15CPL26**

**Index**

|  |  |  |
| --- | --- | --- |
| **Program No.** | **Name of the program** | **Page No.** |
| **1** | Install Linux Operating System and explore the Linux System Environment | **1** |
| **2** | Explore various Linux Internal and External Commands | **7** |
| **3** | a) Program to count the number of characters, words, spaces and lines in a given input file.  b) Program to count the numbers of comment lines in a given C program. Also eliminate them and copy the resulting program into separate file | **13** |
| **4** | a) Program to recognize a valid arithmetic expression and to recognize the identifiers and operators present. Print them separately.  b) Program to recognize whether a given sentence is simple or compound. | **17** |
| **5** | a) Program to recognize and count the number of identifiers in a given input file.  b) Program to evaluate an arithmetic expression involving operators +, -, \* and /. | **21** |
| **6** | a) Program to recognize a valid arithmetic expression that uses operators +, -, \* and /.  b) Program to recognize a valid variable, which starts with a letter, followed by any number of letters or digits. | **26** |
| 7 | a) Program to recognize the grammar (anb, n>= 10).   * 1. b) Program to recognize strings ‘aaab’, ‘abbb’, ‘ab’ and ‘a’ using the grammar (anbn, n>= 0). | **31** |
| 8 | Design, develop and implement YACC/C program to demonstrate *Shift Reduce Parsing* technique for the grammar rules: *E E+T | T, T T\*F | F, F (E) | id* and  parse the sentence: *id + id \* id*. | **36** |
| 9 | a) Shell script that accepts two file names as arguments, checks if the permissions for these files are identical and if the permissions are identical, outputs the common permissions, otherwise outputs each file name followed by its permissions  b) Write a C/Java program that creates a child process to read commands from the standard input and execute them (a minimal implementation of a shell – like program). You can assume that no arguments will be passed to the commands to be executed. | **40** |
| 10 | a) Write a C/Java program that creates a zombie and then calls system to execute the ps command to verify that the process is zombie.  b) Write a C/Java program to avoid zombie process by forking twice. | **45** |
|  | a) Write a C/Java program to illustrate the race condition.  b) Write a C/Java program which demonstrates interprocess communication between a reader process and a writer process. Use mkfifo, open, read, write and close APIs in your program |  |
|  | **Operating Systems:**  Using OpenMP, Design, develop and run a multi-threaded program to generate and print Fibonacci Series. One thread has to generate the numbers up to the specified limit and another thread has to print them. Ensure proper synchronization |  |

## **Do’s and Don’ts**

**Do’s**

1. Read and understand the program thoroughly before coming to the laboratory.
2. Report any broken plugs or exposed electrical wires to your faculty/laboratory technician immediately.
3. Turn off the machine once you have finished using it.
4. Maintain silence while working in the lab.
5. Keep the Computer lab premises clean and tidy.
6. Place backpacks under the table or computer counters.

**Don’ts**

1. Do not talk on cell phones in the lab.
2. Do not eat or drink in the laboratory.
3. Do not touch, connect or disconnect any plug or cable without the faculty/laboratory technician’s permission.
4. Do not misbehave in the computer laboratory.
5. Do not install or download any software or modify or delete any system files on any lab computers.
6. Do not read or modify other users' files.
7. Do not plug in external devices without scanning them for computer viruses.
8. Please treat fellow users of the laboratory, and all equipment within the laboratory, with the appropriate level of care and respect.
9. Do not leave your personal belongings unattended. We are not responsible for any theft.

**Head,**

**Dept. of CSE**

1. Install Linux Operating System and explore the Linux System Environment
2. Explore various Linux Internal and External Commands
3. A) Program to count the number of characters, words, spaces and lines from a given input file.

**Program: lab3a.l**

**Code:**

{

#include<stdio.h>

int wc=0, cc=0, lc=0, bc=0;

char infile[25];

%}

word [^” “\t\n0-9]+

eol [\n]

spc [^a-zA-Z0-9” “\t\n]+

%%

{spc} {cc+=yyleng;}

{word} {wc++;cc+=yyleng;}

{eol} {lc++;cc++;}

[“ “] {bc++;cc++;}

[\t] {bc+=8;cc++;}

• {cc++;}

%%

int yywrap(void)

{

}

int main()

{

printf("\nRead the input file\n");

scanf("%s",infile);

yyin=fopen(infile,"r");

yylex();

fclose(yyin);

printf("Number of characters =%d\n",cc);

printf("Number of words = %d\n",wc);

printf("Number of spaces = %d\n",bc);

printf("Number of lines = %d\n",lc);

return 0;

}

**How to run above program**

**$ lex lab3a.l**  //=> generates lex.yy.c file

**$ cc lex.yy.c** //=> generates y.tab.c and y.tab.h(file of token definitions)

**$ ./a.out**

**Input:**

$. /a.out

Enter the file name

ABC.txt

**Output:**

Number of characters=56

Number of words=9

Number of spaces=8

Number of lines=1

B) Program to count the number of comment lines in a given C Program. Also eliminate them and copy it to a separate file.

**Program:lab3b.l**

**Code:**

%{

int cc=0;

%}

%x CMNT

%%

"/\*" {BEGIN CMNT;}

<CMNT>• ;

<CMNT>\n ;

<CMNT>"\*/" {BEGIN 0; cc++;}

%%

int yywrap(void){}

main(int argc,char \*argv[])

{

if(argc!=3)

{

printf("Usage : %s <scr\_file> <dest\_file>\n",argv[0]);

return;

}

yyin=fopen(argv[1],"r");

yyout=fopen(argv[2],"w");

yylex();

printf("\nNumber of multiline comments = %d\n",cc);

}

**How to run above program**

$ lex lab3b.l

$ cc lex.yy.c

$ ./a.out abc abc1

**Input and output:**

$ cat > abc

/\* my prog\*/

//Prog1

// to say hi to the user

/\* also say hello to the user\*/

main()

{

Printf(“hi”);

Printf(“hello”);

}

^d

$ lex prog1b.l

$ cc lex.yy.c –ll

$./a. Number of comment lines1 (//)=2

Number of comment lines2 (/\*) =2

$ cat abc1

main()

{

Printf(“hi”); Printf(“hello”);

}

1. A) Lex program to recognize a valid arithmetic expression and to recognize identifiers and operators present and print them separately

**Program:** 4**a.l**

**Code:**

%{

int a[]={0,0,0,0},valid=1,opnd=0,top=-1,i;

%}

%x oper

%%

(“(“[a-zA-Z0-9]+)+ {BEGIN oper;opnd++;}

<oper>”+” {if(valid) {valid=0;i=0;} else ext();}

<oper>”-” {if(valid) {valid=0;i=1;} else ext();}

<oper>”\*” {if(valid) {valid=0;i=2;} else ext();}

<oper>”/” {if(valid) {valid=0;i=3;} else ext();}

<oper>”(“ {top++;}

<oper>”)” {top--;}

<oper>[a-zA-Z0-9]+ {opnd++;if(valid==0) {valid=1;a[i]++;} else ext();}

<oper>”\n” {if(valid==1&&top==-1) {printf(“\nValid expression\n”); return 0;} else ext(); }

•\n ext();

%%

ext()

{

printf(“\nInvalid expression\n”);

exit(0);

}

main()

{

printf(“\nEnteran arithmetic expression\n”);

yylex();

printf(“\nNumber of operands = %d\n”,opnd);

printf(“\nNumber of + = %d\n

Number of - = %d\n

Number of \* = %d\n

Number of / = %d\n”,a[0],a[1],a[2],a[3]);

}

**How to run above program**

**$ lex lab4a.l**

**$ cc lex.yy.c**

**$ ./a.out**

**Input & Output:**

Enter an arithmetic expression

a+b\*c/d

^d

Valid expression

Operands are:

a b c d

operators are:

+ \* /

$ ./a.out

Enter an arithmetic expression

a++b\*\*

^d

Invalid expression

B) Program to check whether a given sentence is simple or compound

**Program:** **4b.l**

**Code:**

%{

%}

ws [ \t\n]

%%

{ws}”and”|”AND”{ws} |

{ws}”or”|”OR”{ws} |

{ws}”but”|”BUT”{ws} |

{ws}”because”{ws} |

{ws}”nevertheless”{ws} {printf(“\compound sentence\n”);exit(0);}

• ;

\n return 0;

%%

main()

{

printf(“\nEnter a sentence\n”);

yylex();

printf(“\nSimple sentence\n”);

}

**How to run above program**

$ lex lab2b.l

$ cc lex.yy.c

$ ./a.out

**Input & output:**

1. Enter a sentence B. Enter a sentence

I and you You me

^d ^d

Compound sentence Simple sentence

1. A) Program to recognize and count the number of identifiers in a given input file

**Program: lab5a.l**

**Code:**

%{

intidc=0;

%}

e[=][ ]\*[0-9]+

ws[ \n\t]\*

id[\_a-zA-Z][\_a-zA-Z0-9]\*

decln “int”|”float”|”clear”|”double”|”short”|”long”|”unsigned”

%x defn

%%

{decln} {BEGIN defn;}

<defn>{ws}{id}{ws}\**,** {idc++;}

<defn>{ws}{id}{ws}\**;** {BEGIN 0;idc++;}

<defn>{ws}{id}{ws}{e}{ws}\**,** {idc++;}

<defn>{ws}{id}{ws}{e}{ws}\**;** {BEGIN 0;idc++;}

<\*>\n ;

<\*>• ;

%%

main(int argc,char \*argv[])

{

if(argc==2)

{

yyin=fopen(argv[1],”r”);

yylex();

printf(“\nNumber of identifiers = %d\n”,idc);

}

else

{

printf(“\nUsage : %s <src\_file> \n”,argv[0]);

}

}

**How to run above program**

**$ lex lab5a.l**

**$ cc lex.yy.c**

**$ ./a.out file3**

**Input & Output:** $ cat > myfile $ lex lab3.l

int a, b,c; $ cc lex.yy.c -ll

char a,b; $ ./a.out myfile

float a1, g56; Number of identifiers=7

^d

B) Program to evaluate an arithmetic expression involving operators +,-,\*,/

**Program: lab5b.l**

**Code:**

%{

#include”y.tab.h”

#include<stdlib.h”

%}

%%

[0-9]+ {yylval=atoi(yytext);return NUM;}

[\t] ;

\n return 0;

• return yytext[0];

%%

**Program: lab5b.y**

**Code:**

%{

#include<stdio.h>

%}

%token NUM

%left ‘+’’-‘

%left ‘/’’\*’

%%

expr:e {printf(“Valid expression\n”);

printf(“\nResult : %d\n”,$1);return 0;}

e:e’+’e {$$=$1+$3;}

| e’-‘e {$$=$1-$3;}

| e’\*’e {$$=$1\*$3;}

| e’/’e {$$=$1/$3;}

| ‘(‘e’)’ {$$=$2;}

| NUM {$$=$1;}

%

main()

{

printf(“\nEnteran arithmetic expression\n”);

yyparse();

}

yyerror()

{

printf(“\nInvalid expression\n”);

exit(0);

}

**How to run above program**

**$ lex lab5b.l**

**$ yacc -d lab5b.y**

**$ cc lex.yy.c -ll**

**$ ./a.out**

**Input & Output:**

Enter an arithmetic expression to be evaluated

2+3

^d

5

1. A) Program to recognize a valid arithmetic expression that uses operates +,-,\*,/

**Program: lab6a.l**

**Code:**

%{

#include”y.tab.h”

%}

%%

[0-9]+(\•[0-9]+)? {return NUM;}

[a-zA-Z][\_a-zA-Z0-9]\* {return ID;}

[\t] ;

\n return 0;

• return yytext[0];

%%

**Program: lab6a.y**

**Code:**

%{

#include<stdio.h>

%}

%token NUM ID

%left ‘+’’-‘

%left’\*’’/’

%%

e:e’+’e

| e’-‘e

| e’\*’e

| e’/’e

| ‘(‘e’)’

| NUM

I ID ;

%%

main()

{

printf(“\nType the expression to be tested\n”);

yyparse();

printf(“\nValid expression\n”);

}

yyerror()

{

printf(“\n Invalid expression! \n”);

exit(0);

}

**How to run above program**

**$ lex lab6a.l**

**$ yacc –d lab6a.y // => Generates y.tab.h file**

**$ cc lex.yy.c –ll // => -ll links with lex library**

**$ ./a.out**

**Input & Output:**

Enter an arithmetic expression

a1+b2-9/8

^d

valid expression

B) Program to recognize a valid variable, which starts with a letter, followed by any number of letters or digits

**Program: lab6b.l**

**Code:**

%{

#include”y.tab.h”

%}

%%

[a-z] return L;

[0-9] return D;

%%

**Program: lab6b.y**

**Code:**

%{

#include<stdio.h>

%}

%token L D

%%

var: L E {printf(“Valid Variable\n”);return 0;}

E: E L

| E D

| ;

%%

main()

{

printf(“\nEnter a variable\n”);

yyparse();

}

yyerror()

{

printf(“\nInvalid Variable\n”);

exit(0);

}

**How to run above program**

**$ lex lab6b.l**

**$ yacc –d lab6b.y**

**$ cc lex.yy.c -ll**

**$ ./a.out**

**Input & Output:**

1. Enter a variable b. Enter a variable

Abcd3456 34534ttt

^d ^d

Valid variable Invalid variable

1. A) Program to recognize the grammar (anb, n>=10)

**Program: lab7a.l**

**Code:**

%{

#include”y.tab.h”

%}

%%

a return A;

b return B;

\n return yytext[0];

• return yytext[0];

%%

**Program: lab7a.y**

**Code:**

%{

#include<stdio.h>

%}

%token A B

%%

str: S’\n’ {return 0;}

S:X B

X: A AAAAAAAAA T;

T:T A

| ;

%%

main()

{

printf(“\nEnter a string\n”);

yyparse();

printf(“\nEntered string is valid\n”);

}

yyerror()

{

printf(“\nInvalid!\n”);

exit(0);

}

**How to run above program**

$ lex lab7a.l

$ yacc –d lab7a.y

$ cc lex.yy.c -ll

$ ./a.out

**Input & Output:**

Enter a string with a’s and b’s

aaaaaaaaaaaaaaab

^d

Valid

$./a.out

Enter a string with a’s and b’s

aabbbaaaa

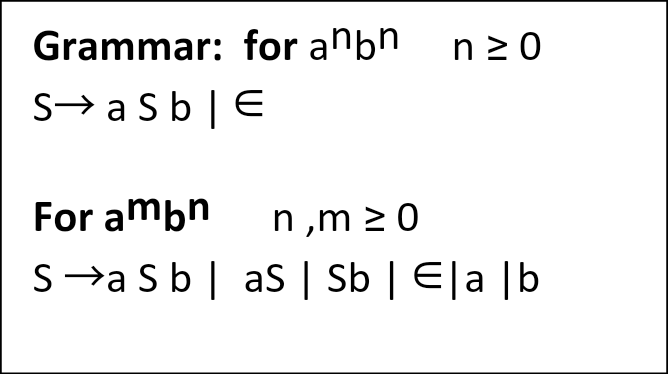
^d

Invalid

B) Program to recognize strings aaab,abbb,ab and a using the grammar (anbn,n>=0)

**Program: lab7b.l**

**Code:**

%{

#include”y.tab.h”

%}

%%

a return A;

b return B;

\n return yytext[0];

• returnyytext[0];

%%

lab5b.y

%{

#include<stdio.h>

%}

%token A B

%%

str: S’\n’ {return 0;}

S:A S

| S B

| ;

%%

main()

{

printf(“\nEnter a string\n”);

yyparse();

printf(“\nEntered string is valid\n”);

}

yyerror()

{

printf(“\nInvalid!\n”);

exit(0);

}

**How to run above program**

**$ lex lab7b.l**

**$ yacc –d lab7b.y**

**$ cc lex.yy.c -ll**

**$ ./a.out**

**Input & Output:**

Enter a string with a’s and b’s

aaab

^d

Valid

$ ./a.out

Enter a string with a’s and b’s

aabbbaaaa

^d

Invalid

1. Design, develop and implement YACC/C program to demonstrate *Shift Reduce Parsing* technique for the grammar rules: *E E+T | T, T T\*F | F, F (E) | id* and parse the sentence: *id + id \* id*.

#include<stdio.h>

#include<conio.h>

#include<string.h>

int k=0,z=0,i=0,j=0,c=0;

char a[16],ac[20],stk[15],act[10];

void check();

void main()

{

puts("GRAMMAR is E->E+E \n E->E\*E \n E->(E) \n E->id");

puts("enter input string ");

gets(a);

c=strlen(a);

strcpy(act,"SHIFT->");

puts("stack \t input \t action");

for(k=0,i=0; j<c; k++,i++,j++)

{

if(a[j]=='i' && a[j+1]=='d')

{

stk[i]=a[j];

stk[i+1]=a[j+1];

stk[i+2]='\0';

a[j]=' ';

a[j+1]=' ';

printf("\n$%s\t%s$\t%sid",stk,a,act);

check();

}

else

{

stk[i]=a[j];

stk[i+1]='\0';

a[j]=' ';

printf("\n$%s\t%s$\t%ssymbols",stk,a,act);

check();

}

}

getch();

}

Void check()

{

strcpy(ac,"REDUCE TO E");

for(z=0; z<c; z++)

if(stk[z]=='i' && stk[z+1]=='d')

{

stk[z]='E';

stk[z+1]='\0';

printf("\n$%s\t%s$\t%s",stk,a,ac);

j++;

}

for(z=0; z<c; z++)

if(stk[z]=='E' && stk[z+1]=='+' && stk[z+2]=='E')

{

stk[z]='E';

stk[z+1]='\0';

stk[z+2]='\0';

printf("\n$%s\t%s$\t%s",stk,a,ac);

i=i-2;

}

for(z=0; z<c; z++)

if(stk[z]=='E' && stk[z+1]=='\*' && stk[z+2]=='E')

{

stk[z]='E';

stk[z+1]='\0';

stk[z+1]='\0';

printf("\n$%s\t%s$\t%s",stk,a,ac);

i=i-2;

}

for(z=0; z<c; z++)

if(stk[z]=='(' && stk[z+1]=='E' && stk[z+2]==')')

{

stk[z]='E';

stk[z+1]='\0';

stk[z+1]='\0';

printf("\n$%s\t%s$\t%s",stk,a,ac);

i=i-2;

}

}

1. A)Write a $hell script that accepts two file names as arguments, checks if the permissions for both the files are identical and if the permissions are identical, outputs the common permissions; otherwise outputs each file name followed by its permissions

**Program: 9a.sh**

**Code:**

clear

if [ $# != 2 ]

then

echo “Invalid input!!!”

else

p1=**`**ls -l $1 | cut -d “ “ -f 1**`**

p2=**`**ls -l $2 | cut -d “ “ -f 1**`**

if [ $p1 == $p2 ]

then

echo “the file permissions are same and it is : “

echo “$p1”

else

echo “The file permissions are different”

echo “$1 : $p1”

echo “$2 : $p2”

fi

fi

**How to run above program**

**$ sh lab9a.sh** **file1 file2**

**Input & Output:**

**$ sh lab9a.sh** **file1 file2**

The permissions are different

Filename: aaa permission: drwxr-xr-x

Filename: tem permission: -rw-r--r--

[root@localhost]# sh 8a.sh tem temp

The permissions are same

-rw-r—r—

B) C program that creates a child process to read commands from the standard input and execute them

**Program: lab9b.c**

**Code:**

#include<stdio.h>

int main()

{

intch,rv;

charcmd[10];

rv=fork();

if(rv==0)

{

do

{

printf(“\nEnter a command\n”);

scanf(“%s”,cmd);

system(cmd);

printf(“\n1 : continue\n2 : exit\n”);

scanf(“%d”,&ch);

}

while(ch!=0);

}

else

{

wait(0);

printf(“\nChild terminated\n”);

}

return 0;

}

**How to run above program**

**$ cc lab9b.c**

**$ ./a.out**

**Output:**

Child process

Enter the command:date

Thu June 12 13:17:32 IST 2013

Enter 1 to continue or 0 to exit:1

Enter the command:ls

1a.sh 2a.sh 3a.sh 4a.sh 5a.sh 6a.sh 7a.sh 1b.c 2b.c 3b.c 4b.c 5b.c 6b.c 7b.c a.out cc

Enter 1 to continue or 0 to exit:1

Enter the command:who

root :0 Sep 10 12:54

root pts/1 Sep 10 12:55 (:0.0)

Enter 1 to continue or 0 to exit:1

Enter the command:ps

PID TTY TIME CMD

3620 pts/1 00:00:00 bash

3705 pts/1 00:00:00 a.out

3706 pts/1 00:00:00 a.out

3711 pts/1 00:00:00 ps

Enter 1 to continue or 0 to exit:1

Enter the command:pwd

/root/dsce/unix

Enter 1 to continue or 0 to exit:0

1. A) Write a C/Java program that creates a zombie and then calls system to execute the ps command to verify that the process is zombie.

#include<stdio.h>

#include <stdlib.h>

#include <sys/types.h>

#include <unistd.h>

int main()

{

pid\_t child\_pid;

/\* Create a child process. \*/

child\_pid = fork ();

if (child\_pid > 0)

{

printf("This is the parent process: %d. Sleep for a minute\n",getpid());

sleep (60);

}

else

{

printf("This is the child process: %d. Exit immediately\n",getpid());

exit (0);

}

system("ps -e -o pid,ppid,stat,cmd");

return 0;

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Program output:**

[root@localhost ~]# vim p10a.c

[root@localhost ~]# cc p10a.c

[root@localhost ~]# ./a.out

This is the child process: 8822. Exit immediately

This is the parent process: 8821. Sleep for a minute

PID PPID STAT CMD

1 0 Ss init [5]

2 0 S< [kthreadd]

3 2 S< [migration/0]

4 2 S< [ksoftirqd/0]

5 2 S< [watchdog/0]

6 2 S< [migration/1]

7 2 S< [ksoftirqd/1]

....

8788 8782 SN sleep 506

8798 2793 T vim p7.c

8821 2793 S+ ./a.out

8822 8821 Z+ [a.out] <defunct>

8825 8821 R+ ps -e -o pid,ppid,stat,cmd

B) Write a C/Java program to avoid zombie process by forking twice.

#include<stdio.h>

#include <stdlib.h>

#include <sys/types.h>

#include <unistd.h>

#include <sys/wait.h>

int main()

{

pid\_t pid;

if ((pid=fork())< 0)

{

printf("Fork error");

}

else if( pid==0)

{

printf("first child pid=%d\n", getpid());

if((pid=fork())< 0)

printf("Fork error");

else if( pid > 0)

exit(0);

sleep(5);

printf("second child pid = %d\n parent pid=%d\n", getpid(), getppid());

exit (0);

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Program output:**

[root@localhost ~]# vi p10b.c

[root@localhost ~]# cc p10b.c

[root@localhost ~]# ./a.out

first child pid=4071

second child pid = 4072

parent of second child pid=4071

PID PPID STAT CMD

1 0 Ss init [5]

2 0 S< [kthreadd]

3 2 S< [migration/0]

4 2 S< [ksoftirqd/0]

5 2 S< [watchdog/0]

6 2 S< [migration/1]

7 2 S< [ksoftirqd/1]

........

4070 2686 S+ ./a.out

4071 4070 S+ ./a.out

4072 4071 Z+ [a.out] <defunct>

4073 4071 R+ ps -e -o pid,ppid,stat,cmd

second child pid = 4071

parent of second child pid=4070

1. A) Write a C/C++ program to illustrate the race condition.

#include<stdio.h>

#include<unistd.h>

#include<stdlib.h>

static void charatatime (char \*);

int main (void)

{

pid\_t pid;

if ((pid=fork( ))< 0)

{

printf("fork error\n");

}

else if(pid==0)

{

charatatime("Output from child\n");

}

else

{

charatatime("Output from parent\n");

}

exit(0);

}

static void charatatime(char \*str)

{

char \*ptr;

int c;

setbuf(stdout,NULL); /\* set unbuffered\*/

for(ptr=str;(c=\*ptr++)!=0;)

putc(c,stdout);

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Program output:**

[root@localhost ~]# vim p11a.c

[root@localhost ~]# cc p11a.c

[root@localhost ~]# ./a.out

Output froOutput from parent

m child

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

#include<stdio.h>

#include<stdlib.h>

#include<fcntl.h>

#include<unistd.h>

#include<sys/types.h>

#include<string.h>

int main(int argc,char \*argv[])

{

int fd,num1,num2;

char buf[100];

if(argc==3)

{

mkfifo(argv[1],0666);

fd=open(argv[1],O\_WRONLY);

num1=write(fd,argv[2],strlen(argv[2]));

printf("no of bytes written%d\n",num1);

}

if(argc==2)

{

fd=open(argv[1],O\_RDONLY);

num2=read(fd,buf,sizeof(buf));

buf[num2]='\0';

printf("the message size %d read is %s",num2,buf);

}

close(fd);

unlink(argv[1]);

return 0;

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Program Output:**

**Terminal 1**

[root@localhost ~]# vi p11b.c

[root@localhost ~]# cc p11b.c –o writerprocess

[root@localhost ~]# writerprocess pipe1 helloworld

Number of bytes written = 10

**Terminal 2**

[root@localhost ~]# vi p11b.c

[root@localhost ~]# cc p11b.c –o readerprocess

[root@localhost ~]# readerprocess pipe1

Message of size 10 bytes received

Message = helloworld

1. Design develop and run a multi-threaded program to generate and print Fibonacci series. One thread has to generate the numbers up to the specified limit and Another thread has to print them. Ensure proper synchronization.

# include<stdio.h>

# include<omp.h>

# include<stdlib.h>

int MAX;

int Fibonacci(int n)

{

int x, y;

if (n < 2)

return n;

else

{

x = Fibonacci(n - 1);

y = Fibonacci(n - 2);

return (x + y);

}

}

int FibonacciTask(int n)

{

int x, y;

if (n < 2)

return n;

else

{

x = Fibonacci(n - 1);

y = Fibonacci(n - 2);

return (x + y);

}

}

int random\_num()

{

int temp;

temp = rand();

temp = temp%24;

MAX = temp;

return(MAX);

}

int main(int argc, char \* argv[])

{

int FibNumber[25] = {0};

int j, temp,tmp,id,i = 0;

int n, tid, nthreads;

printf("Please Enter the number Range :");

scanf("%d",&n);

printf("\n");

omp\_set\_num\_threads(2);

//Parallel region

#pragma omp parallel

{

printf("The number of threads are %d\n",omp\_get\_num\_threads());

#pragma omp for private (tid, tmp, FibNumber)

for(j = 1; j<=n; j++)

{

tmp = random\_num();

/\* Get thread number \*/

/\* tid = omp\_get\_thread\_num();

printf("The number of threads are %d\n",omp\_get\_num\_threads());

printf("The thread id is = %d\n", tid); \*/

/\* The critical section here will enable, not more then one

thread to execute in this section (synchronization) \*/

#pragma omp critical

{

/\* Get thread number \*/

/\* tid = omp\_get\_thread\_num();

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* inside critical section \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf("The thread id is = %d\n", tid); \*/

for(i = 1; i <= tmp; i++)

FibNumber[i] = FibonacciTask(i);

printf("The number value is %d:",tmp);

for(i = 1; i <= tmp; i++)

printf("%d \t", FibNumber[i]);

printf("\n\n");

}

}

}

}

**How to run above program**

$ cc –fopenmp lab12.c

$ ./a.out

**Output:**

Please Enter the number Range : 7

The number value is 7:1 1 2 3 5 8 13

The number value is 22:1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597 2584 4181 6765 10946 17711

The number value is 9:1 1 2 3 5 8 13 21 34

The number value is 19:1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597 2584 4181

The number value is 17:1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597

The number value is 7:1 1 2 3 5 8 13

The number value is 10:1 1 2 3 5 8 13 21 34 55