KARPAGAM ACADEMY OF HIGHER EDUCATION

Coimbatore - 641021.

DEPARTMENT OF COMPUTER SCIENCE, CA & IT

SUBJECT : LINUX PRACTICAL
SEMESTER : III
SUBJECT CODE : 16CSP312 **CLASS : II M.Sc.CS**

---

1. To write a Linux program to display process deadlock state.
2. To write a program to display the allocated memory.
3. To write a program to simulate the DOS Command-Copy.
4. To write a program to implement signal handling.
5. To write a simple Linux program using thread.
6. To write a program to display the date & time using TCP Sockets.
7. To write a program to display the date & time using UDP Sockets.
8. To write a program to display the cpu scheduling
9. To write a Linux program to create a lock file.
10. To write a program to display the user information

EX.NO: 1 DATE:	MEMORY ALLOCATION

AIM:

To write a program to display allocated memory.

ALGORITHM:

STEP1: Start the process.

STEP2: Include all the necessary header files.

STEP3: Use printf and scanf to print and get the output.

STEP4: Declare the necessary integer and size.

STEP5: Assign the memory allocation using **malloc()**.

STEP6: If the memory is full it popup the message that memory is full or otherwise it will allocate the address.

STEP7: Save the program with extension.c and run.

STEP8: Stop the process.

CODING:

```
#include<stdio.h>
#include<stdlib.h>
#define null 0
void main()
{
int *p,*t,size;
printf("\n enter the size");
scanf("%d",&size);
printf("\n");
t=(int*)malloc(size*sizeof(int));
if(t=null)
{
printf("memory space not available");
exit(0);
}
printf("\n address of the 1st byte is=%u",t);
printf("enter the input values\n\n");
for(p=t;p<t+size;p++)
{
scanf("%d",p);
}
for(p=t;p<t+size;p++)
{
printf("%d is stored at address%u",*p,p);
}
}
```

OUTPUT:

```
login004@ku030-OEM ~ $ cd Desktop
login004@ku030-OEM ~/Desktop $ cd 16csp012
login004@ku030-OEM ~/Desktop/16csp012 $ cc -c mem.c
login004@ku030-OEM ~/Desktop/16csp012 $ cc -o mem mem.c
login004@ku030-OEM ~/Desktop/16csp012 $ ./mem
enter the size 5
address of the 1st byte is=144302088
enter the input values
2
3
4
4
1
2 is stored at address144302088
3 is stored at address144302092
4 is stored at address144302096
4 is stored at address144302100
1 is stored at address144302104
```

RESULT:

Thus the programme has been successfully completed and executed.

EX.NO: 2 DATE:	CPU SCHEDULING

AIM:

To write a program to display the cpu scheduling.

ALGORITHM:

STEP1: Start the process.

STEP2: Include all the necessary header files.

STEP3: Declare the necessary variables.

STEP4: Assign the number of jobs and burst time and waiting time using print statement.

STEP5: Using for statement schedule the cpu.

STEP6: Save and run the process in gedit.

STEP7: Stop the process.

CODING:

```
#include<stdio.h>
void main()
{
int n,i,j,bt[10],wt[10],twt,temp;
float awt;
printf("\n CPU SHEDULING\n");
printf("\n enter the no.of jobs");
scanf("%d",&n);
for(i=1;i<=n;i++)
{
printf("\n enter the burst time of job %d:",i);
scanf("%d",&bt[i]);
}
for(i=1;i<n;i++)
{
for(j=i+1;j<=n;j++)
{
if(bt[i]>bt[j])
{
temp=bt[i];
bt[i]=bt[j];
bt[j]=temp;
}
}
}
wt[0]=0;

twt=0;
```

```
for(i=1;i<=n;i++)
{
wt[i]=bt[i]+wt[i-1];
twt=twt+wt[i];
printf("\n WAITING TIME OF JOB %d IS:%d\n",i+1,wt[i]);
}
printf("\n the total waiting time is:%d\n",twt);
awt=twt/n;
printf("\n average waiting time is: %f \n",awt);
}
```

OUTPUT:

```
login004@ku030-OEM ~ $ cd Desktop
login004@ku030-OEM ~/Desktop $ cd 16csp012
login004@ku030-OEM ~/Desktop/16csp012 $ cc -c cpushedule.c
login004@ku030-OEM ~/Desktop/16csp012 $ cc -o cpushedule cpushedule.c
login004@ku030-OEM ~/Desktop/16csp012 $ ./cpushedule
CPU SHEDULING
enter the no.of jobs3
enter the burst time of job 1:3
enter the burst time of job 2:1
enter the burst time of job 3:5
WAITING TIME OF JOB 2 IS:1
WAITING TIME OF JOB 3 IS:4
WAITING TIME OF JOB 4 IS:9
the total waiting time is:14
average waiting time is:4.000000
```

RESULT:

Thus the programme has been successfully completed and executed.

EX.NO: 3 DATE:	DOS COMMAND COPY

AIM:

To write a program to simulate the dos command copy.

ALGORITHM:

STEP1: Start the process.

STEP2: Include all the necessary header files.

STEP3: Declare the pointer variable.

STEP4: Enter the name of the file to be copied with read-only permission.

STEP5: fgetc is used to get each character from the source file till the end of the file.

STEP6: fputc is used to put all the character from the source to destination

STEP7: close the source and destination file.

STEP8: Save and run the process.

STEP9: Stop the process.

CODING:

```
#include<stdio.h>
#include<stdlib.h>
void main()
{
char ch,srcfile[20],destnfile[20];
FILE *fdsrc,*fddstn;
printf("enter the name of the file copied:");
scanf("%s",&srcfile);
fdsrc=fopen(srcfile,"r");
printf("enter the name of thre destination file:");
scanf("%s",&destnfile);
fddstn=fopen(destnfile,"w");
while((ch=fgetc(fdsrc))!=EOF)
{
fputc(ch,fddstn);
}
printf("file copied successfully\n");
fclose(fdsrc);
fclose(fddstn);
}
```

OUTPUT:

login004@ku030-OEM ~/Desktop/16csp012 $ ./filecopy

enter the name of the file copied:

enter the name of thre destination file:malar

file copied successfully

RESULT:

Thus the programme has been successfully completed and executed.

EX.NO: 4 DATE:	SIGNAL HANDLER

AIM:

To write a Linux program to implement signal handling

ALGORITHM:

STEP1: Start the process.

STEP2: Include all the necessary header files.

STEP3: Using print statement specify the type of signal received.

STEP4: If the signal is receives it print the “hello world” if not means it goes to sleep mode.

STEP5: Save and run the process in.

STEP6: Stop the process.

CODING:

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<signal.h>
void ouch(int sig)
{
printf("\n OUCH ! - I GOT A SIGNAL %d\n",sig);
}
int main()
{
struct sigaction act;
act.sa_handler=ouch;
sigemptyset(&act.sa_mask);
act.sa_flags=0;
sigaction(SIGINT,&act,0);
while(1)
{
printf("\n WELCOME \n");
sleep(1);
}
return(0);
}
```

OUTPUT:

```
login004@ku030-OEM ~/Desktop/16csp012 $ cc -c signal.c
login004@ku030-OEM ~/Desktop/16csp012 $ cc -o signal signal.c
login004@ku030-OEM ~/Desktop/16csp012 $ ./signal
VENKY
VENKY
VENKY
VENKY
VENKY
^Z   //Terminates the command (cntrl+z)
[2]+
Stopped
./signal
```

RESULT:

Thus the programme has been successfully completed and executed.

EX.NO: 5 DATE:	THREADING

AIM:

To write a Linux program for threading.

ALGORITHM:

STEP1: Start the process.

STEP2: Include all the necessary header files.

STEP3: Enter the number of jobs and the burst time for each job.

STEP4: Using for loop specify the waiting time for each job.

STEP5: Finally perform the total waiting time and average waiting time for the entire process.

STEP6: Save and run the process in gedit.

STEP7: Stop the process.

CODING:

```
#include<stdio.h>
#include<string.h>
#include<pthread.h>
#include<stdlib.h>
#include<unistd.h>
pthread_t tid[2];
void *doSomeThing(void *arg)
{
unsigned long i=0;
pthread_t id=pthread_self();
if(pthread_equal(id,tid[0]))
{
printf("\n first thread processing\n");
}
else
{
printf("\n second thread processing\n");
}
return NULL;
}
int main()
{
int i=0;
int err;
while(i<2)
{
err=pthread_create(&(tid[i]),NULL,&doSomeThing,NULL);
if(err !=0)
```

```
printf("\n can't create thread:[%s]",strerror(err));
else
printf("\n thread created successfully\n");
i++;
}
sleep(5);
return 0;
}
```

OUTPUT:

```
login004@ku030-OEM ~/Desktop/16csp012 $ cc -c thread.c
login004@ku030-OEM ~/Desktop/16csp012 $ cc -o thread thread.c -pthread
login004@ku030-OEM ~/Desktop/16csp012 $ ./thread
thread created successfully
first thread processing
thread created successfully
second thread processing
```

RESULT:

Thus the programme has been successfully completed and executed.

EX.NO: 6 DATE:	DATE AND TIME USING TCP SOCKET

AIM:

To write a program to display date and time using **TCP** socket

ALGORITHM:

STEP1: Start the process.

STEP2: Include all the necessary header files for client and server separately.

STEP3: Create a socket connection using the port number 8000.

STEP4: If the socket is open the client and server is binded successfully.

STEP5: For() statement is used as null loop.

STEP6: Repeat the same for the client process and display the message read from client.

STEP7: Save and run the process.

STEP8: Stop the process.

CODING:

Client.c

```
#include <netinet/in.h>
#include <sys/socket.h>
#include <stdio.h>
void main()
{
struct sockaddr_in sa,cli;
int n,sockfd;
int len;
char buff[100];
sockfd=socket(AF_INET,SOCK_STREAM,0);
if(sockfd<0)
{
printf("\n Enter in socket");
}
printf("\n socket is opened");
bzero(&sa,sizeof(sa));
sa.sin_family=AF_INET;
sa.sin_port=htons(3600);
if(connect(sockfd,(struct sockaddr*)&sa,sizeof(sa))<0)
{
printf("\n connected successfully");
if(n=read(sockfd,buff,sizeof(buff))<0)
{
printf("\n Error in reading");
}
else
{
```

```
printf("\n Message read%s",buff);
}
}
}
```

Server.c

```
#include<netinet/in.h>
#include<sys/socket.h>
#include<stdio.h>
#include<string.h>
#include<time.h>
#define PORT 8000
Void main()
{
Struct sockaddr_in sa;
Struct sockaddr_in cli;
Int sock_fd,coonfd;
Int len,ch;
Char len,ch;
Char str[100];
Time_t tick;
Sockfd=socket(AF_INET_SOCK_STREAM,0);
If(socket<0)
{
Printf(“error in socket\n”);
}
Else
Printf(“socket opened”);
Bzero(&sa,sizeof(sa));
Sa.sin_port=htons(8000);
```

```
Sa.sin_addrs_addr=hton(0);
If(bind(sockfd,(struct sock addr*)&sa,sizeof(sa)y<0)
{
Printf("\n error in bind");
}
Else
Printf("\n binded successfully");
Listen(sockfd,sa);
For *(;;)
{
Len=sizeof(ch);
Connfd=accept(sockfd,(struct sockaddr*)cli,elen);
Printf("accepted");
Tick=time(NULL);
Snprintf("str,sizeof(str),"%s",(time(&tick));
Printf("%s",str);
Write(connfd,str,100);
}
}
```

OUTPUT:

server

Socket is opened

Binded successfully

Accepted July wed 26 2:06:20 2017

client

Socket is opened

Connected successfully

Message read July wed 26 2:06:20 2017

RESULT:

Thus the program has been successfully completed and executed.

EX.NO: 7 DATE:	DEADLOCK

AIM:

To write a Linux program to display process deadlock state.

ALGORITHM:

STEP1: Start the process.

STEP2: Include all the necessary header files.

STEP3: Declare all the necessary variables and arrays

STEP4: Enter the number of process, resource amount in each resource type.

STEP5: Allocate the resource for each process.

STEP6: Allocate the claim process for getting how much process for getting how much the resource needed.

STEP7: Specify how much process and resource is used and allocated.

STEP8: After that check the deadlock for the process if the process is considerable resource is it safe.

STEP9: Save and run the process.

STEP10: Stop the process.

CODING:

```
#include<stdio.h>
#include<stdlib.h>
void main()
{
int num_p,num_re;
int all[10][10],re[10],cl[10][10],used[10],av[10];
int more[10];
int i=0,check;
int ap,ar;
int cp,cr;
int pi,ci;
int j=0;
printf("enter the number of process:");
scanf("%d",&num_p);
printf("enter the number of resource type:");
scanf("%d",&num_re);
printf("enter the amount in each resource type:");
for(i=0;i<num_re;i++)
{
printf("\n R%d:",i+1);
scanf("%d",&re[i]);
}
for(ap=0;ap<num_p;ap++)
{
printf("enter the allocation of resource process%d:",ap+1);
for(ar=0;ar<num_p;ar++)
scanf("%d",&all[ar][ap]);
}
```

```
for(cp=0;cp<num_p;cp++)
{
printf("enter claim for process%d:",cp+1);
for(cr=0;cr<num_re;cr++)
scanf("%d",&cl[cr][cp]);
}
for(i=0;i<num_re;i++)
{
used[i]=0;
for(j=0;j<num_p;j++)
used[i]=used[i]+all[i][j];
av[i]=re[i]-used[i];
}
for(pi=0;pi<num_p;pi++)
{
check=0;
for(ci=0;ci<num_re;ci++)
{
more[ci]=cl[ci][pi]-all[ci][pi];
if(more[ci]<=av[ci])
check++;
}
if(check==0)
printf("\n \t DEADLOCK\n\n");
else
printf("\n safe");
}
}
```

OUTPUT:

```
login004@ku030-OEM ~ $ cd Desktop
login004@ku030-OEM ~/Desktop $ cd 16csp012
login004@ku030-OEM ~/Desktop/16csp012 $ cc -c deadlock.c
login004@ku030-OEM ~/Desktop/16csp012 $ cc -o deadlock deadlock.c
login004@ku030-OEM ~/Desktop/16csp012 $ ./deadlock
enter the number of process:1
enter the number of resource type:1
enter the amount in each resource type: R1:2
enter the allocation of resource process1:2
enter claim for process1:1
safe
login004@ku030-OEM ~/Desktop/16csp012 $ ./deadlock
enter the number of process:1
enter the number of resource type:1
enter the amount in each resource type: R1:2
enter the allocation of resource process1:1
enter claim for process1:3
DEADLOCK
login004@ku030-OEM ~/Desktop/16csp012 $
```

RESULT:

Thus the program has been successfully completed and executed.

EX.NO: 8 DATE:	LOCKING AND UNLOCKING FILES

AIM:

To write a Linux program for locking and unlocking files.

ALGORITHM:

STEP1: Start the process.

STEP2: Include all the necessary header files.

STEP3: Declare the necessary variables.

STEP4: Based on the priority assign the job to be executed.

STEP5: Perform the process, process time and the priority based on the if and for statement.

STEP6: Save and run the process.

STEP7: Stop the process.

CODING:

```
#include<stdio.h>
#include<string.h>
#include<unistd.h>
#include<sys/fcntl.h>
#define O_WRONLY 01
int main(int argc,char* argv[])
{
char* file=argv[1];
int fd;
struct flock lock;
printf("opening %s\n",file);
fd=open(file,O_WRONLY);
printf("locking\n");
memset(&lock,0,sizeof(lock));
lock.l_type=F_WRLCK;
fcntl(fd,F_SETLKW,&lock);
printf("locked;hit enter the lock.....");
getchar();
printf("unlocking");
lock.l_type=F_UNLCK;
fcntl(fd,F_SETLKW,&lock);
close(fd);
return 0;
}
```

OUTPUT:

```
login004@ku030-OEM ~/Desktop/16csp012 $ cc -c lock.c
login004@ku030-OEM ~/Desktop/16csp012 $ cc -o lock lock.c
login004@ku030-OEM ~/Desktop/16csp012 $ ./lock
q.txt
opening q.txt
locking
locked;
hit enter the lock.....
```

RESULT:

Thus the program has been successfully completed and executed.

EX.NO: 9 DATE:	TO DISPLAY LOGGED IN USER

AIM:

To write a Linux program to display logged in user.

ALGORITHM:

STEP1: Start the process.

STEP2: Include all the necessary header files.

STEP3: Declare the pointer variable and allocate the buffer.

STEP4: The size of character must be of 10 characters.

STEP5: getLogin() is used what type of user is logging in

STEP6: Save and run the process.

STEP7: Stop the process.

CODING:

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
main()
{
char* buf;
buf=(char*)malloc(10* sizeof(char));
buf=getlogin();
printf("\n%s\n",buf);
}
```

OUTPUT:

```
login004@ku030-OEM ~/Desktop/16csp012 $ cc -c userinfo.c
login004@ku030-OEM ~/Desktop/16csp012 $ cc -o userinfo  userinfo.c
login004@ku030-OEM ~/Desktop/16csp012 $ ./userinfo
login004
```

RESULT:

Thus the program has been successfully completed and executed.

EX.NO: 10 DATE:	DISPLAY DATE AND TIME USING UDP SOCKET

AIM:

To write a program to create display the date and time chat between client and server.

ALGORITHM:

STEP1: Start the process.

STEP2: Include all the necessary header files for client and server separately.

STEP3: Define MAX as constant and PORT which refer a value and create a structure with an object.

STEP4: Define a variable for server address and socfd is used as built in function.

STEP5: bzero is define and fetches the value from the server.

STEP6: Define the IP address and execute using null loop.

STEP7: Save and run the process.

STEP8: Stop the process.

CODING:

UDP.C

```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
#include<sys/socket.h>
#include<sys/types.h>
#include<netinet/in.h>
#include<time.h>
int main()
{
int sfd,r,bi,port;
char buff[1024];
struct sockaddr_in servaddr,cliaddr;
socklen_t clilen;
sfd=socket(AF_INET,SOCK_DGRAM,0);
if(sfd==-1)
{
perror("socket");
return 0;
}
printf("\n enter the port no:");
scanf("%d",&port);
printf("the port no is:%d\n",port);
servaddr.sin_family=AF_INET;
servaddr.sin_port=htons(port);
servaddr.sin_addr.s_addr=INADDR_ANY;
bi=bind(sfd,(struct sockaddr*)&servaddr,sizeof(servaddr));
if(bi==-1)
```

```
{
perror("bind()");
return 0;
}
clilen=sizeof(cliaddr);
r=recvfrom(sfd,buff,sizeof(buff),0,(struct sockaddr*)&cliaddr,&clilen);
buff[r]=0;
time_t ticks;
ticks=time(NULL);
snprintf(buff,sizeof(buff),"%24s\r\n",ctime(&ticks));
sendto(sfd,buff,sizeof(buff),0,(struct sockaddr*)&cliaddr,sizeof(cliaddr));
exit(0);
return 0;
}
```

UDP1.C

```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
#include<sys/socket.h>
#include<sys/types.h>
#include<netinet/in.h>
int main()
{
int listenfd,port,r;

char buff[1024];
struct sockaddr_in servaddr,cliaddr;
socklen_t servlen;
```

```
listenfd=socket(AF_INET,SOCK_DGRAM,0);
if(listenfd==-1)
{
perror("socket");
return 0;
}
printf("\n enter the port no:");
scanf("%d",&port);
printf("the port no is:%d\n",port);
servaddr.sin_family=AF_INET;
servaddr.sin_port=htons(port);
servaddr.sin_addr.s_addr=INADDR_ANY;
sendto(listenfd,buff,sizeof(buff),0,(structsockaddr*)&servaddr,sizeof(serr);
r=recvfrom(listenfd,buff,sizeof(buff),0,(structsockaddr*)&servaddr,&sern;
buff[r];
printf("\n the time recieved from the server:%s\n",buff);
exit(0);
return 0;
}
```

OUTPUT:

Server

Enter the port no:8000

Time received from server

Sat Aug 23 2:06:20 2017

client

Enter the port no:8000

Sat Aug 23 2:06:20 2017

RESULT:

Thus the program has been successfully completed and executed.