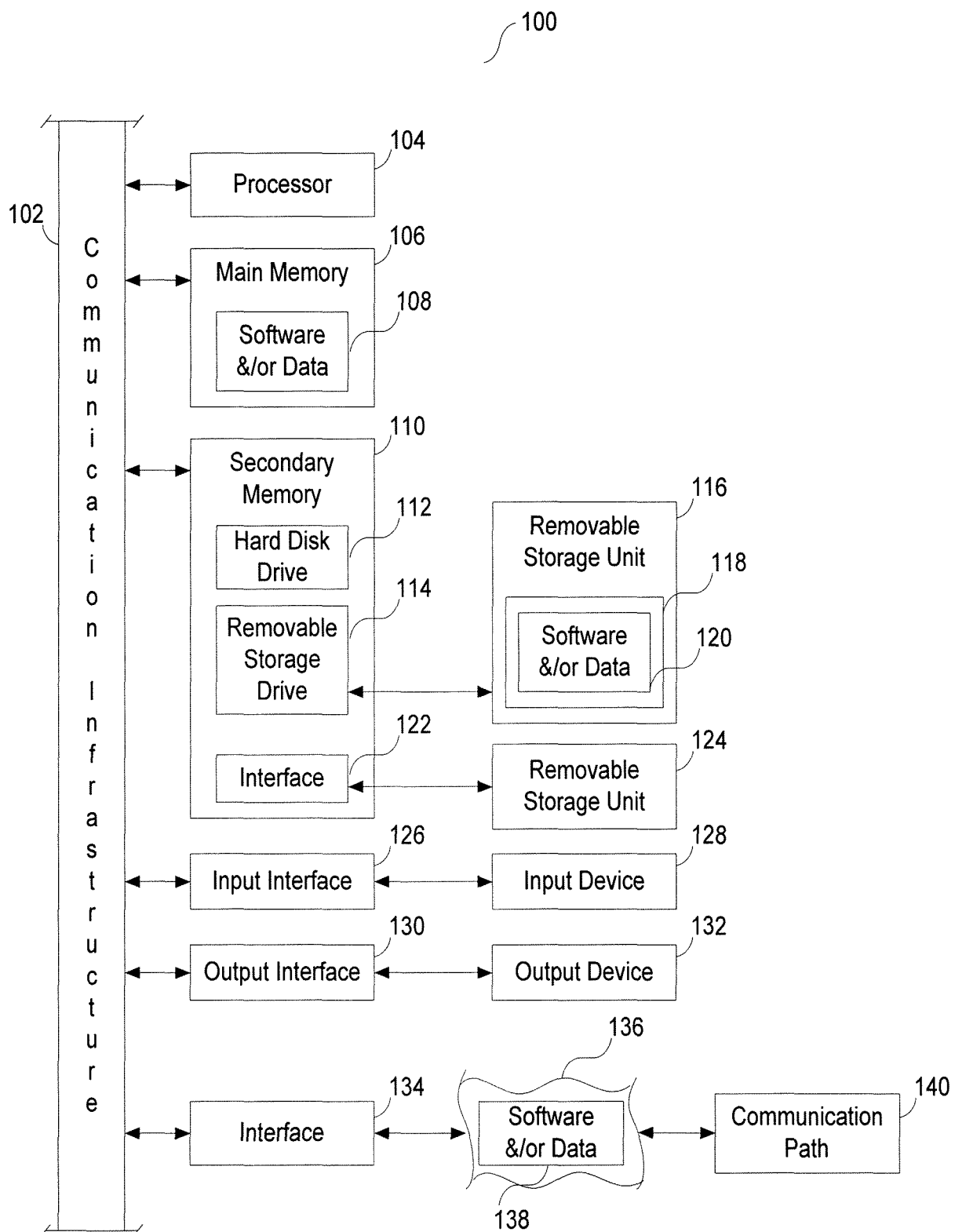


**FIG. 1**

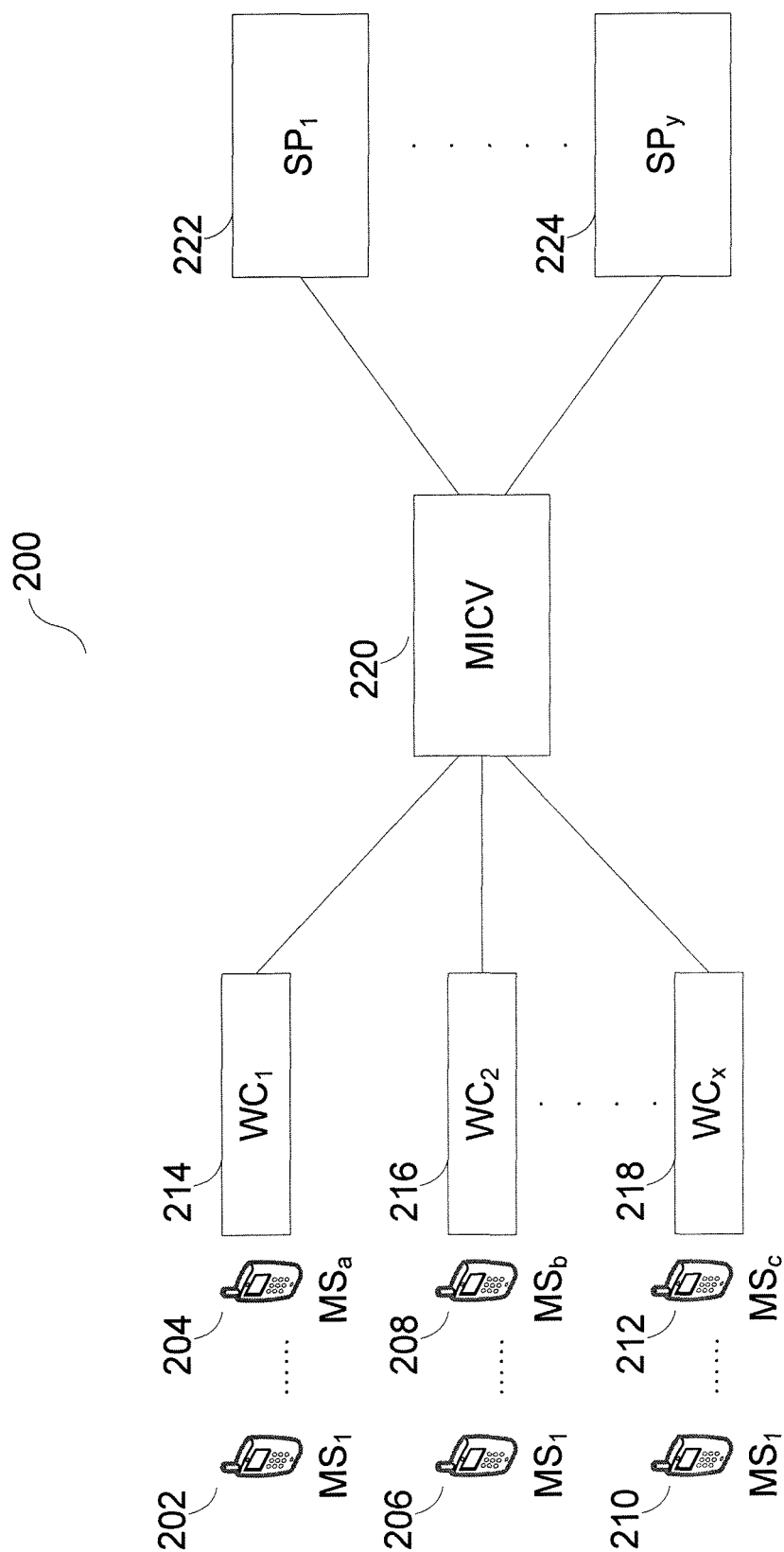


FIG. 2

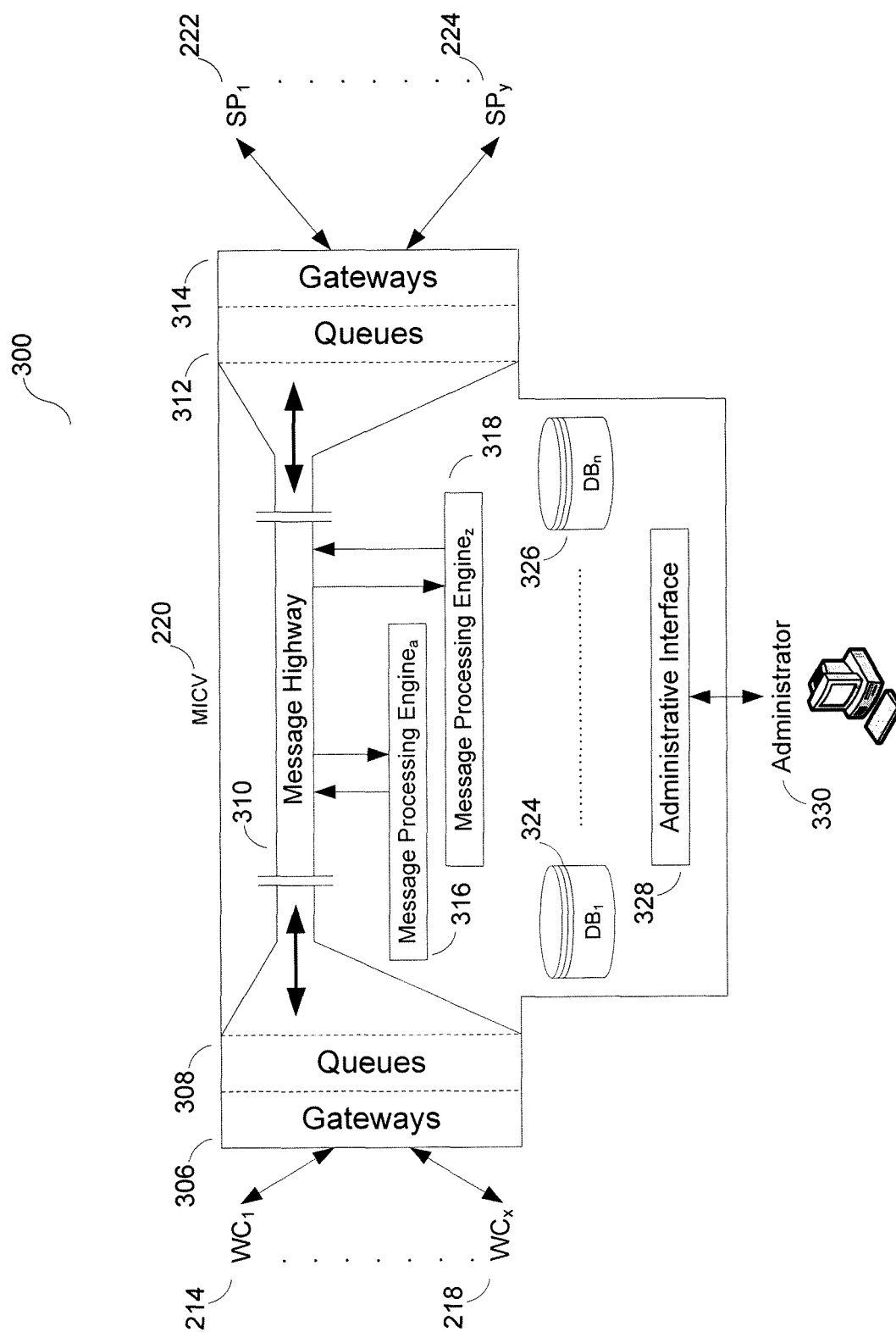


FIG. 3

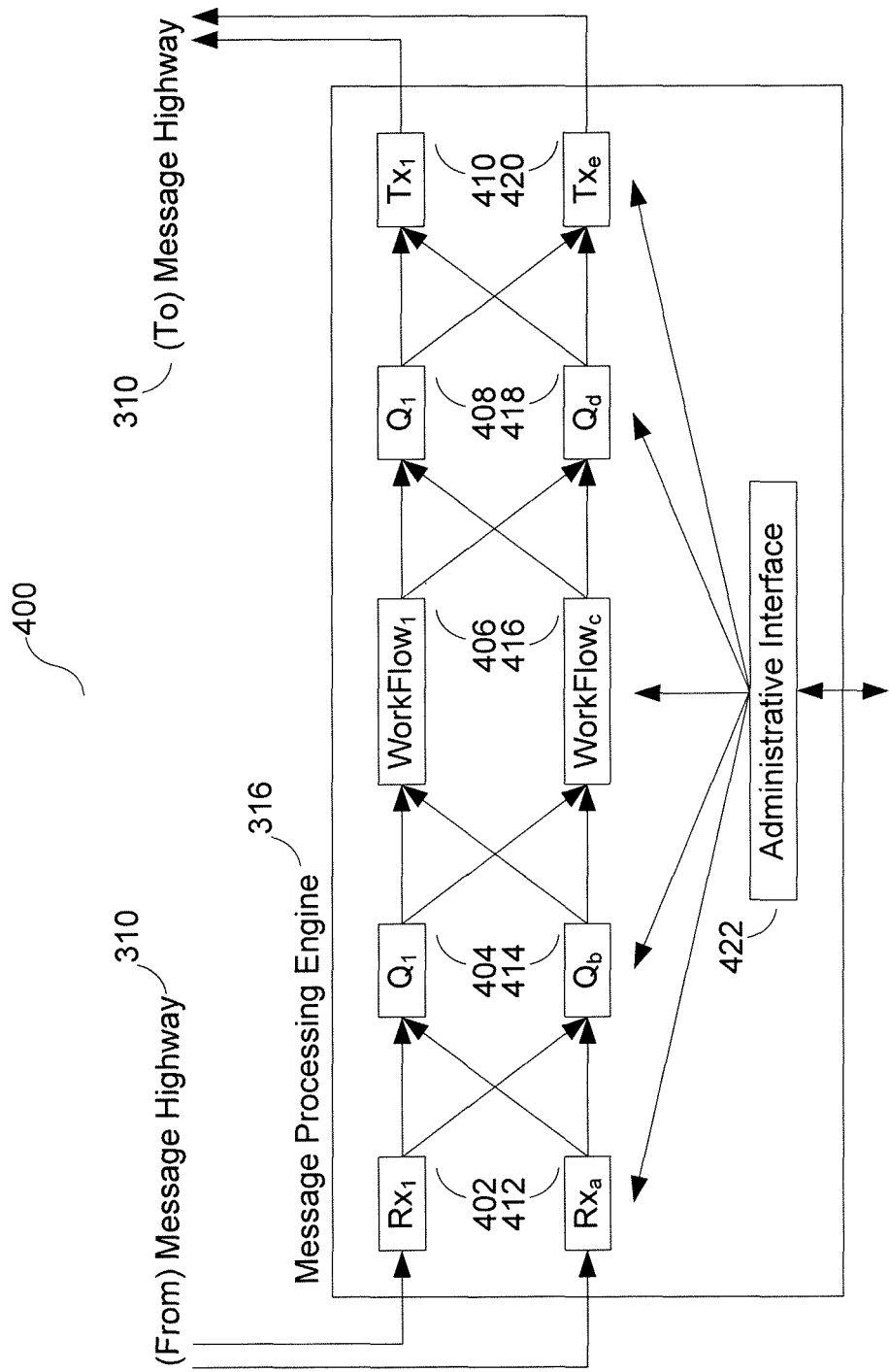
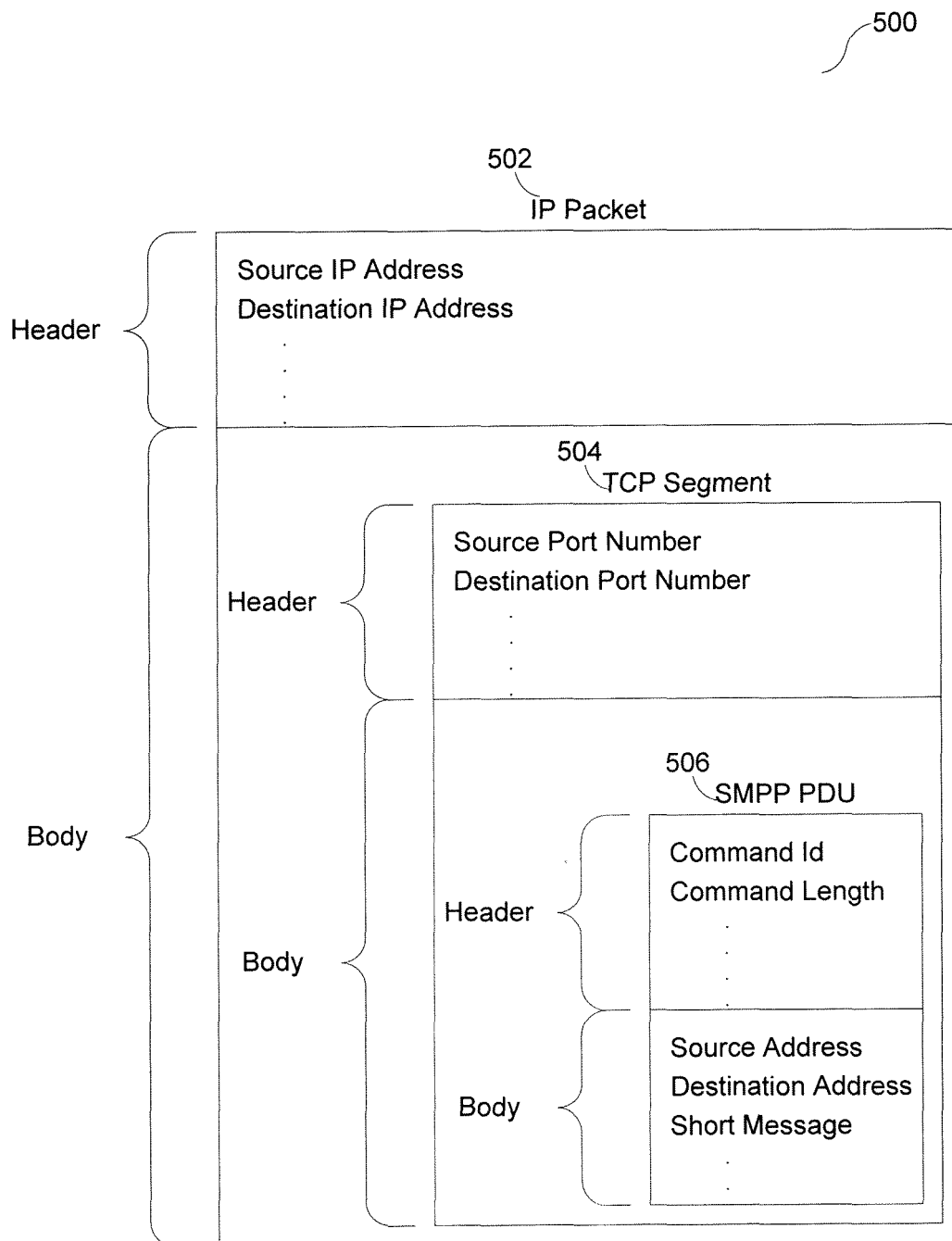
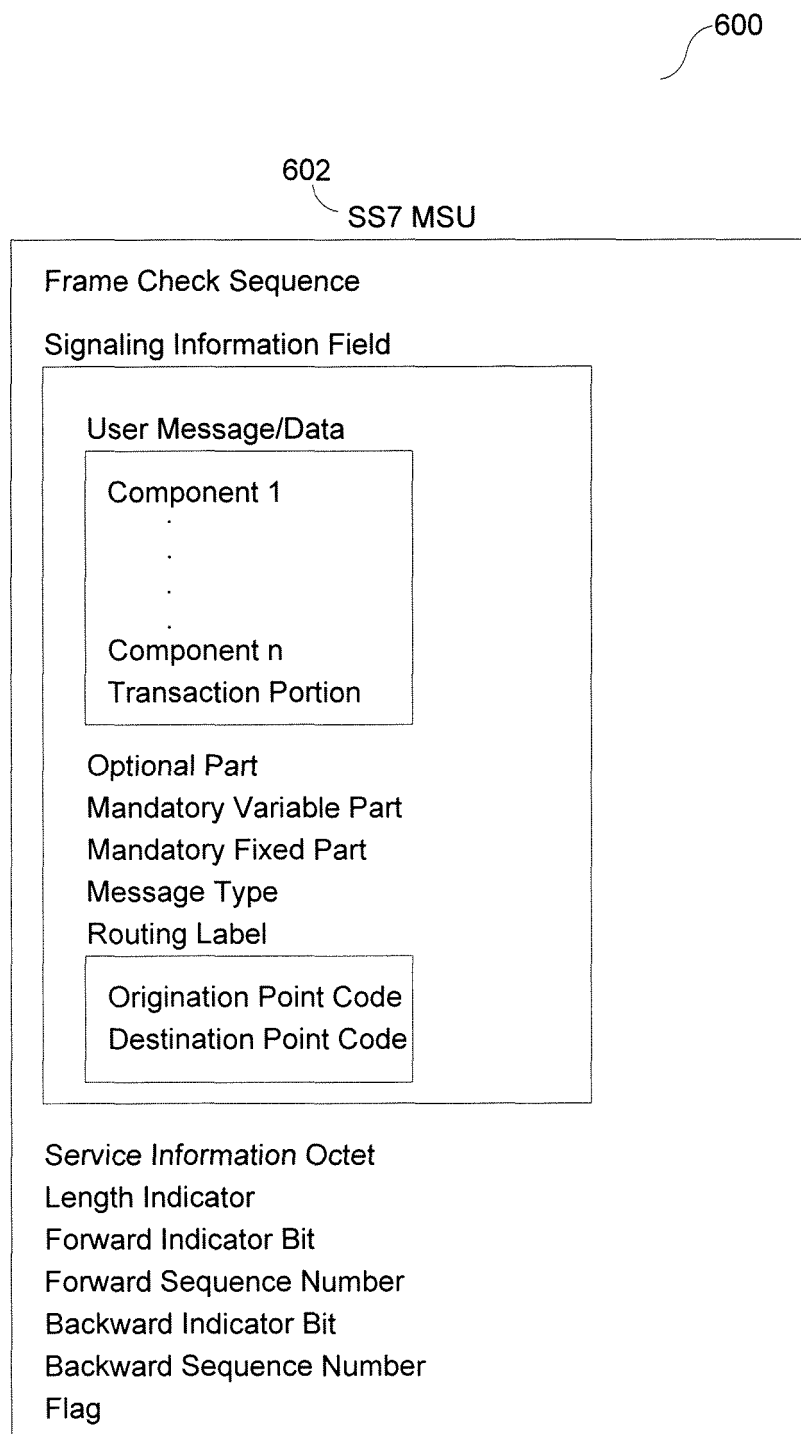
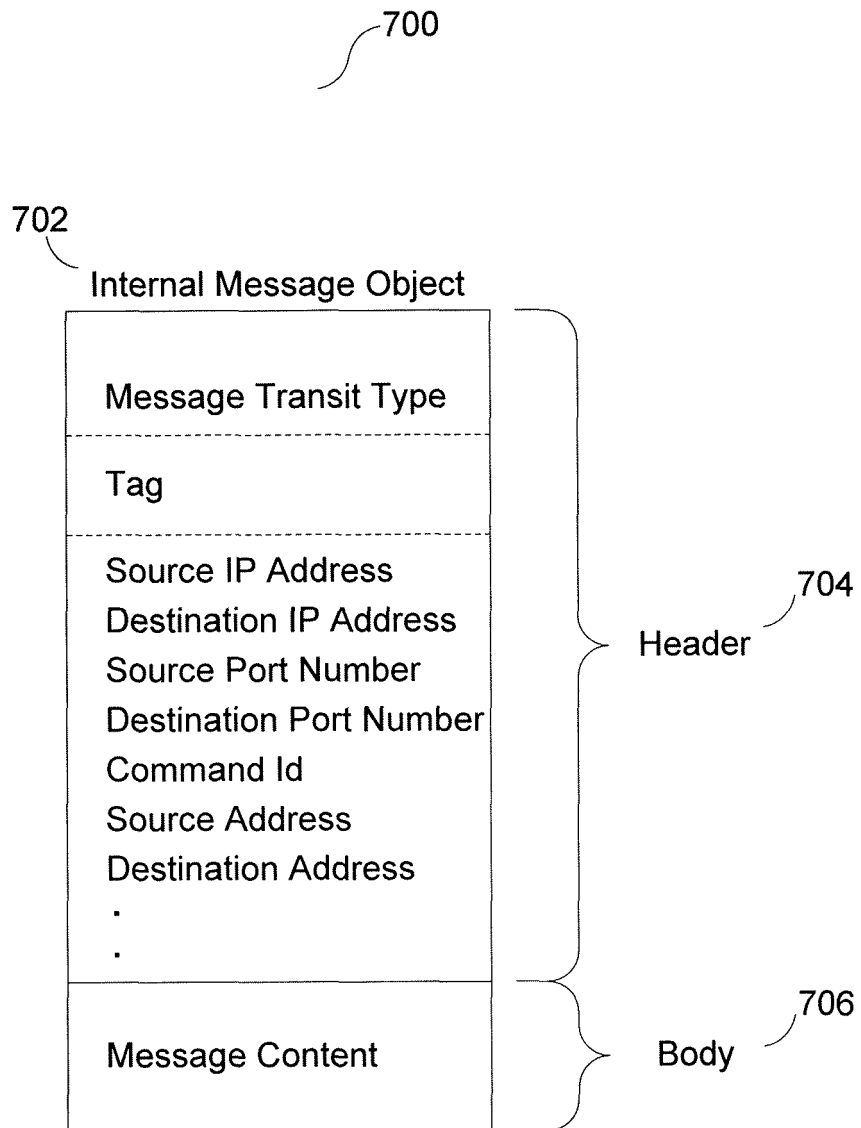
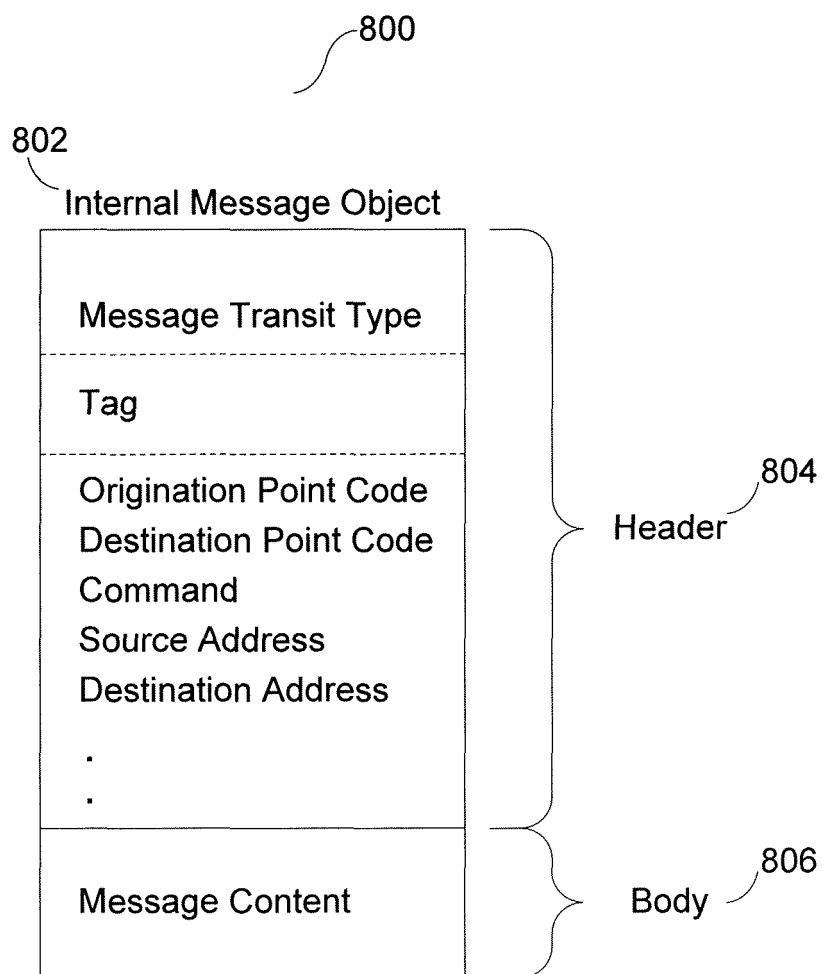


FIG. 4

**FIG. 5**

**FIG. 6**

**FIG. 7**

**FIG. 8**

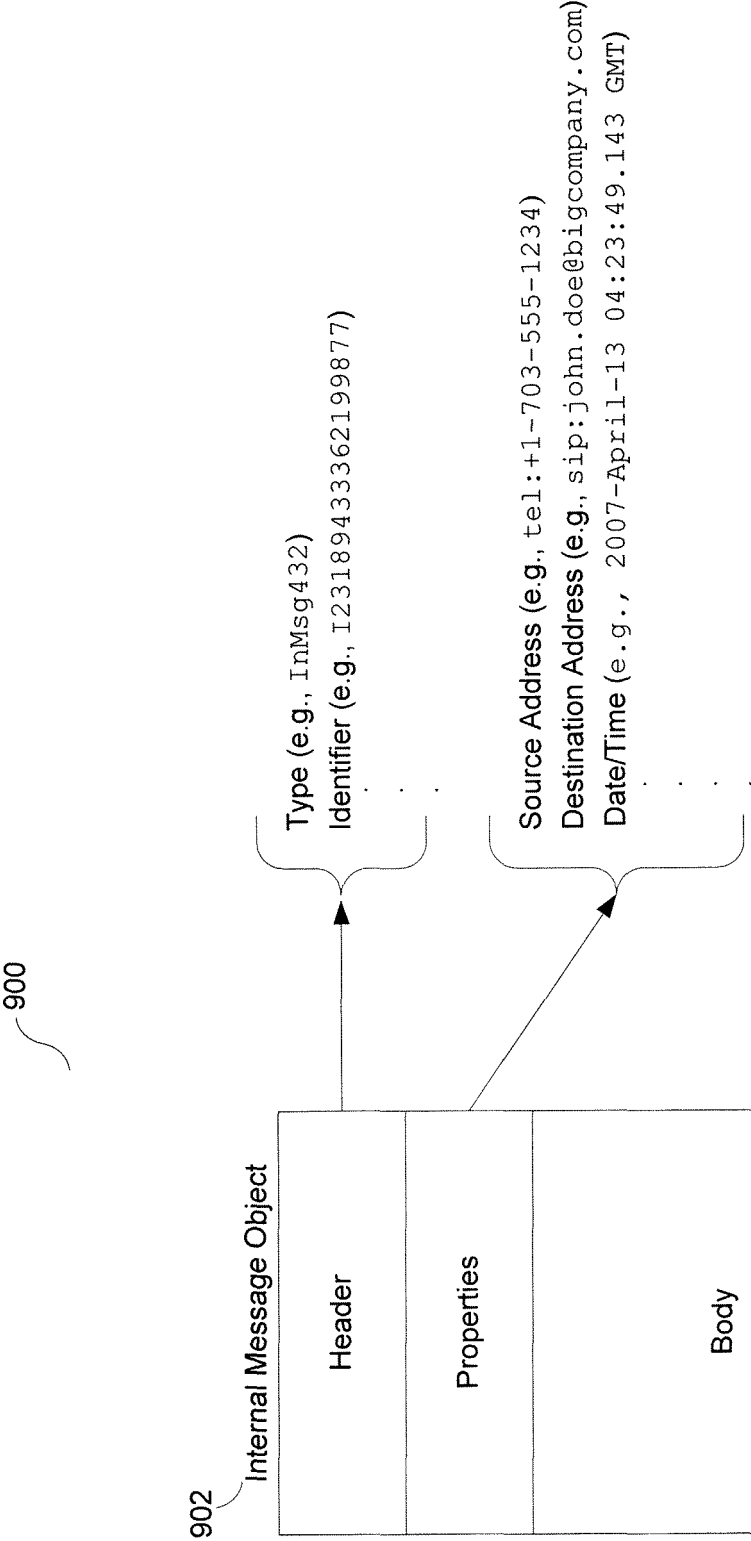
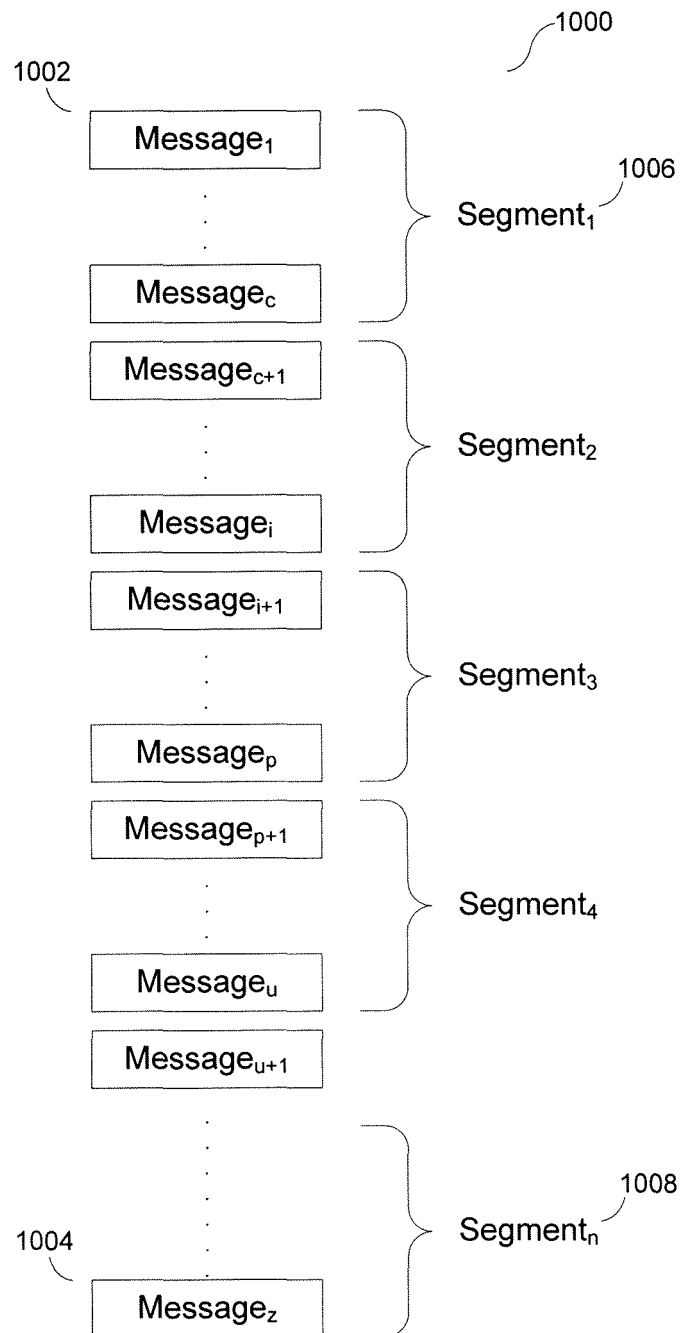


FIG. 9

**FIG. 10**

1100

Segment	Number of Messages
1	435
2	435
3	435
4	435
Total	1740

FIG. 11a

1102

Segment	Number of Messages
1	290
2	290
3	290
4	290
5	290
6	290
Total	1740

FIG. 11b

1200

Segment	Number of Messages
1	265
2	415
3	390
4	150
Total	1220

FIG. 12a

1202

Segment	Number of Messages
1	163
2	235
3	143
4	216
5	92
6	170
7	201
Total	1220

FIG. 12b

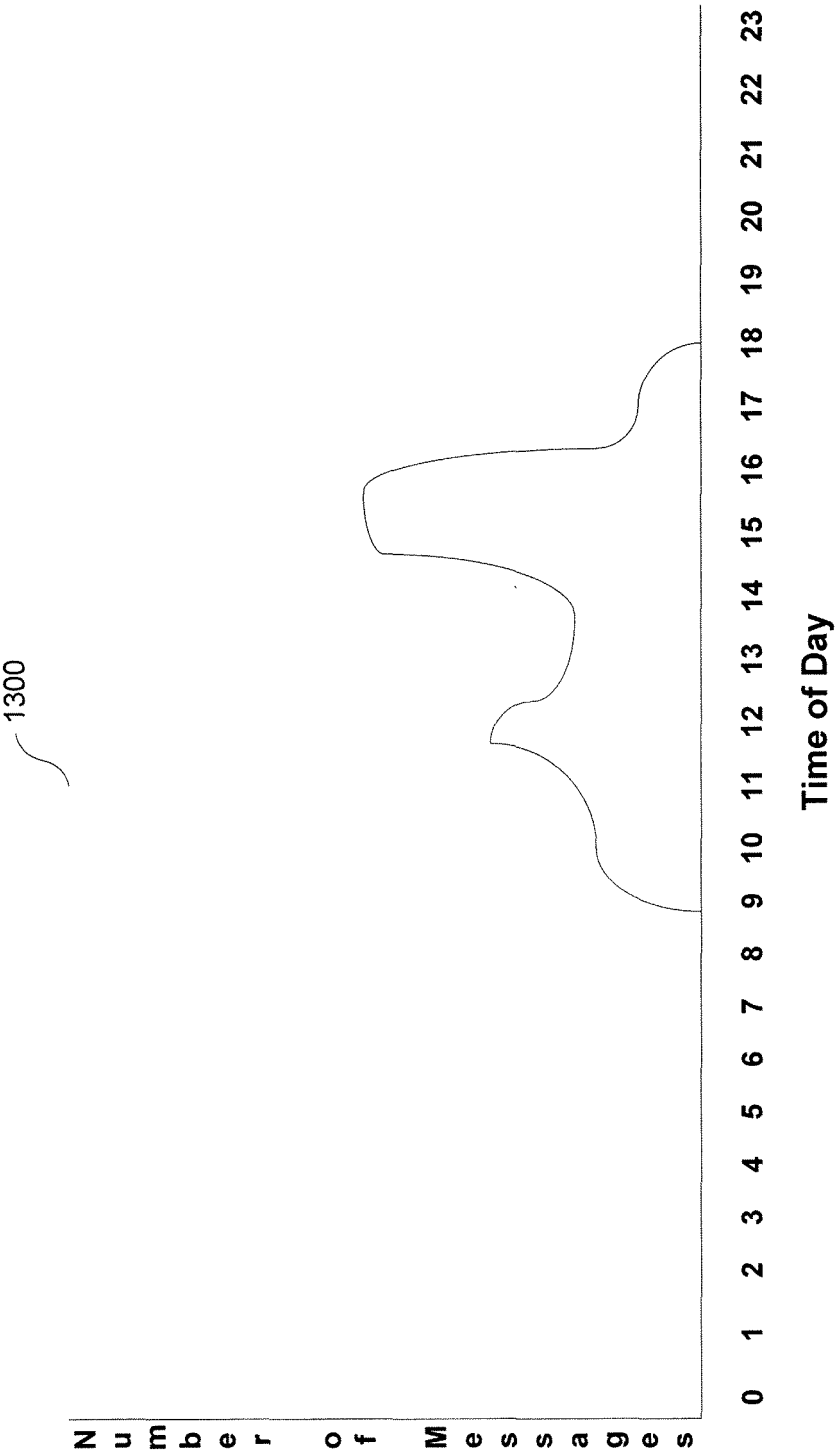
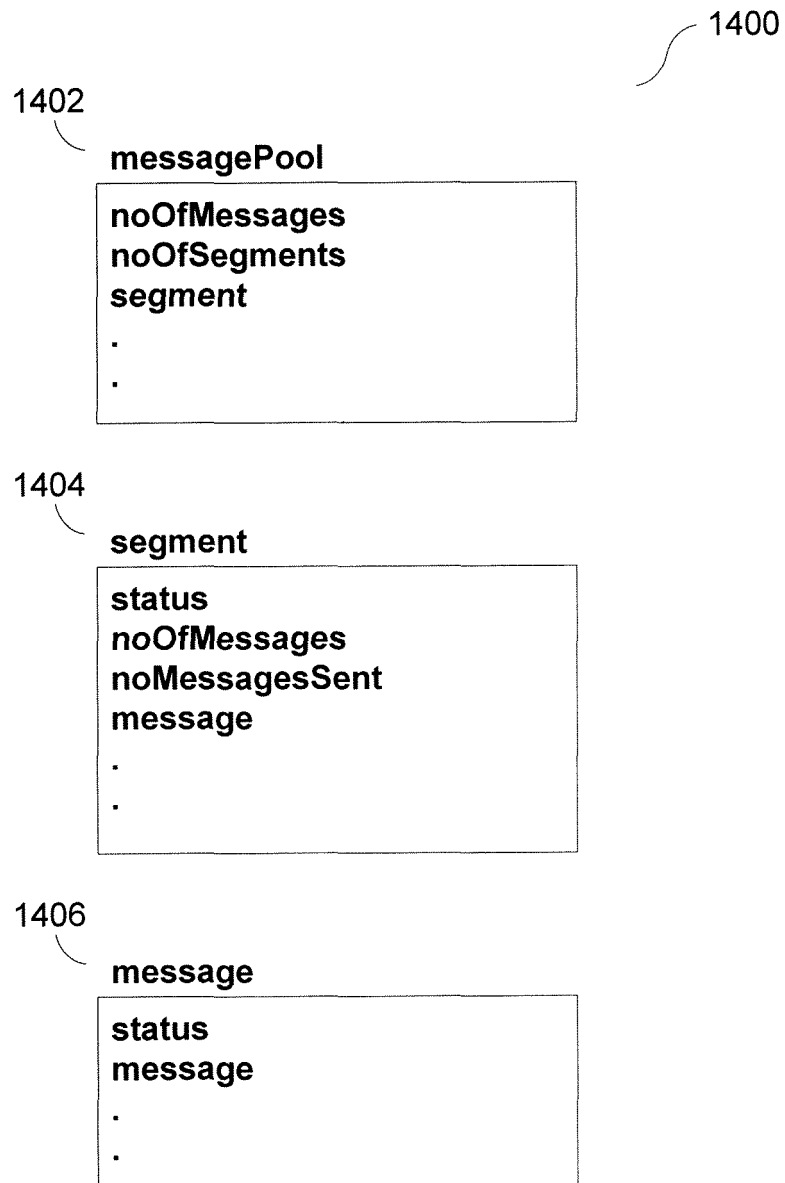


FIG. 13

**FIG. 14**

1500

1502

```

ProcessPool
{
    /* Perform any initialization activities */
    Initialize ();

    /* Process each segment in the message pool */
    for (segmentNo = 1; segmentNo <= messagePool.noOfSegments; segmentNo++)
    {
        /* Process each message in segment segmentNo */
        for (messageNo = 1; messageNo <= messagePool.segment [segmentNo].noOfMessages; messageNo++)
        {
            /* Send message messageNo in segment segmentNo */
            send message messagePool.segment [segmentNo].message [messageNo].message;

            /* Optionally wait for and optionally process message send ACK */

            /* Wait (after message is sent) */
            pause (time1);
        }

        /* Wait (after segment is done) */
        pause (time2);
    }

    /* Perform any cleanup activities */
    Finalize ();
}

```

FIG. 15

1602

```

ProcessPool
{
    /* Perform any initialization activities */
    Initialize ();

    /* Spin through all of the segments in the message pool */
    for (segmentNo = 1; segmentNo <= messagePool.noOfSegments; segmentNo++)
    {
        /* A segment starts out as closed with no messages sent */
        messagePool.segment [segmentNo].status = CLOSED;
        messagePool.segment [segmentNo].noOfMessagesSent = 0;
        for (messageNo = 1; messageNo <= messagePool.segment [segmentNo].noOfMessages;
            messageNo++)
            messagePool.segment [segmentNo].message [messageNo].status = UNSENT;
    }

    /* Start processing */
    notDone = TRUE;
    noOfChildThreads = 0;
    while (notDone)
    {
        /* Is the window closed? */
        if (window.status == CLOSED)
        {
            /* Yes, stop any sending activity ... */
            for (segmentNo = 1; segmentNo <= messagePool.noOfSegments; segmentNo++)
                if (messagePool.segment [segmentNo].status == WORKING)
                {
                    /* Cleanly terminate child thread with ProcessSegment (segmentNo);
                     * noOfChildThreads--;
                     * messagePool.segment [segmentNo].status = CLOSED;
                     */
                }

            /* ... and loop (wait for the window to open) */
            continue;
        }
    }
}

```

1600

Continued on Figure 16b

FIG. 16a

Continued from Figure 16a

1604

```

/* The window is open. Any segments with unsent messages that are not */
/* currently being worked? */
for (segmentNo = 1; segmentNo <= messagePool.noOfSegments; segmentNo++)
    if (messagePool.segment [segmentNo].status == CLOSED)
    {
        /* Yes, can we launch another child thread?
        if (noOfChildThreads < maxNoOfChildThreads)
        {
            /* Yes, start working this segment */
            messagePool.segment [segmentNo].status = WORKING;
            launch child thread with ProcessSegment (segmentNo);
            noOfChildThreads++;
        }
    }

/* Has a segment that is currently being worked sent all of its messages? */
for (segmentNo = 1; segmentNo <= messagePool.noOfSegments; segmentNo++)
    if (messagePool.segment [segmentNo].status == WORKED)
    {
        /* Yes, mark the segment as done */
        messagePool.segment [segmentNo].status == DONE;
        Cleanly terminate child thread with ProcessSegment (segmentNo);
        noOfChildThreads--;
    }

```

Continued on Figure 16c

FIG. 16b

Continued from Figure 16b

1606

```

/* Has a segment that is currently being worked reached a messages-sent */
/* threshold such that we can now open another segment? */
foundSegment = FALSE;
for (segmentNo = 1; segmentNo <= messagePool.noOfSegments; segmentNo++)
{
    if (messagePool.segment [segmentNo].status == WORKING)
    {
        if ( (messagePool.segment [segmentNo].noOfMessagesSent /
            messagePool.segment [segmentNo].noOfMessages) >= messageSentThreshold)
        {
            /* Yes, start working another segment */
            for (segmentNo = 1; segmentNo <= messagePool.noOfSegments; segmentNo++)
            {
                if (messagePool.segment [segmentNo].status == CLOSED)
                {
                    messagePool.segment [segmentNo].status = WORKING;
                    launch child thread with ProcessSegment (segmentNo);
                    noOfChildThreads++;

                    foundSegment = TRUE;
                    break;
                }
            }

            if (foundSegment == TRUE)
                break;
        }
    }

    /* Are we all done? */
    noOfSegmentsDone = 0;
    for (segmentNo = 1; segmentNo <= messagePool.noOfSegments; segmentNo++)
    {
        if (messagePool.segment [segmentNo].status == DONE)
            noOfSegmentsDone++;
    }
    if (noOfSegmentsDone == messagePool.noOfSegments)
    {
        /* Yes */
        notDone = FALSE;
    }
}

```

Continued on Figure 16d

FIG. 16c

Continued from Figure 16c

1608

```

        pause (time5);
    }

    /* Perform any cleanup activities */
    Finalize ();
}
ProcessSegment (segmentNo)
{
    /* Process each message in segment segmentNo */
    for (messageNo = 1; messageNo <= messagePool.segment (segmentNo).noOfMessages; messageNo++)
    {
        /* Send message messageNo in segment segmentNo */
        send message messagePool.segment [segmentNo].message [messageNo].message;

        /* Optionally wait for and optionally process message send ACK */

        messagePool.segment [segmentNo].message [messageNo].status = SENT;
        messagePool.segment [segmentNo].noOfMessagesSent++;

        /* Wait (after message is sent) */
        pause (time3);
    }

    messagePool.segment [segmentNo].status = WORKED;
    /* Wait (after segment is done */
    pause (time4);

    /* Exit */
}

```

FIG. 16d