Inter Process Communication - 2

Contents

- System V IPC
  - Message Queues
  - Shared Memory
  - Semaphores

Common properties

- Each mechanism contains a table whose entries describe all instances of the mechanism.
- Each entry contains a numeric key, which is its user-chosen name.
- Each mechanism contains a “get” system call to create a new entry or to retrieve an existing one.
- Each IPC entry has a permissions structure that includes the user ID and group ID of the process that created the entry.
- Each mechanism contains a “control” system call to query status of an entry, to set status information, or to remove the entry from the system.

Message Queues

- A Message queue is a header pointing at a link list of messages.
- Each message contains a type field(4byte), followed by a data area.

Message Queues

Queue structure contains the following fields

- Owners's uid and gid
- Creator's uid and gid
- Permissions

Message Queues

Queue structure contains the following fields

- Owners's uid and gid
- Creator's uid and gid
- Permissions

- Pointers to first and last messages on the linked list
- Number of messages and total number of data bytes on the linked list

Message Queues

Queue structure contains the following fields

- Owners's uid and gid
- Creator's uid and gid
- Permissions

- Pointers to first and last messages on the linked list
- Number of messages and total number of data bytes on the linked list
- Max number of data bytes that can be on the linked list
- Pid of last processes to send and receive messages
- Timestamps – last msgsnd time, last msgrcv time, last change time

Message Queues

- The system call to get a message queue identifier is

  int msgget(key, flag)

- It creates a new entry / retrieve a existing entry.
- Keys are numbers used to identify an IPC object on a UNIX system.
- Flag specifies the access permissions.
- Msgget returns a descriptor which is used as an index into an array of message queue headers.

Message Queues

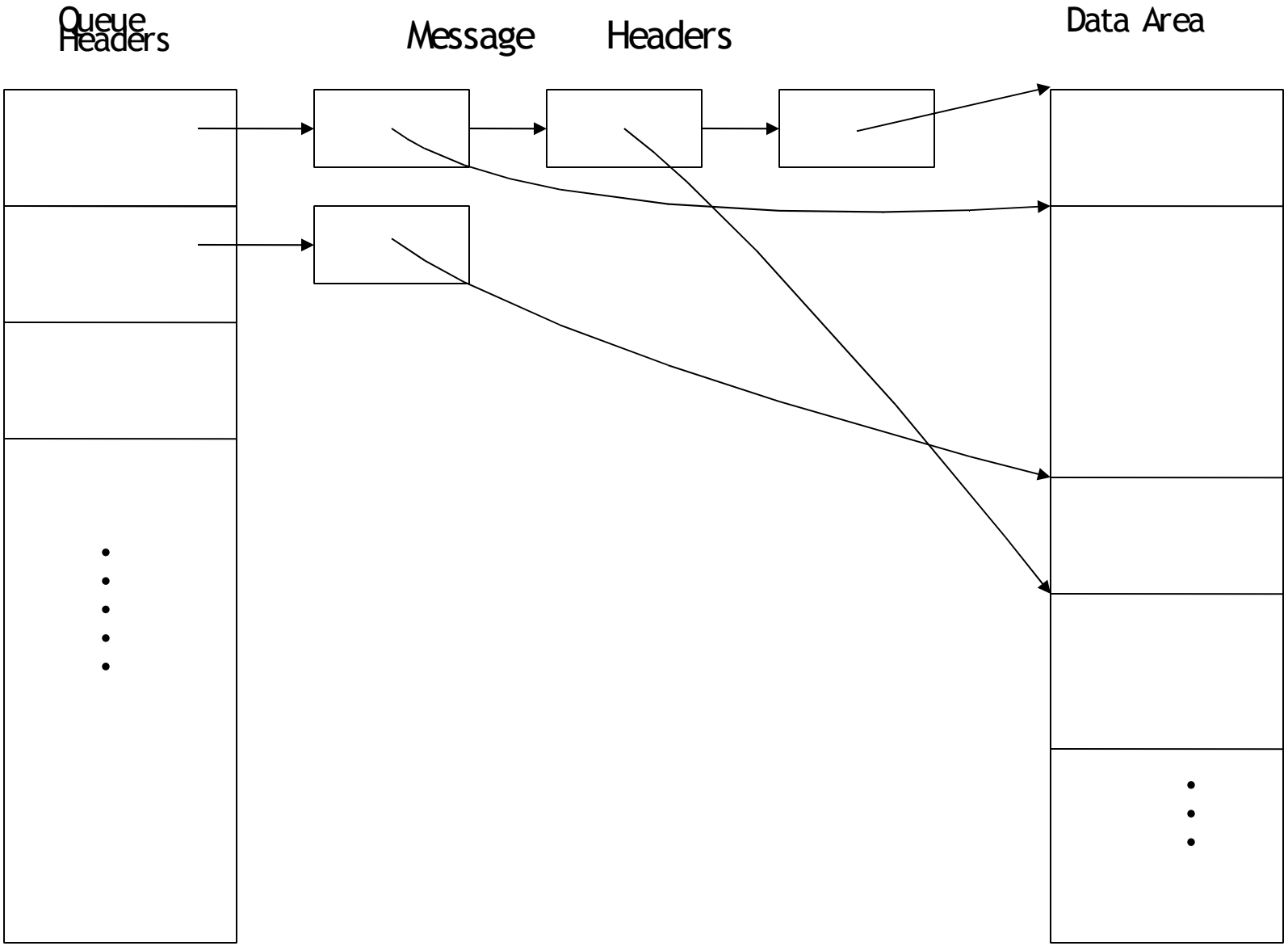
- New message queue is created

  - if key has value IPC_PRIVATE

  - If new key value is specified and IPC_CREAT flag is asserted in flags

- If IPC_EXCL and IPC_CREAT are both set and if message queue with the key value exists, then *msgget* fails

Message Queues



Message Send

- The system call to send a message is

  int msgsnd(msgqid, msg, count, flag)

- *msgqid* - descriptor of message queue
- *msg* – pointer to the message structure

```
struct msgbuf
{
        long mtype;
        char mtext[1];
}
```

- *count* - size of the data array
- *flag* – action kernel should take if it runs out of internal buffer space

Message Send

```
Algorithm msgsnd                    /*send a message*/
input : (1) message queue descriptor     (2) address of message structure
        (3) size of message              (4) f lags
output : number of bytes sent
{
    check legality of descriptor, permissions;
    while(not enough space to store message)
    {
        if(flags specify no to wait)
            return;
        sleep(until event enough space is available);
    }
    get message header;
    read message text  from user space to kernel;
    adjust  data structures : enqueue message header, message header points to
    data, counts, timestamps, process ID;
    wakeup all processes waiting to read message from queue;
}
```

Message Receive

- The system call to receive a message is

  int msgrcv(msgqid, msg, count, type, flag)

- *msgqid* - descriptor of message queue
- *msg* – address of a user structure to contain received message
- *count* - size of the data array
- *type* - message type user wants to read
- *flag* – action kernel should do if no messages are on the queue

Message Receive

```
Algorithm msgrcv                    /*receive message*/
input : (1) message queue descriptor     (2) address of data array for incoming
    message
        (3) size of data array                  (4) requested message type   (5) f lags
output : number of bytes in returned message
{
    check permissions;
  loop:
    check legality of message descriptor;
    /* find messageto return to user */
    if(requested message type == 0)
          consider first message on queue;
    else if( requested message type > 0)
          consider first message on queue with given type;
    else              /* requested message type < 0 */
          consider first message of the lowest typed message on queue, such that its
          type  is <= absolute value of requested type;
```

Message Receive

```
    If ( there is a message)
    {
        adjust message size or return error if user size too small;
        copy message type, text from kernel space to user space;
        unlink message from queue;
        return;
    }
    /* no message */
    if ( flags specify not to sleep )
        return with error;
    sleep ( event message arrives on queue);
    goto loop;
}
```

Message Control Operations

- A process uses the msgctl system call to query status and set parameters for the message queue

  int msgctl(msgqd, cmd, buf)

- msgqid - identifies the message queue table entry
- Cmd - specifies the type of operation
- Buf - address of user buffer

Message Control Operations

Cmds available are

- IPC_STAT
  - Fetch the *msqid_ds* structure.

Message Control Operations

- IPC_SET
  - Set owner's user id, owners's group id, access modes, maximum number of bytes in the queue(only by super user).
  - Permissions required by the process for the command
    - → superuser privilege
    - → process's effective uid equals creator's user id / owner's user id

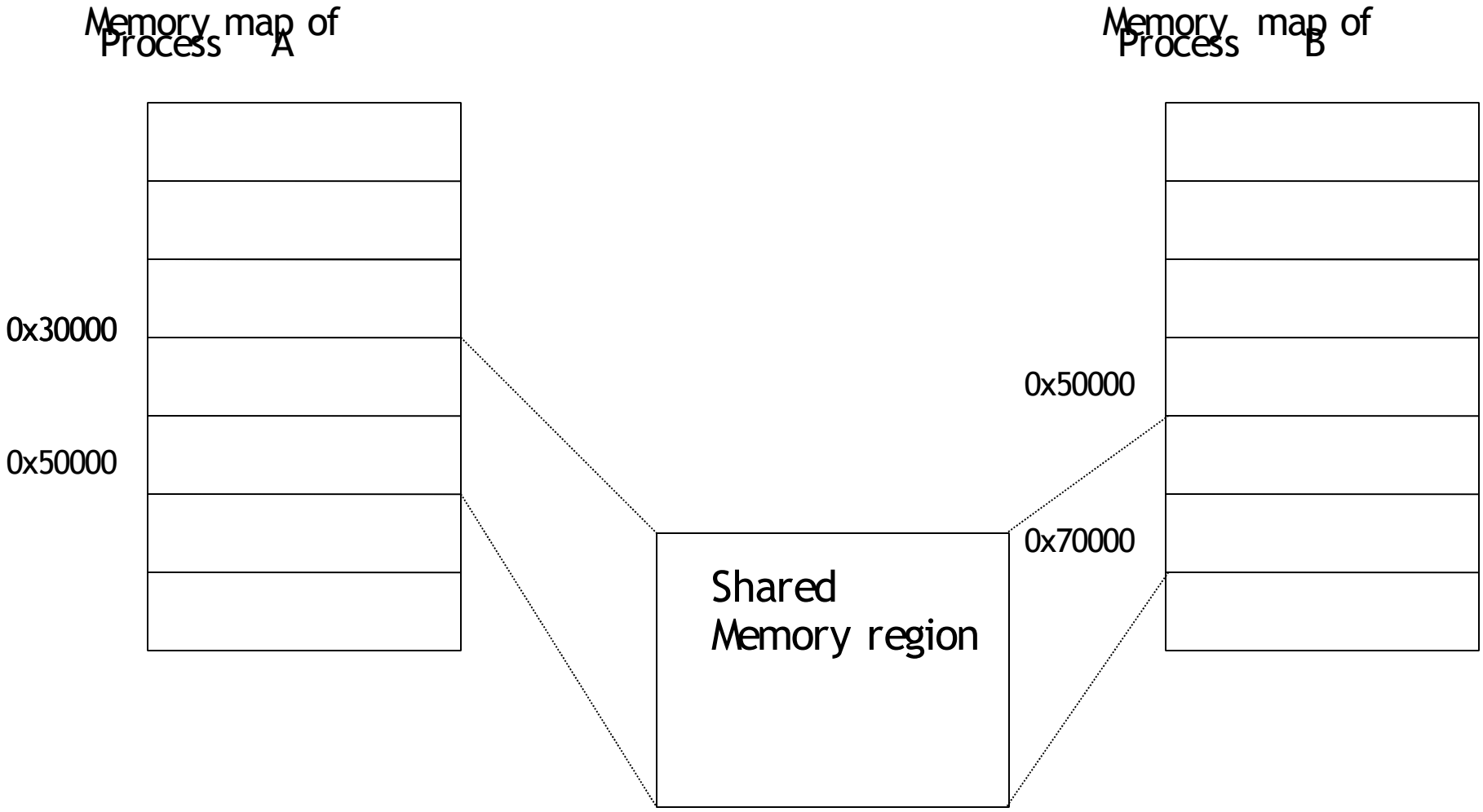
Message Control Operations

- IPC_RMID
  - Remove the message queue from the system.
  - Removal is immediate.
  - Permissions required by the process for the command
    - → superuser privilege
    - → process's effective uid equals creator's user id / owner's user id

Shared Memory

- Shared memory is a portion of memory that is shared by multiple processes
- Processes may attached this region to their virtual address space
- Shared memory provides the fastest mechanism for processes to share data
- If a process writes to a shared memory location, the new contents of that location are immediately visible to all processes sharing the region

Shared Memory



Shared Memory

Shared memory table contains the following fields

- Owners's uid and gid
- Creator's uid and gid
- Permissions

Shared Memory

Shared memory table contains the following fields

- Owners's uid and gid
- Creator's uid and gid
- Permissions

- Size of the segment in bytes
- Pid of the creator and last operator
- Number of current attaches (nattach)
- Marked for deletion
- Timestamps – last attach time, last detach time, last change time

Shared Memory

- The system call to create a new region of shared memory or get an existing one is

  int shmget(key, size, flag)

- It creates a new entry / retrieve a existing entry.
- Keys are numbers used to identify an IPC object on a UNIX system.
- Size is the number of bytes in the region
- Flag specifies the access permissions.
- shmget returns a segment descriptor

Shared Memory

Shared memory availability after the following system calls

- Fork() - After fork() child inherits the attached shared memory segments

- Exec() - After exec() all attached shared memory segments are detached (not destroyed)

- Exit() - On exit() all attached shared memory segments are detached (not destroyed)

Shared Memory Attach

- A Process attaches shared memory region to its virtual address space with the system call

  virtaddr = shmat (shmid, addr, flags)

- *Shmid* - descriptor returned by shmget
- *Addr* - virtual address where the user wants to attach the shared memory
- *Flag* - specifies whether the region is read-only and whether the kernel should round off the user specified address

Shared Memory Attach

```
Algorithm shmat              /* attach shared memory*/
input : (1) shared memory descriptor
        (2) virtual address to attach shared memory      (3)  f lags
output : virtual address where shared memory was attached
{
    check validity of descriptor, permissions;
    if ( user specified virtual address )
    {
        round off virtual address, as specified by flags;
        check legality of virtual address, size of region;
    }
    else       /* user wants kernel to find good address */
    {
        kernel picks virtual address : error if none available;
        attach region to process address space;
        if ( region being attached for the first time )
            allocate page tables, memory for region;
        return ( virtual address attached );
}
```

Shared Memory Detach

- A Process detaches shared memory region from its virtual address space with the system call

  int shmdt (addr)

- *Addr* - virtual address returned by shmat system call

Shared Memory Control

- A process uses the shmctl system call to query status and set parameters for the shared memory region

  int shmctl(shmid, cmd, buf)

- Shmid - identifies the shared memory table entry
- Cmd - specifies the type of operation
- Buf - address of user buffer

Shared Memory Control

Cmds available are

- IPC_STAT
  - Fetch the *shmid_ds* structure.

Shared Memory Control

Cmds available are

- IPC_SET
  - Set owner's user id, owners's group is, access modes.
  - Permissions required by the process for the command
    - superuser privilege
    - process's effective uid equals creator's user id / owner's user id

Shared Memory Control

Cmds available are

- IPC_RMID
  - Remove the shared memory segment set from the system if number of current attaches becomes 0.
  - Permissions required by the process for the command
    - superuser privilege
    - process's effective uid equals creator's user id / owner's user id

Thank you