```
        for( i=1 ; i<=2000 ; i++ ) {
                wait(consumed);
                n++;
                signal(produced);
        }
}

/*------------------------------------------------------------------------
 * cons2  -  Print n 2000 times, waiting for it to be produced
 *------------------------------------------------------------------------
 */
void    cons2(
          sid32         consumed,
          sid32         produced
        )
{
        int32   i;

        for( i=1 ; i<=2000 ; i++ ) {
                wait(produced);
                printf("n is %d \n", n);
                signal(consumed);
        }
}
```

2.9 Semaphores And Mutual Exclusion

Semaphores serve another important purpose, *mutual exclusion*. Two or more processes engage in mutual exclusion when they cooperate so that only one of them obtains access to a shared resource at a given time. For example, suppose two executing processes each need to insert items into a shared linked list. If they access the list concurrently, pointers can be set incorrectly. Producer–consumer synchronization does not handle the problem because the two processes do not alternate accesses. Instead, a mechanism is needed that allows either process to access the list at any time, but guarantees mutual exclusion so that one process will wait until the other finishes.

To provide mutual exclusion for use of a resource such as a linked list, the processes create a single semaphore that has an initial count of 1. Before accessing the shared resource, a process calls *wait* on the semaphore, and calls *signal* after it has completed access. The calls to *wait* and *signal* can be placed at the beginning and end of the functions designed to perform the update, or they can be placed around the lines of code that access the shared resource. We use the term *critical section* to refer to the code that cannot be executed by more than one process at a time.