**SHELL PROGRAMMING**

**ASSIGNMENT 1:**

**PROBLEM:**

Create a database file *employee.lst* with six fields: employee\_id, name, designation, department, date\_of\_birth and basic\_pay separated by delimiter “/”. Perform the following operations on the file:

1. Sort the file with respect to:
2. Date of birth
3. Designation as primary key and name as secondary key
4. Find out various departments that occurred in the file.
5. Find out number of persons in each department.
6. Find out pay slip of each employee using D.A. = 60% of Basic Pay, H.R.A. = 5% of Basic Pay, P.F. = 10% of Basic Pay. Store pay slip in a file with: name, department, basic pay, D.A., H.R.A., P.F., Gross.
7. Find out total gross salary in each department.
8. Find out the people having the same month of birth as the current month.

**CODE:**

src="Employee\_list.txt"

#ID/NAME/DESIGNATION/DATE-OF-BIRTH/BASIC-PAY/DEPARTMENT

echo -e "\tMENU"

echo -e "1.Sort"

echo -e "2.Departments"

echo -e "3.Persons in each department"

echo -e "4.Create pay slip"

echo -e "5.Gross salary in each department"

echo -e "6.Persons born this month"

echo -e "7.Exit"

read -p "Enter choice:- " ch

case $ch in

'1')

sort -t"/" -k 4 -k 3 -k 2 $src -o temp.txt

echo -v"/" | cat temp.txt

rm temp.txt

;;

'2')

sort -u -t"/" -k 6 $src | cut -d"/" -f 6

;;

'3')

sort -t"/" -k 6 $src | cut -d"/" -f 6 | uniq -c

;;

'4')

noln=`wc $src -l | cut -d" " -f 1`

i=1;

dst="Pay\_Slip.txt"

#echo "NAME|DEPT|Basic|DA|HRA|PF|GROSS"|cat>>$dst

while [ $i -le $noln ]

do

bp=`head -n $i $src | tail -n 1 | cut -d"/" -f 5`

da=`expr $bp \\* 6`

da=`expr $da / 100`

hra=`expr $bp \\* 15`

hra=`expr $hra / 100`

pf=`expr $bp \\* 1`

pf=`expr $pf / 100`

gross=`expr $bp + $da + $hra - $pf`

name=`head -n $i $src | tail -n 1 | cut -d"/" -f 2`

dept=`head -n $i $src | tail -n 1 | cut -d"/" -f 6`

echo "$name / $dept / $bp / $da / $hra / $pf / $gross"|cat>>$dst

i=`expr $i + 1`

done

;;

'5')

dst="Pay\_Slip.txt"

noln=`wc $dst -l | cut -d" " -f 1`

i=1;

while [ $i -le $noln ]

do

gs=`head -n $i $dst | tail -n 1 | cut -d"/" -f 7`

dept=`head -n $i $dst | tail -n 1 | cut -d"/" -f 2`

echo "$gs / $dept" | cat>>temp.txt

i=`expr $i + 1`

done

sort -t"/" -k 2 temp.txt | cat>>temp1.txt

temp=`sort -t"/" -k 2 temp.txt | cut -d"/" -f 2 | uniq -c`

echo "$temp" | cat>>temp2.txt

m=`wc temp2.txt -l | cut -d" " -f 1`

i=1

k=1

while [ $i -le $m ]

do

n=`head -n $i temp2.txt | tail -n 1 | cut -d" " -f 7`

j=1

tgs=0

while [ $j -le $n ]

do

gs=`head -n $k temp1.txt | tail -n 1 | cut -d" " -f 2`

tgs=`expr $tgs + $gs`

k=`expr $k + 1`

echo $k

j=`expr $j + 1`

done

dept=`head -n $i temp2.txt | tail -n 1 | cut -d" " -f 11`

echo "$dept $tgs"

i=`expr $i + 1`

done

rm temp.txt

rm temp1.txt

rm temp2.txt

;;

'6')

m=`date +%m`

x=`wc $src -l | cut -d" " -f 1`

i=1

while [ $i -le $x ]

do

gs=`head -n $i $src | tail -n 1 | cut -d"/" -f 4`

echo "$gs" | cat>>temp3.txt

mn=`tail -n 1 temp3.txt | cut -d"-" -f2`

if [ $mn -eq $m ]

then

name=`head -n $i $src | tail -n 1 | cut -d"/" -f 2`

echo "$name" | cat>>birth.txt

fi

i=`expr $i + 1`

done

rm temp3.txt

;;

'7')

exit

;;

\*)

;;

esac

**OUTPUT:**

**ASSIGNMENT 2:**

**PROBLEM:**

Write a shell script for library database in the following way: Create the database file library with seven fields: Accession No., Book Name, Author Name, Subject, Publication, No. of Copies, Year of Publication.

1. Enter some new books.
2. Update subject of a particular book.
3. List all the books of some publisher.
4. Sort with respect to subject and primary key and author name, publisher name as secondary key.
5. Delete a particular book.
6. Find out the no. of copies of a book referred by book name, author name.
7. Sort the books according to the publisher.
8. Find publisher having highest no. of books.

**CODE:**

src="library.txt"

#ACCESNO/BOOKNAME/AUTHNAME/SUBJECT/PUBLICATION/NOCOPIES/YEARPUB

echo -e "\tMENU"

echo -e "1.Enter new books"

echo -e "2.Update subject of a book"

echo -e "3.List all books of some publication"

echo -e "4.Sort by Subject-Author Name-Publisher"

echo -e "5.Delete a book"

echo -e "6.Number of copies of a book"

echo -e "7.Sort by Publisher and number of books by the Publisher"

echo -e "8.Publisher with highest number of books"

echo -e "9.Exit"

read -p "Enter choice:- " ch

case $ch in

'1')

echo "ACCESNO/BOOKNAME/AUTHNAME/SUBJECT/PUBLICATION/NOCOPIES/YEARPUB"|cat>>$src

while true

do

read -p "Enter Book Access Number:- " no

read -p "Enter Book Name :- " name

read -p "Enter Author Name :- " aname

read -p "Enter Subject :- " sub

read -p "Enter Publisher :- " pub

read -p "Enter Number of Copies :- " nocop

read -p "Enter Year of Publication :- " ypub

echo "$no/$name/$aname/$sub/$pub/$nocop/$ypub"|cat>>$src

read -p "Do you want to enter more?(Y/y) " c

if [ $c != 'y' ]

then

if [ $c != 'Y' ]

then

break

fi

fi

done

;;

'2')

read -p "Enter Book Access Number to update subject:- " acno

read -p "Enter Subject :- " ssub

while read t

do

no=`echo "$t"|cut -d'/' -f'1'`

name=`echo "$t"|cut -d'/' -f'2'`

aname=`echo "$t"|cut -d'/' -f'3'`

sub=`echo "$t"|cut -d'/' -f'4'`

pub=`echo "$t"|cut -d'/' -f'5'`

nocop=`echo "$t"|cut -d'/' -f'6'`

ypub=`echo "$t"|cut -d'/' -f'7'`

if [ $no == $acno ]

then

echo "$no/$name/$aname/$ssub/$pub/$nocop/$ypub"|cat>>temp.txt

else

echo "$no/$name/$aname/$sub/$pub/$nocop/$ypub"|cat>>temp.txt

fi

done<$src

mv temp.txt $src

;;

'3')

read -p "Enter Publication :- " publi

while read t

do

no=`echo "$t"|cut -d'/' -f'1'`

name=`echo "$t"|cut -d'/' -f'2'`

aname=`echo "$t"|cut -d'/' -f'3'`

sub=`echo "$t"|cut -d'/' -f'4'`

pub=`echo "$t"|cut -d'/' -f'5'`

nocop=`echo "$t"|cut -d'/' -f'6'`

ypub=`echo "$t"|cut -d'/' -f'7'`

if [ $pub == $publi ]

then

echo "Book Name :- $name"

echo "By :- $aname"

echo "On :- $sub"

echo "Number of Copies Available :- $nocop"

echo "Published in the Year :- $ypub"

fi

done<$src

;;

'4')

sort -t"/" -k 4 -k 3 -k 5 $src -o temp1.txt

echo -v"/" | cat temp1.txt

rm temp1.txt

;;

'5')

read -p "Enter Book Access Number to delete :- " acno

while read t

do

no=`echo "$t"|cut -d'/' -f'1'`

name=`echo "$t"|cut -d'/' -f'2'`

aname=`echo "$t"|cut -d'/' -f'3'`

sub=`echo "$t"|cut -d'/' -f'4'`

pub=`echo "$t"|cut -d'/' -f'5'`

nocop=`echo "$t"|cut -d'/' -f'6'`

ypub=`echo "$t"|cut -d'/' -f'7'`

if [ $no != $acno ]

then

echo "$no/$name/$aname/$sub/$pub/$nocop/$ypub"|cat>>temp2.txt

fi

done<$src

mv temp2.txt $src

;;

'6')

read -p "Enter Book Name :- " bname

read -p "Enter Author Name :- " athname

while read t

do

no=`echo "$t"|cut -d'/' -f'1'`

name=`echo "$t"|cut -d'/' -f'2'`

aname=`echo "$t"|cut -d'/' -f'3'`

sub=`echo "$t"|cut -d'/' -f'4'`

pub=`echo "$t"|cut -d'/' -f'5'`

nocop=`echo "$t"|cut -d'/' -f'6'`

ypub=`echo "$t"|cut -d'/' -f'7'`

if [ $name == $bname -a $aname == $athname ]

then

echo "Number of Copies Available :- $nocop"

fi

done<$src

;;

'7')

sort -t"/" -k 5 $src -o temp1.txt

echo -v"/" | cat temp1.txt

rm temp1.txt

sort -t"/" -k 5 $src | cut -d"/" -f 5 | uniq -c

;;

'8')

sort -t"/" -k 5 $src | cut -d"/" -f 5 | uniq -c | cat>>temp.txt

sort -k 1 temp.txt | tail -n 1 | cat>>t.txt

cat t.txt

rm temp.txt

rm t.txt

;;

'9')

exit

;;

\*)

;;

esac

**OUTPUT:**

**ASSIGNMENT 3:**

**PROBLEM:**

Write a shell script which list out the line numbers from a file which does not contain five fields.

**CODE:**

# -------------ASSIGNMENT 3-----------

echo "Enter filename: "

read fname

awk -F "|" '{print NF -1}' $fname > file

lc=`wc -l $fname | cut -d " " -f1`

i=1

echo "The file contents are:"

cat $fname

echo "Required line numbers are:"

while [ $i -le $lc ]

do

c=`sed -n -e "$i p" file`

if [ $c -lt 4 ]

then

echo $i

fi

i=`expr $i + 1`

done

echo "END"

**OUTPUT:**

**ASSIGNMENT 4:**

**PROBLEM:**

Write a shell script to find out the following information for all the process in the system:

1. User name and user id of all users working.
2. List the name of the sleeping processes along with their ids.
3. List the name of running processes along with their ids.

**CODE:**

# -------------ASSIGNMENT 4-----------

who | cut -d " " -f1 >file1

term=`tty`

echo "User name ID"

exec <file1

i=1

while read line

do

echo "$line "

ids=`id -u $line`

echo $ids

i=`expr $i + 1`

done

exec <$term

ps -el | awk -F " " '{print $2}' > file2

ps -el | awk -F " " '{print $4}' > file22

ps -el | awk -F " " '{print $14}' > file3

lc=`wc -l file2 | cut -d " " -f1`

i=2

echo "-------------------------------------------"

echo "Sleeping Prosesses"

echo "-------------------------------------------"

while [ $i -le $lc ]

do

str1=`sed -n -e "$i p" file2`

if [ $str1 = "S" ]

then

d=`sed -n -e "$i p" file22`

d1=`sed -n -e "$i p" file3`

echo "$d "

echo $d1

fi

i=`expr $i + 1`

done

i=2

echo "-------------------------------------------"

echo "Running Processes"

echo "-------------------------------------------"

while [ $i -le $lc ]

do

str1=`sed -n -e "$i p" file2`

if [ $str1 = "R" ]

then

d=`sed -n -e "$i p" file22`

d1=`sed -n -e "$i p" file3`

echo "$d "

echo $d1

fi

i=`expr $i + 1`

done

echo "END"

**OUTPUT:**

**SYSTEM PROGRAMMING**

**ASSIGNMENT 5:**

**PROBLEM:**

Write a C program where a child process is created and child process increments one local and global variable. Print values of PID, values of local and global variables within the child process and parent process.

**CODE:**

#include <stdio.h> /\* printf, stderr, fprintf \*/

#include <unistd.h> /\* fork \*/

#include <stdlib.h> /\* exit \*/

#include <errno.h> /\* errno \*/

int main(void)

{

pid\_t pid;

int gv=0;

/\* Output from both the child and the parent process

\* will be written to the standard output,

\* as they both run at the same time.

\*/

pid = fork();

if (pid == 0)

{

int clv=0;

/\* Child process:

\* When fork() returns 0, we are in

\* the child process.

\* Here we count up to ten, one each second.

\*/

gv++;

clv++;

int j;

for (j = 0; j < 2; j++)

{

printf("Child Process:-\nC PID :-- %d\nC Local Variable :-- %d\nC Global Variable :-- %d\n",pid,clv,gv);

sleep(1);

}

exit(0);

}

else if (pid > 0)

{

int plv=0;

/\* Parent process:

\* When fork() returns a positive number, we are in the parent process

\* (the fork return value is the PID of the newly-created child process).

\* Again we count up to ten.

\*/

gv++;

plv++;

int i;

for (i = 0; i < 2; i++)

{

printf("Parent Process:-\nP PID :-- %d\nP Local Variable :-- %d\nP Global Variable :-- %d\n",pid,plv,gv);

sleep(1);

}

exit(0);

}

else

{

/\* Error:

\* When fork() returns a negative number, an error happened

\* (for example, number of processes reached the limit).

\*/

fprintf(stderr, "can't fork, error %d\n", errno);

exit(EXIT\_FAILURE);

}

}

**OUTPUT:**

**ASSIGNMENT 6:**

**PROBLEM:**

Write two programs to implement Producer-Consumer process using ‘pipe’ call. In the first program open a pipe, create a child process and the parent process writes data into the pipe. In the second program child process reads data from the pipe.

**CODE:**

#include <stdlib.h>

#include <stdio.h>

#include <pthread.h>

#include <semaphore.h>

#define RAND\_DIVISOR 100000000

#define TRUE 1

#define BUFFER\_SIZE 5

typedef int buffer\_item;

/\* The mutex lock \*/

pthread\_mutex\_t mutex;

/\* the semaphores \*/

sem\_t full, empty;

/\* the buffer \*/

buffer\_item buffer[BUFFER\_SIZE];

/\* buffer counter \*/

int counter;

pthread\_t tid; //Thread ID

pthread\_attr\_t attr; //Set of thread attributes

void \*producer(void \*param); /\* the producer thread \*/

void \*consumer(void \*param); /\* the consumer thread \*/

void initializeData()

{

/\* Create the mutex lock \*/

pthread\_mutex\_init(&mutex, NULL);

/\* Create the full semaphore and initialize to 0 \*/

sem\_init(&full, 0, 0);

/\* Create the empty semaphore and initialize to BUFFER\_SIZE \*/

sem\_init(&empty, 0, BUFFER\_SIZE);

/\* Get the default attributes \*/

pthread\_attr\_init(&attr);

/\* init buffer \*/

counter = 0;

}

/\* Producer Thread \*/

void \*producer(void \*param)

{

buffer\_item item;

while(TRUE)

{

/\* sleep for a random period of time \*/

int rNum = rand() / RAND\_DIVISOR;

sleep(rNum);

/\* generate a random number \*/

item = rand();

/\* acquire the empty lock \*/

sem\_wait(&empty);

/\* acquire the mutex lock \*/

pthread\_mutex\_lock(&mutex);

if(insert\_item(item))

{

fprintf(stderr, " Producer report error condition\n");

}

else

{

printf("producer produced %d\n", item);

}

/\* release the mutex lock \*/

pthread\_mutex\_unlock(&mutex);

/\* signal full \*/

sem\_post(&full);

}

}

/\* Consumer Thread \*/

void \*consumer(void \*param)

{

buffer\_item item;

while(TRUE)

{

/\* sleep for a random period of time \*/

int rNum = rand() / RAND\_DIVISOR;

sleep(rNum);

/\* aquire the full lock \*/

sem\_wait(&full);

/\* aquire the mutex lock \*/

pthread\_mutex\_lock(&mutex);

if(remove\_item(&item))

{

fprintf(stderr, "Consumer report error condition\n");

}

Else

{

printf("consumer consumed %d\n", item);

}

/\* release the mutex lock \*/

pthread\_mutex\_unlock(&mutex);

/\* signal empty \*/

sem\_post(&empty);

}

}

/\* Add an item to the buffer \*/

int insert\_item(buffer\_item item)

{

/\* When the buffer is not full add the item

and increment the counter\*/

if(counter < BUFFER\_SIZE)

{

buffer[counter] = item;

counter++;

return 0;

}

Else

{ /\* Error the buffer is full \*/

return -1;

}

}

/\* Remove an item from the buffer \*/

int remove\_item(buffer\_item \*item)

{

/\* When the buffer is not empty remove the item

and decrement the counter \*/

if(counter > 0)

{

\*item = buffer[(counter-1)];

counter--;

return 0;

}

Else

{ /\* Error buffer empty \*/

return -1;

}

}

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

{

/\* Loop counter \*/

int i;

/\* Verify the correct number of arguments were passed in \*/

if(argc != 4)

{

fprintf(stderr, "USAGE:./main.out <INT> <INT> <INT>\n");

}

int mainSleepTime = atoi(argv[1]); /\* Time in seconds for main to sleep \*/

int numProd = atoi(argv[2]); /\* Number of producer threads \*/

int numCons = atoi(argv[3]); /\* Number of consumer threads \*/

/\* Initialize the app \*/

initializeData();

/\* Create the producer threads \*/

for(i = 0; i < numProd; i++)

{

/\* Create the thread \*/

pthread\_create(&tid,&attr,producer,NULL);

}

/\* Create the consumer threads \*/

for(i = 0; i < numCons; i++)

{

/\* Create the thread \*/

pthread\_create(&tid,&attr,consumer,NULL);

}

/\* Sleep for the specified amount of time in milliseconds \*/

sleep(mainSleepTime);

/\* Exit the program \*/

printf("Exit the program\n");

exit(0);

}

**OUTPUT:**

**ASSIGNMENT 7:**

**PROBLEM:**

Write a program to implement Client-Server exam. The client reads a file name from a standard input and writes it to the IPC channel. The server reads this file and writes it to the IPC channel. The client then reads the file and writes it to the standard output.

**CODE:**

/\* Client.c \*/

#include <stdlib.h> /\* for exit() \*/

#include <sys/types.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#include <netdb.h>

#include <stdio.h>

#include <unistd.h>

#include <string.h> /\* memset() \*/

#include <sys/time.h> /\* select() \*/

#define REMOTE\_SERVER\_PORT 1500

#define MAX\_MSG 100

#define SOCKET\_ERROR -1

int isReadable(int sd,int \* error,int timeOut)

{ // milliseconds

fd\_set socketReadSet;

FD\_ZERO(&socketReadSet);

FD\_SET(sd,&socketReadSet);

struct timeval tv;

if (timeOut)

{

tv.tv\_sec = timeOut / 1000;

tv.tv\_usec = (timeOut % 1000) \* 1000;

}

else

{

tv.tv\_sec = 0;

tv.tv\_usec = 0;

} // if

if (select(sd+1,&socketReadSet,0,0,&tv) == SOCKET\_ERROR)

{

\*error = 1;

return 0;

} // if

\*error = 0;

return FD\_ISSET(sd,&socketReadSet) != 0;

} /\* isReadable \*/

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

{

int sd, rc, i, n, echoLen, flags, error, timeOut;

struct sockaddr\_in cliAddr, remoteServAddr, echoServAddr;

struct hostent \*h;

char msg[MAX\_MSG];

/\* check command line args \*/

if(argc<3)

{

printf("usage : %s <server> <data1> ... <dataN> \n", argv[0]);

exit(1);

}

/\* get server IP address (no check if input is IP address or DNS name \*/

h = gethostbyname(argv[1]);

if(h==NULL)

{

printf("%s: unknown host '%s' \n", argv[0], argv[1]);

exit(1);

}

printf("%s: sending data to '%s' (IP : %s) \n", argv[0], h->h\_name,

inet\_ntoa(\*(struct in\_addr \*)h->h\_addr\_list[0]));

remoteServAddr.sin\_family = h->h\_addrtype;

memcpy((char \*) &remoteServAddr.sin\_addr.s\_addr,

h->h\_addr\_list[0], h->h\_length);

remoteServAddr.sin\_port = htons(REMOTE\_SERVER\_PORT);

/\* socket creation \*/

sd = socket(AF\_INET,SOCK\_DGRAM,0);

if(sd<0)

{

printf("%s: cannot open socket \n",argv[0]);

exit(1);

}

/\* bind any port \*/

cliAddr.sin\_family = AF\_INET;

cliAddr.sin\_addr.s\_addr = htonl(INADDR\_ANY);

cliAddr.sin\_port = htons(0);

rc = bind(sd, (struct sockaddr \*) &cliAddr, sizeof(cliAddr));

if(rc<0)

{

printf("%s: cannot bind port\n", argv[0]);

exit(1);

}

/\* BEGIN jcs 3/30/05 \*/

flags = 0;

timeOut = 100; // ms

/\* send data \*/

for(i=2;i<argc;i++)

{

rc = sendto(sd, argv[i], strlen(argv[i])+1, flags,

(struct sockaddr \*) &remoteServAddr,

sizeof(remoteServAddr));

if(rc<0)

{

printf("%s: cannot send data %d \n",argv[0],i-1);

close(sd);

exit(1);

}

/\* init buffer \*/

memset(msg,0x0,MAX\_MSG);

while (!isReadable(sd,&error,timeOut)) printf(".");

printf("\n");

/\* receive echoed message \*/

echoLen = sizeof(echoServAddr);

n = recvfrom(sd, msg, MAX\_MSG, flags,

(struct sockaddr \*) &echoServAddr, &echoLen);

if(n<0)

{

printf("%s: cannot receive data \n",argv[0]);

continue;

}

/\* print received message \*/

printf("%s: echo from %s:UDP%u : %s \n",

argv[0],inet\_ntoa(echoServAddr.sin\_addr),

ntohs(echoServAddr.sin\_port),msg);

}

return 1;

}

/\* Server.c \*/

#include <stdlib.h>

#include <sys/types.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#include <netdb.h>

#include <stdio.h>

#include <unistd.h> /\* close() \*/

#include <string.h> /\* memset() \*/

#define LOCAL\_SERVER\_PORT 1500

#define MAX\_MSG 100

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

{

int sd, rc, n, cliLen, flags;

struct sockaddr\_in cliAddr, servAddr;

char msg[MAX\_MSG];

/\* socket creation \*/

sd=socket(AF\_INET, SOCK\_DGRAM, 0);

if(sd<0)

{

printf("%s: cannot open socket \n",argv[0]);

exit(1);

}

/\* bind local server port \*/

servAddr.sin\_family = AF\_INET;

servAddr.sin\_addr.s\_addr = htonl(INADDR\_ANY);

servAddr.sin\_port = htons(LOCAL\_SERVER\_PORT);

rc = bind (sd, (struct sockaddr \*) &servAddr,sizeof(servAddr));

if(rc<0)

{

printf("%s: cannot bind port number %d \n",

argv[0], LOCAL\_SERVER\_PORT);

exit(1);

}

printf("%s: waiting for data on port UDP %u\n",

argv[0],LOCAL\_SERVER\_PORT);

flags = 0;

/\* server infinite loop \*/

while(1)

{

/\* init buffer \*/

memset(msg,0x0,MAX\_MSG);

/\* receive message \*/

cliLen = sizeof(cliAddr);

n = recvfrom(sd, msg, MAX\_MSG, flags,

(struct sockaddr \*) &cliAddr, &cliLen);

if(n<0)

{

printf("%s: cannot receive data \n",argv[0]);

continue;

}

/\* print received message \*/

printf("%s: from %s:UDP%u : %s \n",

argv[0],inet\_ntoa(cliAddr.sin\_addr),

ntohs(cliAddr.sin\_port),msg);

sleep(1);

sendto(sd,msg,n,flags,(struct sockaddr \*)&cliAddr,cliLen);

}/\* end of server infinite loop \*/

return 0;

}

**OUTPUT:**