Chapter#07 codes:  
  
1- Bounded Buffer aka Producer Consumer:  
while(TRUE)

{

    wait(stick[i]);

    /\*

        mod is used because if i=5, next

        chopstick is 1 (dining table is circular)

    \*/

    wait(stick[(i+1) % 5]);

    /\* eat \*/

    signal(stick[i]);

    signal(stick[(i+1) % 5]);

    /\* think \*/

}

2-Dining Philosopher:

monitor DiningPhilosophers {

    enum { THINKING, HUNGRY, EATING } state[5]; // Array to track the state of each philosopher

    condition self[5]; // Array of condition variables, one for each philosopher

    // Method to simulate a philosopher picking up chopsticks and eating

    method pickup(int i) {

        state[i] = HUNGRY;

        test(i); // Check if the philosopher can start eating

        if (state[i] != EATING)

            self[i].wait(); // If not, wait until they can eat

    }

    // Method to simulate a philosopher putting down chopsticks

    method putdown(int i) {

        state[i] = THINKING;

        test((i + 4) % 5); // Check if the left neighbor can start eating

        test((i + 1) % 5); // Check if the right neighbor can start eating

    }

    // Method to check if a philosopher can start eating

    method test(int i) {

        if (state[(i + 4) % 5] != EATING && state[i] == HUNGRY && state[(i + 1) % 5] != EATING) {

            state[i] = EATING;

            self[i].signal(); // Signal the philosopher to start eating

        }

    }

    // Method for a philosopher to simulate thinking

method think(int i) {

        // Simulate thinking

    }

    // Method for a philosopher to simulate eating

    method eat(int i) {

        // Simulate eating

    }

}

// Philosopher thread function

void philosopher(int i) {

    while (true) {

        DiningPhilosophers.pickup(i); // Philosopher picks up chopsticks

        DiningPhilosophers.eat(i); // Philosopher eats

        DiningPhilosophers.putdown(i); // Philosopher puts down chopsticks

        DiningPhilosophers.think(i); // Philosopher thinks

    }

}

3- Reader-Writer:

Using semaphores:  
semaphore mutex = 1; // Controls access to shared variables

semaphore wrt = 1;   // Controls writing access

int readCount = 0;   // Number of readers reading the shared resource

// Writer process

writer() {

    wait(wrt);        // Wait for writing access

    // Write to the shared resource

    signal(wrt);      // Release writing access

}

// Reader process

reader() {

    wait(mutex);      // Gain access to modify readCount

    readCount++;      // Increment the number of readers

    if (readCount == 1)  // If this is the first reader

        wait(wrt);    // Prevent writers from accessing the shared resource

    signal(mutex);    // Release access to modify readCount

    // Read from the shared resource

    wait(mutex);      // Gain access to modify readCount

    readCount--;      // Decrement the number of readers

    if (readCount == 0)  // If there are no more readers

        signal(wrt);  // Allow writers to access the shared resource

    signal(mutex);    // Release access to modify readCount

}

Using Monitors:

monitor ReaderWriter {

    int readers = 0; // Number of active readers

    bool writing = false; // Indicates whether a writer is writing

    condition noReaders; // Condition variable for no readers

    condition noWriters; // Condition variable for no writers

    // Method for readers to access the shared resource

    method startRead() {

        if (writing)

            noReaders.wait(); // Wait if a writer is writing

        readers++;

    }

    // Method for readers to release access to the shared resource

    method endRead() {

        readers--;

        if (readers == 0)

            noWriters.signal(); // Signal waiting writers if no readers

    }

    // Method for writers to access the shared resource

    method startWrite() {

        if (writing || readers > 0)

            noWriters.wait(); // Wait if a writer or reader is active

        writing = true;

    }

    // Method for writers to release access to the shared resource

    method endWrite() {

        writing = false;

        noReaders.signal(); // Signal waiting readers

        noWriters.signal(); // Signal waiting writers

    }

}

// Reader thread function

void reader() {

    while (true) {

        ReaderWriter.startRead(); // Reader starts reading

        // Read from the shared resource

        ReaderWriter.endRead(); // Reader finishes reading

        // Perform other tasks

    }

}

// Writer thread function

void writer() {

    while (true) {

        ReaderWriter.startWrite(); // Writer starts writing

        // Write to the shared resource

        ReaderWriter.endWrite(); // Writer finishes writing

        // Perform other tasks

    }

}









