

```

// Vahe Karamian - CS 431 - Project 1 - Dr. Lee
// Filename: pcbNode.h
// 07/20/2001

#include <iostream>
#include <string>

using namespace std;

#ifndef PCBNODE_H
#define PCBNODE_H

class PCBNODE
{
    friend class PCB;

public:
    PCBNODE( string state,           // set process STATE
            string id,              // set process ID
            int mode );             // set process MODE
    PCBNODE( );

    string getState( );             // return process STATE
    string getID( );                // return process ID
    int  getMode( );                // return process MODE

    void setMode( int mode );        // set process MODE
    void setState( string state );   // set process STATE
    void setID( string id );         // set process ID

    PCBNODE* copy1( );              // return pointer to PCBNODE
    PCBNODE  copy2( );              // make copy of PCBNODE

private:
    string STATE;                   // process STATE
    string ID;                       // process ID
    int    MODE;                     // process MODE

    PCBNODE* nextPtr;                // pointer to next NODE
    PCBNODE* previousPtr;            // pointer to previous NODE
};
#endif

```

```

// Vahe Karamian - CS 431 - Project 1 - Dr. Lee
// Filename: pcbNode.cpp
// 07/20/2001

#include "pcbNode.h"

PCBNODE::PCBNODE( )
{
    STATE = " ";                    // set process STATE
    ID = " ";                        // set process ID
    MODE = 5;                        // set process MODE

    nextPtr = NULL;                 // set next pointer to NULL
    previousPtr = NULL;             // set previous pointer to NULL
}

PCBNODE::PCBNODE( string state, string id, int mode )
{
    STATE = state;                  // set process STATE
    ID = id;                        // set process ID
    MODE = mode;                    // set process MODE

    nextPtr = NULL;                 // set next pointer to NULL
    previousPtr = NULL;             // set previous pointer to NULL
}

// return PCBNODE STATE
string PCBNODE::getState( ) { return STATE; }

// return PCBNODE ID
string PCBNODE::getID( ) { return ID; }

// return PCBNODE MODE
int  PCBNODE::getMode( ) { return MODE; }

```

```

// set PCBNODE STATE
void PCBNODE::setState( string state )
{
    STATE = state;
}

// set PCBNODE ID
void PCBNODE::setID( string id )
{
    ID = id;
}

// set PCBNODE MODE
void PCBNODE::setMode( int mode ) { MODE = mode; }

// return PCBNODE pointer
PCBNODE* PCBNODE::copy1( )
{
    PCBNODE* node = new PCBNODE( STATE, ID, MODE );
    return( node );
}

// return PCBNODE
PCBNODE PCBNODE::copy2( )
{
    PCBNODE node( STATE, ID, MODE );
    return( node );
}

// Vahe Karamian - CS 431 - Project II - Dr. Lee
// Filename: semaphore.h

#include "pcbNode.h"
#include <queue>

template< class T>
class Semaphore
{
public:
    Semaphore( );           // Constructor
    ~Semaphore( );        // Destructor

    void P( T &s );       // P operation
    void V( T &s );       // V operation

private:
    int value;           // BSEMA or CSEMA
    string id;           // process id

    queue<T> sq;         // Semaphore queue
};

// Semaphore constructor
template<class T>
Semaphore<T>::Semaphore( ) { value = 1; }

// Semaphore destructor
template<class T>
Semaphore<T>::~~Semaphore( ) { }

// Semaphore P operation
template<class T>
void Semaphore<T>::P( T &s )
{
    if(value > 0)         // if value > 0
    {
        value--;         // decrement value
        s.setState("cs"); // set process state to CS
    }
    else                 // otherwise
    {
        id = s.getID( ); // get process id
        cout<<id<<" (BW)"; // display a message
        s.setState("blk"); // set process state to BLOCKED
        sq.push( s );     // push the process into the queue
    }
}

```

```

// Semaphore V operation
template<class T>
void Semaphore<T>::V( T &s )
{
    value++;                // increment value
    s.setState("run");     // set process state to run

    if(!sq.empty())       // if queue not empty
    {
        sq.pop( );        // get next process from queue
    }
}

// Vahe Karamian - CS 431 - Project II - Dr. Lee
// Filename: project2a.cpp

#include "sema.h"

int main()
{
    Semaphore<PCBNODE> s;    // Create semaphore

    PCBNODE P1, P2;         // Create Process 1 and Process 2
    P1.setID( "P1" );       // Set process id
    P1.setState("run");     // Set process state

    P2.setID( "P2" );       // Set process id
    P2.setState("run");     // Set process state

    // Initialize Time, P1 instruction counter, P2 instruction counter
    // P1 critical section counter, P2 critical section counter
    int Time = 0, p1 = 0, p2 = 0, cs1 = 0, cs2=0;

    while(Time < 1000)
    {
        Time++;             // Increment clock
        if(P1.getState() == "run") p1++; // Increment P1 counter if P1 state is runnable
        if(P2.getState() == "run") p2++; // Increment P2 counter if P2 state is runnable

        // Print out table
        cout<<"Time: "<<Time<<" P1: "<<p1<<" cs1: "<<cs1<<" P2: "<<p2<<" cs2: "<<cs2<<" "; //<<endl;

        if(p1 > 3)          // if P1 counter > 4 enter CS
        {
            if(P1.getState() != "cs" ) // check process P1 state if not CS
            {
                s.P(P1);             // check semaphore
                if(P1.getState()=="cs") // set P1 state to CS
                    cs1++;           // P1 critical section counter incremented
            }
            else                // if P1 state is CS
            {
                cs1++;             // P1 critical section counter incremented
                if(cs1 > 4)        // if P1 CS > 4
                {
                    s.V(P1);       // P1 CS complete, release semaphore

                    if( P2.getState() == "blk" ) // if P2 state is BLOCKED, UNBLOCK it
                        s.P( P2 ); // check semaphore

                    cs1 = 0;        // reset P1 CS counter
                    p1 = 0;        // reset P1 counter
                }
            }
        }

        if(p2 > 5)          // if P2 counter > 6 enter CS
        {
            if(P2.getState() != "cs") // check process P2 state if not CS
            {
                s.P(P2);             // check semaphore
                if(P2.getState()=="cs") // set P1 state to CS
                    cs2++;           // P2 critical section counter incremented
            }
            else                // if P2 state is CS
            {

```

```

cs2++; // P2 critical section counter incremented
if(cs2 > 7) // if P2 CS > 6
{
    s.V(P2); // P2 CS complete, release semaphore

    if( P1.getState() == "blk" ) // if P1 state is BLOCKED, UNBLOCK it
        s.P( P1 ); // check semaphore

    cs2 = 0; // reset P2 CS counter
    p2 = 0; // reset P2 counter
}
}
}
cout<<endl;
}
return( 0 );
}

```