IPC
Message Queues

- Interprocess communication
  - pipes
  - signals
    - very restrictive
    - need better IPC mechanisms
  - UNIX IPC
    - semaphores
    - shared memory

- Interprocess Communication (IPC)
  - `message queues`
    - "selective write"
    - "selective read"
    - FIFO queue
    - data structure maintained by kernel
  - `semaphores`
    - synchronization
    - "communicate small amounts of data"
  - `shared memory`
    - communication via a shared data space
    - random access

-related/unrelated processes *on the same system*

- IPC resources
  - creator
  - owner
  - access permissions
  - established upon creation
  - can be modified via system calls
  - information can be obtained via `ipcs` command

ipcs utility

One shared memory segment attached (shared) by three processes

```
linux# ipcs
------ Shared Memory Segments ------
    key        shmid      owner     perms    bytes   nattch  status
0x00000000   25198594     root       666    247264     3
```

Four sets of semaphores all owned by root

```
------ Semaphore Arrays ------
    key        semid      owner     perms    nsems     status
0x00000000     65537      root       666       4
0x00000000     98306      root       666      16
0x00000000    131075      root       666      16
0x00000000    163844      root       666      16
```

No message queues are currently allocated

```
------ Message Queues------
    key        msqid      owner     perms    used-bytes messages
```

- ipcs utility

```
ipcs -s 622592

Semaphore Array semid=622592
uid=33   gid=33  cuid=0  cgid=0
mode=0600, access_perms=0600
nsems = 1
otime = Mon Sep 26 10:49:00 2005
ctime = Sun Sep 25 20:12:11 2005
semnum     value      ncount     zcount     pid
0          0          9          0          24077
```

-a	all
-m	shared memory
-q	message queues
-s	semaphores

-c	creator
-l	limits
-p	process id
-t	time
-u	summary

- limits
  - set when kernel is generated

```
# ipcs -l
------ Shared Memory Limits --------
max number of segments = 4096
max seg size (kbytes) = 32768
max total shared memory (pages) = 2097152
min seg size (bytes) = 1

------ Semaphore Limits --------
max number of arrays = 128
max semaphores per array = 250
max semaphores system wide = 32000
max ops per semop call = 32
semaphore max value = 32767

------ Messages: Limits --------
max queues system wide = 16
max size of message (bytes) = 8192
default max size of queue (bytes) = 16384
```

- `ipcrm` utility
  - remove resource (owner)

```
-M shmkey
            Mark the shared memory segment associated with key shmkey for removal.
            This marked segment will be destroyed after the last detach.

-m shmid
            Mark the shared memory segment associated with id shmid for removal.
            This marked segment will be destroyed after the last detach.

-Q msgkey
            Remove the message queue associated with key msgkey from the system.

-q msqid
            Remove the message queue associated with the id msqid from the system.

-S semkey
            Remove the semaphore set associated with key semkey from the system.

-s semid
            Removes the semaphore set associated with id semid from the system.
```

- `ipcs` system calls

Functionality	Message queue	Semaphore	Shared memory
Allocate IPC resource; gain access to an existing IPC resource	`msg`**`get`**	`sem`**`get`**	`shm`**`get`**
Control an IPC resource: obtain/ modify status information, remove the resource	`msg`**`ctl`**	`sem`**`ctl`**	`shm`**`ctl`**
IPC operations: send/receive messages, perform semaphore operations, attach/free a shared memory segment	`msg`**`snd`** `msg`**`rcv`**	`sem`**`op`**	`shm`**`at`** `shm`**`dt`**

IPC


```
struct ipc_perm {
   __key_t   __key                    /* Key                    */
   __uid_t   uid;                     /* Owner's user ID.       */
   __gid_t   gid;                     /* Owner's group ID       */
   __uid_t   cuid;                    /* Creator's user ID      */
   __gid_t   cgid;                    /* Creator's group ID     */
   unsigned short int mode;           /*  Access permissions    */
   unsigned short int __pad1;
   unsigned short ins __seg;          /* Sequence number        */
   unsigned short int __ pad2;
   unsigned long int __ unused1;
   unsigned long int __ unused2;
   };
```

- get system calls
  - msgget
  - semget
  - shmget

- allocate new resource or gain access to existing resource
- permission
- returns ipc identifier
  - used to reference the resource
  - used by system as an index to IPC permission structure

```
struct ipc_perm {
  __key_t   __key      /* Key                 */
  __uid_t   uid;       /* Owner's user ID.    */
  __gid_t   gid;       /* Owner's group id    */
  . . .
```

- common arguments
  - key
    - used to generate an IPC identifier
    - one-to-one relation
    - **`ftok`** library function
      unrelated processes can generate same key

```
key_t ftok(char *pathname, char proj);
```

- common arguments
  - `IPC_PRIVATE`
    - create IPC resource with unique IPC identifier
    - no other process creating/accessing an IPC resource will have same IPC identifier
    - in related processes
      - parent creates IPC resource
      - child inherits
      - exec pass as argument or environment variable
    - non-related processes
      - server creates
      - send key to clients

- Common arguments (cont.)
  - message flag
    - Access permissions
    - Lower nine bits of flag used for permissions

Table 6.4 Required Permissions for IPC System Calls.

Permissions Required	Message Queues	Semaphores	Shared Memory
write (alter)	`msgsnd` place message in the queue	`semop` increase or decrease a semaphore value	`shmat` to write to the shared memory segment
	`msgctl` write out modified IPC status information	`semctl` set the value of one semaphore or a whole set; write out modified IPC status information	`shmctl` write out modified IPC status information
read	`msgrcv` obtain message from queue	`semop` block until a semaphore becomes 0	`shmat` read from the shared memory segment
	`msgctl` to retrieve IPC status information	`semctl` to retrieve IPC status information	`shmctl` to retrieve IPC status information

- flags can be ORed
  - `IPC_CREAT`
    - create if it does not exists
    - if present and not created with `IPC_PRIVATE`

      returns IPC identifier
  - `IPC_CREAT | IPC_EXCL`
    - create exclusive
    - fails if resource exists

- `ctl` system calls
  - `msgctl`
  - `semctl`
  - `shmctl`

- act upon permissions
  - `IPC_STAT`
    - status information
  - `IPC_SET`
    - set owner/group/mode
  - `IPC_RMID`
    - destroy and remove

`IPC_SET` and `IPC_RMID` only by owner, creator or superuser

- operations system calls
  - `msgsnd/msgrcv`
    - blocks on write-full read-empty
      - until `signal` received, can read/write or resource removed
    - overwrite with `IPC_NOWAIT`
  - `semop`
    - semaphore operations
    - sets and test semaphore values
    - blocks when attempt to decrement a semaphore currently at 0 or waiting for a semaphore to become 0
  - `shmat/shmdt`
    - map/attach shared memory blocks
    - unmap/detach shared memory blocks
    - non-blocking

- message queues
  - Creation

```
int msgget (key_t key, int msgflg);
```

returns:

nonnegative integer queue identifier associated with key used to reference the message queue

`key`:

-specified directly

-using `ftok`

–used to produce unique identifier

- `msgflg`
  - low order bits
    - access permissions
  - additional flags ORed
    - IPC_CREAT
    - IPC_EXCL
  - new message queue is created
    - IPC_PRIVATE used as key
    - use IPC_CREAT flag and no queue exists
      - returns `id` if queue exists and IPC_EXCL was not specified

- Example

```
/*  Message queue generation */
#define _GNU_SOURCE
#include <cstdio>
#include <unistd.h>
#include <linux/limits.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
using namespace std;
const int MAX=5;
int
main( ){
  FILE *fin;
  char  buffer[PIPE_BUF], proj = 'A';
  int   i, n, mid[MAX];
  key_t key;
  for (i = 0; i < MAX; ++i, ++proj) {
    key = ftok(".", proj);
    if ((mid[i] = msgget(key, IPC_CREAT | 0660)) == -1) {
      perror("Queue create");
      return 1;
    }
  }
  fin = popen("ipcs", "r");
  while ((n = read(fileno(fin), buffer, PIPE_BUF)) > 0)
    write(fileno(stdout), buffer, n);
  pclose(fin);
  for (i = 0; i < MAX; ++i )
    msgctl(mid[i], IPC_RMID, (struct msqid_ds *) 0);
  return 0;
}
```

IPC


```
gcos:test dcanas$ ./msg
IPC status from <running system> as of Tue Sep 25 13:29:53 EDT 2013
T     ID     KEY        MODE       OWNER    GROUP
Message Queues:
q 262144 0x410297f6 --rw-rw----   dcanas    staff
q 131073 0x420297f6 --rw-rw----   dcanas    staff
q 131074 0x430297f6 --rw-rw----   dcanas    staff
q 131075 0x440297f6 --rw-rw----   dcanas    staff
q 131076 0x450297f6 --rw-rw----   dcanas    staff

T     ID     KEY        MODE       OWNER    GROUP
Shared Memory:
m 393216 0x53414e44 --rw-rw-rw-   dcanas    staff

T     ID     KEY        MODE       OWNER    GROUP
Semaphores:
```

```
gcos:test dcanas$ ipcs
IPC status from <running system> as of Tue Sep 25 13:32:28 EDT 2013
T     ID     KEY        MODE       OWNER    GROUP
Message Queues:

T     ID     KEY        MODE       OWNER    GROUP
Shared Memory:
m 393216 0x53414e44 --rw-rw-rw-   dcanas    staff

T     ID     KEY        MODE       OWNER    GROUP
Semaphores:
```

- Creation
  - system message-queue data structure
    - `msqid_ds`
  - maintained by system

```
struct msqid_ds {
  struct ipc_perm msg_perm;
  struct msg *msg_first;          /* first message on queue, unused  */
  struct msg *msg_last;           /* last message in queue, unused */
  __kernel_time_t msg_stime;      /* last msgsnd time */
  __kernel_time_t msg_rtime;      /* last msgrcv time */
  __kernel_time_t msg_ctime;      /* last change time */
  unsigned long  msg_lcbytes;     /* Reuse junk fields for 32 bit */
  unsigned long  msg_lqbytes;     /* ditto */
  unsigned short msg_cbytes;      /* current # of bytes on queue */
  unsigned short msg_qnum;        /* number of messages in queue */
  unsigned short msg_qbytes;      /* max number of bytes on queue */
  __kernel_ipc_pid_t msg_lspid;   /* pid of last msgsnd */
  __kernel_ipc_pid_t msg_lrpid;   /* last receive pid */
};
```

- creation
  - set user/group id
  - **`msg_first/last`** point to first/last message in queue
  - linked list of messages

```
struct msg {
    struct msg      *msg_next;      /* ptr to next message on q */
    long            msg_type;       /* message type */
    ushort          msg_ts;         /* message text size */
    short           msg_spot;       /* address of text message */
};
```

- Message Queue structure

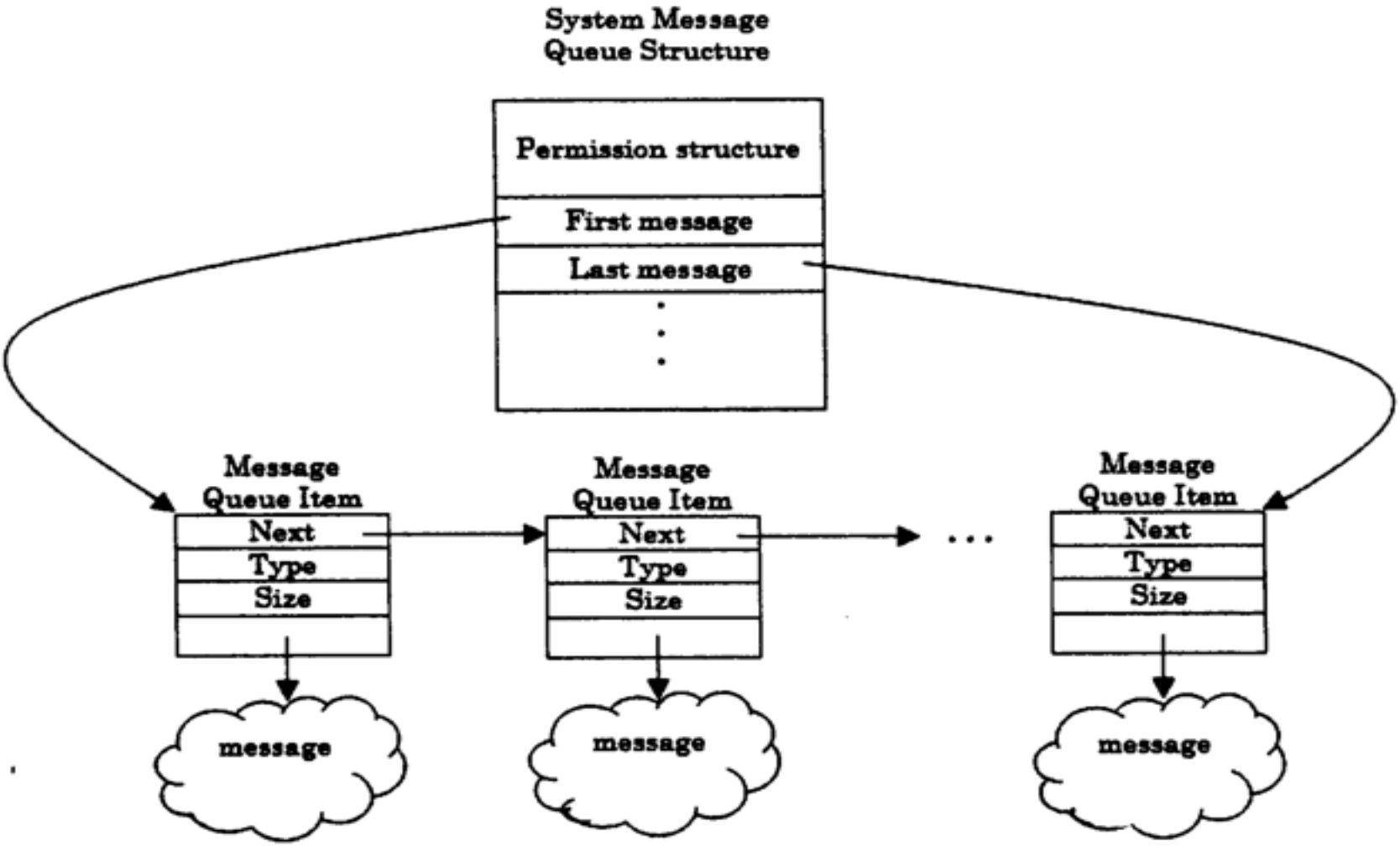

Figure 6.5 A message queue with N items.

- Message Queue Control

```
int msgctl( int msqid, int cmd, struct msqid_s  *buf);
```

- IPC_STAT
  - returns current values of msqid_ds in buf
- IPC_SET
  - modify values
- IPC_RMID
  - removes message queue

IPC


```
/* Display message queue status information */
#include <iostream>
#include <cstdio>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
using namespace std;
int main () {
        int     mid;
        key_t   key;
        struct msqid_ds buf;
 key = ftok(".",'z');
 if ((mid = msgget(key, IPC_CREAT | 0660 )) == -1) {
   perror("Queue create:");
  return 1;
}
 msgctl(mid, IPC_STAT,&buf);
 cout << "Message Queue *Permissions*  Structure information" << endl;
 cout << "Owner's user ID     \t" << buf.msg_perm.uid  <<  endl;
 cout << "Owner's group ID    \t" << buf.msg_perm.gid  <<  endl;
 cout << "Creator's user ID   \t" << buf.msg_perm.cuid  <<  endl;
 cout << "Creator's group ID  \t" << buf.msg_perm.cgid  <<  endl;
 cout << "Access mode in HEX\t" << hex << buf.msg_perm.mode  <<  endl;
 cout << "\nAdditional Selected Message Queue Structure Information\n";
 cout << "Current # of bytes on queue     \t" << dec << buf.msg_cbytes  <<  endl;
 cout << "Current # of messages on queue  \t" << dec << buf.msg_qnum    <<  endl;
 cout << "Maximum # of bytes on queue     \t" << dec << buf.msg_qbytes  <<  endl;
 msgctl(mid, IPC_RMID, (struct msqid_ds *) 0 );
return 0;
}
```

```
Message Queue *Permissions*  Structure information
Owner's user ID          501
Owner's group ID         501
Creator's user ID        501
Creator's group ID       501
Access mode in HEX       1b0
```

1b0 => 0001 1011 0000 => 000 **110 110 000**
Permission (lower nine bits) => 660

```
Additional Selected Message Queue Structure Information
Current # of bytes on queue              0
Current # of messages on queue           0
Maximum # of bytes on queue              16384
```

Message Queue Operations

- send and receive messages

```
struct msgbuf {
    long int mtype;        /* type of message   */
    char mtext[1];         /* text of message   */
};
```

- `mtype`
  - type of message
    - long integer
    - used by **`msgrcv`** to selectively retrieve messages
- `mtext`
  - body of the message
    - any valid structure
    - long integer
    - followed by 0 or more bytes

- sending messages

```
int msgsnd( int msqid, struct msgbuf *msgp,
            size_t msgsz, int msgflg );
```

- `msgp`
  - pointer to message
- `msgsz`
  - size of message
    - size of message structure - **`size`** of message type
- `msgflg`
  - actions if limits reached
    - **`IPC_NOWAIT`**
      - return and message not sent
    - 0
      - block
      - not beyond limits, receive a signal, message queue removed

receiving messages

```
int msgrcv( int msqid, struct msgbuf *msgp,
            size_t msgsz, long msgtyp, int msgflg );
```

- `msgp`
  - pointer to receiving buffer
    - long integer
    - text
- `msgsz`
  - maximum size of message
- `msgflg`
  - actions (if message not in queue or message too large)
    - **`IPC_NOWAIT`**
      - message type requested not in queue
    - **`MSG_NOERROR`**
      - silently truncate
    - **`MSG_EXCEPT`**

- `msgtyp`
  - type of message

`msgtyp`	Action
0	retrieves the first message of **any** `msgtyp`
> 0	retrieves the first message **equal** to `msgtyp` if `MSG_EXCEPT` is not specified. If `MSG_EXCEPT` is specified, the first message that is **not equal** to `msgtyp`
< 0	retrieve the first message of the lowest type less than or equal to absolute value of `msgtyp`

- Example

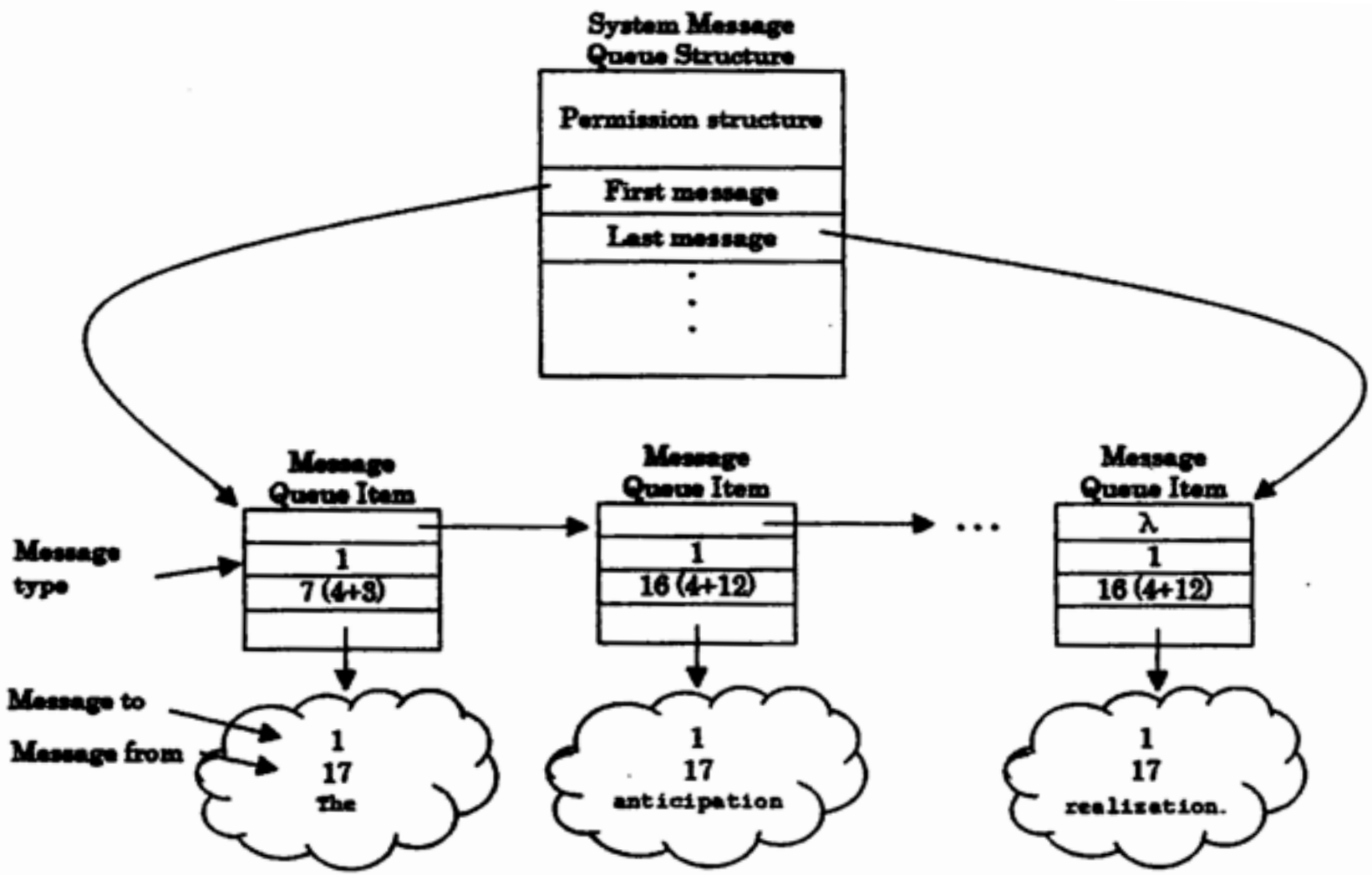

Figure 6.7 Conceptual view of message queue after the client has sent all seven messages

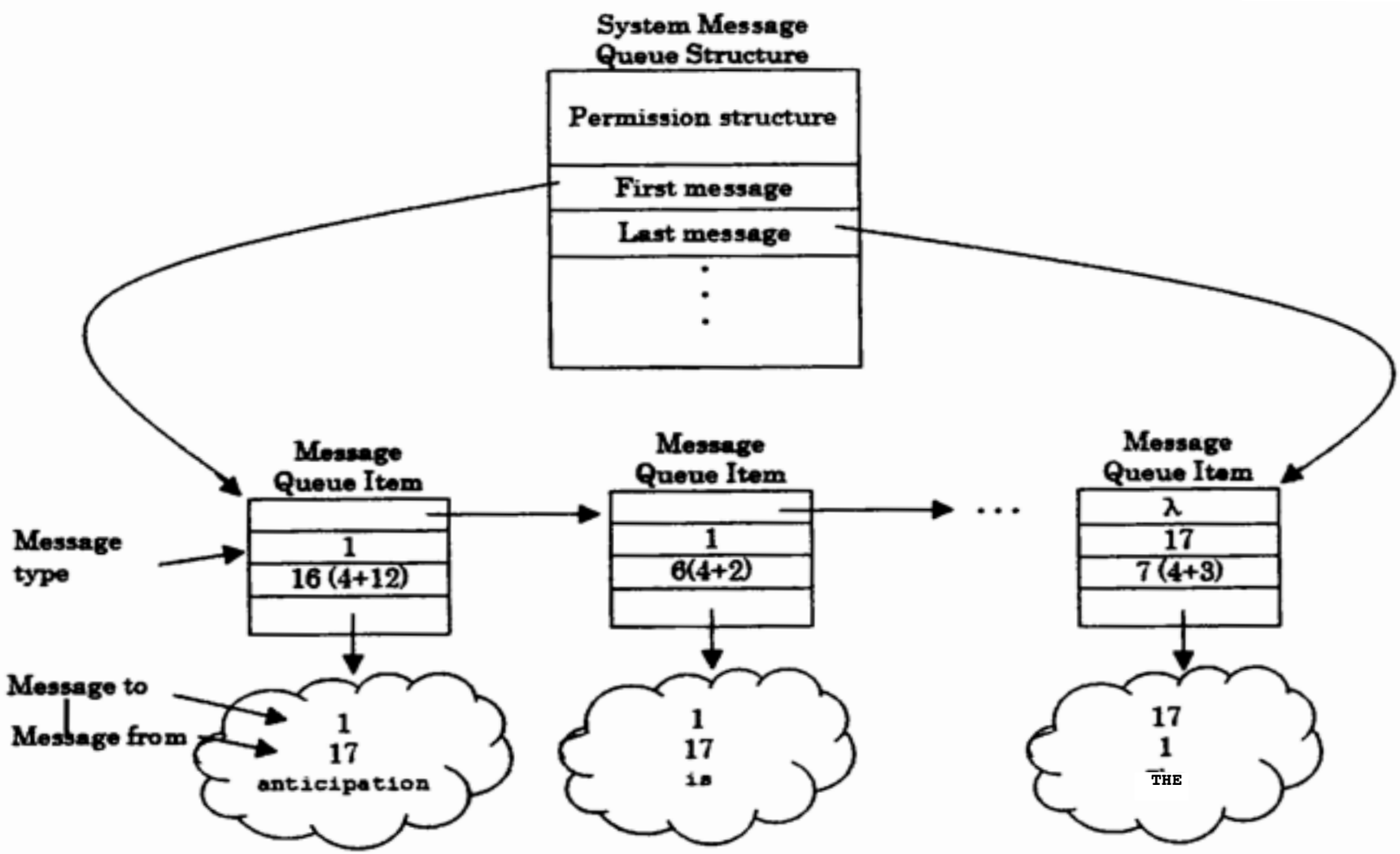

Figure. 6.8 Conceptual view of message queue after the first client message has been processed.