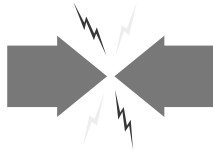
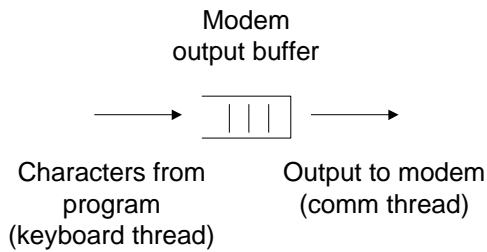


Process Synchronization

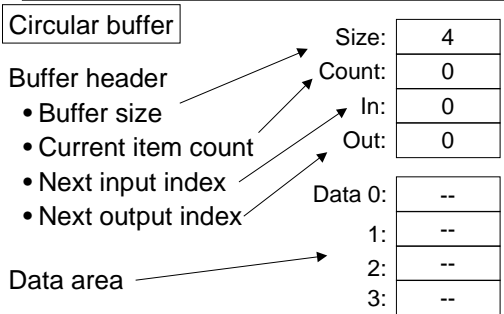
- **Concurrent processes**
 - Executing at the same time
- **Shared data access**
- **Possible conflict**
 - Inconsistent data state



Shared Buffer Problem



Shared Buffer Data Structure



Shared Buffer Example

Circular buffer

- Insert 'O'
- Insert 'p', 's', 'y'
- Remove next ('O')
- Remove next ('p')
- Insert 's'

Size:	4
Count:	3
In:	1
Out:	2
Data 0:	's'
1:	--
2:	's'
3:	'y'

Shared Buffer Class

```
class Buffer
{
  unsigned int Size;
  unsigned int Count;
  unsigned int In;
  unsigned int Out;
  char * Data;
  ...
};
```

Size:	4
Count:	0
In:	0
Out:	0
Data 0:	--
1:	--
2:	--
3:	--

Shared Buffer Initialization

```
Buffer::Buffer (unsigned int size)
{
  Size = size;
  Count = 0;
  In = 0;
  Out = 0;
  Data = new char[size];
}
```

Buffer destructor = exercise for student

Shared Buffer Input

```
void Buffer::PutBuf (char ch)
{
    while (Count >= Size)
        /* wait */;

    Data[In] = ch;
    In = (In++) % Size;
    Count++;
}
```

Shared Buffer Output

```
char Buffer::GetBuf ()
{
    char ch = 0;
    while (Count == 0) /* wait */;

    ch = Data[buf->Out];
    Out = (Out++) % Size;
    Count--;

    return ch;
}
```

Shared Buffer Access

Possible access conflict	Producer (input)	Consumer (output)
Size:	X	X
Count:	✓	✓
In:	✓	X
Out:	X	✓
Data:	✓	✓

??

All items must remain in a consistent state.

Counter Access Detail

Producer

Consumer

Count++;

Count--;

LDAA Count
INCA
STAA Count

LDAA Count
DECA
STAA Count

Counter Access Conflict

Producer

Consumer

Count = 3

LDAA Count
INCA

STAA Count

LDAA Count
DECA
STAA Count

Count = ??

Critical Section

Also known as "critical region"

- **Exclusive access needed**
 - To data (e.g., "Count")
 - Simultaneous access produces inconsistent state
- **Mutual exclusion**
 - Accessing process excludes others

Critical Section Sequence

Entry section	Request permission to enter critical section
Critical section	Perform operations on shared data
Exit section	May grant permission to other processes
Remainder section	Rest of program (not related to critical section)

Critical Section Solution

- **Mutual exclusion**
- **Progress**
 - Only non-remainder processes involved
 - Access not postponed indefinitely
- **Bounded waiting**
 - Limit on other procs "ahead"



Two-Process Solution - Algorithm 1

```

volatile int turn=0;
int MyPID = 0; /* or 1 for 2nd */
for (;;)
{
  while (turn != MyPID) /* wait */;
  /* critical section */
  turn = (1 - MyPID);
  /* remainder section */
}

```

What is this?

Page 159

Identify the sections Does this work?

Two-Process Solution - Algorithm 2

Page 159

```

volatile int flag[2] = { 0,0 };
int MyPID = 0; /* or 1 for 2nd */
for (;;)
{
  flag[MyPID] = 1;
  while (flag[1-MyPID]) /* wait */;
  /* critical section */
  flag [MyPID] = 0;
  /* remainder section */
}

```

How about this one?

Two-Process Solution - Algorithm 3

Page 161

```

volatile int turn=0;
volatile int flag[2] = { 0,0 };
int MyPID = 0; /* or 1 for 2nd */
for (;;)
{
  flag[MyPID] = 1;
  turn = (1 - MyPID);
  while ((flag[1-MyPID]) &&
        (turn != MyPID)) /* wait */;
  /* critical section */
  flag [MyPID] = 0;
  /* remainder section */
}

```

Multiple (>2) Processes

- **Processes “takes a number”**
 - No lower than any other “number”
- **Compare number with others**
 - Enter critical section if lowest
- **Might get “equal” number**
 - Break tie with process ID

Lamport's Bakery Algorithm, page 162
