Fork()

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

int main()
{
    int a=0, b=1, n=a+b,i,ii;
    pid_t pid;

    printf("Enter the number of a Fibonacci
        Sequence:\n");
    scanf("%d", &ii);

    if (ii < 0)
        printf("Please enter a non-negative
            integer!\n");
    else
```

```
{
        pid = fork();
        if (pid == 0)
        {
                printf("Child is producing the Fibonacci
                        Sequence...\n");
                printf("%d %d",a,b);
                for (i=2;i<ii;i++)
                {
                        n=a+b;
                        printf("%d ", n);
                        a=b;
                        b=n;
                }
                printf("Child ends\n");
        }
        else
```

```
        {
            printf("Parent is waiting for child to
                complete...\n");
            wait(NULL);
            printf("Parent ends\n");
        }
    }
    return 0;
}
```

fork working

```
#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>
int main()
{
pid_t pid, pid1;
/* fork a child process */
pid = fork();
if (pid < 0) { /* error occurred */
```

```
fprintf(stderr, "Fork Failed");
return 1;
}
else if (pid == 0) { /* child process */
pid1 = getpid();
printf("child: pid = %d",pid); /* A */
printf("child: pid1 = %d",pid1); /* B */
}
else { /* parent process */
pid1 = getpid();
printf("parent: pid = %d",pid); /* C */
printf("parent: pid1 = %d",pid1); /* D */
wait(NULL);
}
return 0;
}
```

if actual pids of the parent and child are 2600 and 2603 respectively, then

A=0
B=2603
C=2603
D=2600

createprocess()

```
#include <stdio.h>
#include <windows.h>
int main(VOID)
{
STARTUPINFO si;
PROCESS INFORMATION pi;
// allocate memory
ZeroMemory(&si, sizeof(si));
si.cb = sizeof(si);
ZeroMemory(&pi, sizeof(pi));
// create child process
if (!CreateProcess(NULL,"C:\\WINDOWS\\
system32\\fib.exe", NULL,NULL, FALSE, 0, NULL,
NULL,&si,&pi))
{
fprintf(stderr, "Create Process Failed");
return -1;
}else{
int a=0, b=1, n=a+b,i,ii;
printf("Enter the number of a Fibonacci
Sequence:\n");
scanf("%d", &ii);

if (ii < 0)
printf("Please enter a non-negative integer!\n");
```

```
else
{printf("%d %d",a,b);
for (i=2;i<ii;i++)
{
n=a+b;
printf("%d ", n);
a=b;
b=n;
}
printf("Parent is waiting for child to complete...\
n");
printf("Child ends\n");
printf("Parent ends\n");
}

}
// parent will wait for the child to complete
WaitForSingleObject(pi.hProcess, 5000);
printf("Child Complete");
// close handles
CloseHandle(pi.hProcess);
CloseHandle(pi.hThread);
}
```

Remote procedure call (Socket)

DATA SERVER

```
import java.net.*;
import java.io.*;
public class DateServer {
public static void main(String[] args) {
try { ServerSocket sock = new ServerSocket(6013);
// now listen for connections
while (true) { Socket client = sock.accept();
PrintWriter pout = new
PrintWriter(client.getOutputStream(), true);
 // write the Date to the socket
pout.println("Some random jokes.\n");
// close the socket and resume
// listening for connections
client.close();
}
}
catch (IOException ioe){
System.err.println(ioe);
}
}
}
```

DATA CLIENT

```
import java.net.*;
```

```
import java.io.*;
public class DateClient
{
public static void main(String[] args) {
try {
//make connection to server socket
Socket sock = new Socket("127.0.0.1",6013);
InputStream in = sock.getInputStream();
BufferedReader bin = new
BufferedReader(new InputStreamReader(in));
// read the date from the socket
String line;
while ( (line = bin.readLine()) != null)
System.out.println(line);
// close the socket connection
sock.close();
}
catch (IOException ioe) {
System.err.println(ioe);
}
}
}
```

code for Echo (i.e. server returns what client sends)

ECHO SERVER

```
import java.io.*;
import java.net.*;
class echos {
   public static void main(String args[]) throws
Exception
   {
      String echoin;
      ServerSocket svrsoc;
      Socket soc;
      BufferedReader br;
      try
      {
         svrsoc = new ServerSocket(2000);
         soc = svrsoc.accept();
         br = new BufferedReader (new
InputStreamReader(soc.getInputStream()));
         PrintStream ps = new
PrintStream(soc.getOutputStream());
         System.out.println("Connected for
echo:");
         while((echoin=br.readLine())!=null)
         {
            if(echoin.equals("end"))
```

```
            {
                System.out.println("Client
disconnected");
                br.close();
                break;
            }
            else
                ps.println(echoin);
        }
    }
    catch(UnknownHostException e)
    {
        System.out.println(e.toString());
    }
    catch(IOException ioe)
    {
        System.out.println(ioe);
    }
  }
}
```

ECHO CLIENT

```
import java.io.*;
import java.net.*;
```

```
class echoc {
public static void main(String args[]) throws
Exception
{
    String sockin;
    try
    {
        Socket csock = new
Socket(InetAddress.getLocalHost(),2000);
        BufferedReader br = new
BufferedReader(new
InputStreamReader(System.in));
        BufferedReader br_sock = new
BufferedReader(new
InputStreamReader(csock.getInputStream()));
        PrintStream ps = new
PrintStream(csock.getOutputStream());
        System.out.println("Start echoing... type
'end' to terminate");
        while((sockin=br.readLine())!=null)
        {
            ps.println(sockin);
            if(sockin.equals("end"))
                break;
            else
```

```
                System.out.println("echoed from
server:"+br_sock.readLine());

        }
    }
catch(UnknownHostException e)
{
    System.out.println(e.toString());

}
catch(IOException ioe)
{
    System.out.println(ioe);
}
}
}
```

posix shared memory

```
#include <stdio.h>
#include <sys/shm.h>
#include <sys/stat.h>
int main()
{
```

```
/* the identifier for the shared memory segment
*/
int segment id;
/* a pointer to the shared memory segment */
char *shared memory;
/* the size (in bytes) of the shared memory
segment */
const int size = 4096;
/* allocate a shared memory segment */
segment id = shmget(IPC PRIVATE, size, S IRUSR |
S IWUSR);
/* attach the shared memory segment */
shared memory = (char *) shmat(segment id,
NULL, 0);
/* write a message to the shared memory
segment */
sprintf(shared memory, "Hi there!");
/* now print out the string from shared memory
*/
printf("*%s\n", shared memory);
/* now detach the shared memory segment */
shmdt(shared memory);
/* now remove the shared memory segment */
shmctl(segment id, IPC RMID, NULL);
return 0;
}
```

POSIX Thread

```
#include <pthread.h>
#include <stdio.h>
int sum; /* this data is shared by the thread(s) */
void *runner(void *param); /* the thread */
int main(int argc, char *argv[])
{
pthread_t tid; /* the thread identifier */
pthread_attr_t attr; /* set of thread attributes */
if (argc != 2) {
fprintf(stderr,"usage: a.out <integer value>\n");
return -1;
}
if (atoi(argv[1]) < 0) {
fprintf(stderr,"%d must be >= 0\n",atoi(argv[1]));
return -1;
}
/* get the default attributes */
pthread_attr init(&attr);
/* create the thread */
pthread create(&tid,&attr,runner,argv[1]);
/* wait for the thread to exit */
pthread join(tid,NULL);
printf("sum = %d\n",sum);
}
```

```
/* The thread will begin control in this function */
void *runner(void *param)
{
int i, upper = atoi(param);
sum = 0;
for (i = 1; i <= upper; i++)
sum += i;
pthread exit(0);
}
```

JAVA Thread

(for creating threads in a Java program implement the Runnable interface. [studied in java lab 3rd sem])

```
public interface Runnable
{
public abstract void run();
}
```

Windows thread

```
#include <windows.h>
#include <stdio.h>
DWORD Sum; /* data is shared by the thread(s) */
```

```
/* the thread runs in this separate function */
DWORD WINAPI Summation(LPVOID Param)
{
DWORD Upper = *(DWORD*)Param;
for (DWORD i = 0; i <= Upper; i++)
Sum += i;
return 0;
}
int main(int argc, char *argv[])
{
DWORD ThreadId;
HANDLE ThreadHandle;
int Param;
/* perform some basic error checking */
if (argc != 2) {
fprintf(stderr,"An integer parameter is required\
n");
return -1;
}
Param = atoi(argv[1]);
if (Param < 0) {
fprintf(stderr,"An integer >= 0 is required\n");
return -1;
}
// create the thread
ThreadHandle = CreateThread(
```

```
NULL, // default security attributes
0, // default stack size
Summation, // thread function
&Param, // parameter to thread function
0, // default creation flags
&ThreadId); // returns the thread identifier
if (ThreadHandle != NULL) {
// now wait for the thread to finish
WaitForSingleObject(ThreadHandle,INFINITE);
// close the thread handle
CloseHandle(ThreadHandle);
printf("sum = %d\n",Sum);
}
}
```