Coral S. Schmidt #148830

Monitors make solving the producer-consumer a little easier. Mutual exclusion is achieved by placing the critical section of a program inside a monitor. In the code below, the critical sections of the producer and consumer are inside the monitor Producer/Consumer. Once inside the monitor, a process is blocked by the Wait and Signal primitives if it cannot continue.

Complete the following implementation writing the missing statements (see the comments in green color).

Note: Use the wait and signal methods with the appropriate condition variables. If variable count is equal to N, the buffer is full, if variable count is equal to 0, the buffer is empty. Use the object ProducerConsumer

monitor ProducerConsumer

condition full, empty;

int count;

procedure enter();

{

// if buffer is full, block

put item(widget); // put item in buffer

count = count + 1; // increment count of full slots

// if buffer was empty, wake consumer

}

procedure remove();

{

// if buffer is empty, block

remove\_item(widget); // remove item from buffer

count = count - 1; // decrement count of full slots

// if buffer was full, wake producer

}

count = 0;

end monitor;

Producer();

{

while (TRUE)

{

make\_item(widget); // make a new item

// call enter function in monitor

}

}

Consumer();

{

while (TRUE)

{

// call remove function in monitor

consume\_item; // consume an item

}

}

***Complete code:***

monitor ProducerConsumer {

    condition full, empty;

**int** count;

    procedure enter() {

        if (count == N)                     *// if buffer is full, block*

            full.wait();                    *// wait until buffer is not full*

        put\_item(widget);                   *// put item in buffer*

        count = count + 1;                  *// increment count of full slots*

        if (count == 1)                     *// if buffer was empty, wake consumer*

            empty.signal();

    }

    procedure remove() {

        if (count == 0)                     *// if buffer is empty, block*

            empty.wait();                   *// wait until buffer is not empty*

        remove\_item(widget);                *// remove item from buffer*

        count = count - 1;                  *// decrement count of full slots*

        if (count == N - 1)                 *// if buffer was full, wake producer*

            full.signal();

    }

    count = 0;

}

Producer() {

    while (TRUE) {

        make\_item(widget);                  *// make a new item*

        ProducerConsumer.enter();           *// call enter function in monitor*

    }

}

Consumer() {

    while (TRUE) {

        ProducerConsumer.remove();          *// call remove function in monitor*

        consume\_item(widget);               *// consume an item*

    }

}