Inter process Communication

Description of 3 types of IPC's and the structures used

1. Shared Memory

Constants for shared memory

```
SHMMNI   /* Max.  number of shared memories open at a time*/
SHMMIN   /* Min.  size of shared memory */
SHMMAX  /* Max. size of shared memory */
SHMSEG   /* Max. size of shared Memory Segments per process */
```

All these are configurable by the user using the control panel applet.

Data structures for shared memory

Data structures for shared memory

```
struct Pshm
{
    long          refcount;    /* Number of processes referencing this structure */
    HANDLE      handle;   /* Handle to memory mapped file */
    HANDLE      event;     /* Synchronization event. There is basically a numerical
                                    value. Whenever there is a change in the data this
                                     will      be set   */
    DWORD       ntpid;      /* Process which created the event */
    HANDLE      mutex;   /* to make all ipc operations atomic */
    DWORD       ntmpid;  /* Process which created the mutex */
    ...
};
```

The ntpid and ntmpid is used here because ipc's are used across processes and the handle value doesn't make sense across process. Handle value is valid only in the process region that created it. So this handle needs to be duplicated before using. To duplicate the handle we need to specify the handle and the process that created it.

2. Semaphores

Constants used for semaphore

```
SEMMNI  /* max semaphores that can be open */
SEMMSL  /* max number of semaphores per id (process) */
SEMMNS  /* max number of semaphores in system */
SEMOPM  /* max number of operations per semop call */
SEMVMX  /* semaphore maximum value */
```

These values can again be configurable using control panel applet.

Data structures for semaphores

```
struct Psem
{
    long          refcount;
    HANDLE      handle;
    HANDLE      event;
    DWORD       ntpid;
    HANDLE      mutex;
    DWORD       ntmpid;
    ...
};
```

These fields defined within the structure are specific to UWIN. The rest (...) are standard and provided by UNIX.

3. Message Queues

Constants for message queues

```
MSGMNI   /* Max. number of messages (msgid) */
MSGMAX  /* Max. size of message (bytes) */
MSGMNB  /* Max. size of a message queue */
```

Data structures for message queues

```
struct Pmsg
{
    long            refcount;
    HANDLE      handle;
    HANDLE      event;
    DWORD        ntpid;
    HANDLE      mutex;
    DWORD       ntmpid;
    long            first; /* offset to the first message in the queue */
    long            last; /* offset to the last message in the queue */
    long            next; /* offset to the location where a new msg can be stored */
    ...
};
```

Note
Any changes to the data made in the memory mapped file is signaled through the event .The memory mapped file is defined in /temp/.ipc. Mutex is used to make the operations atomic.

Message structure

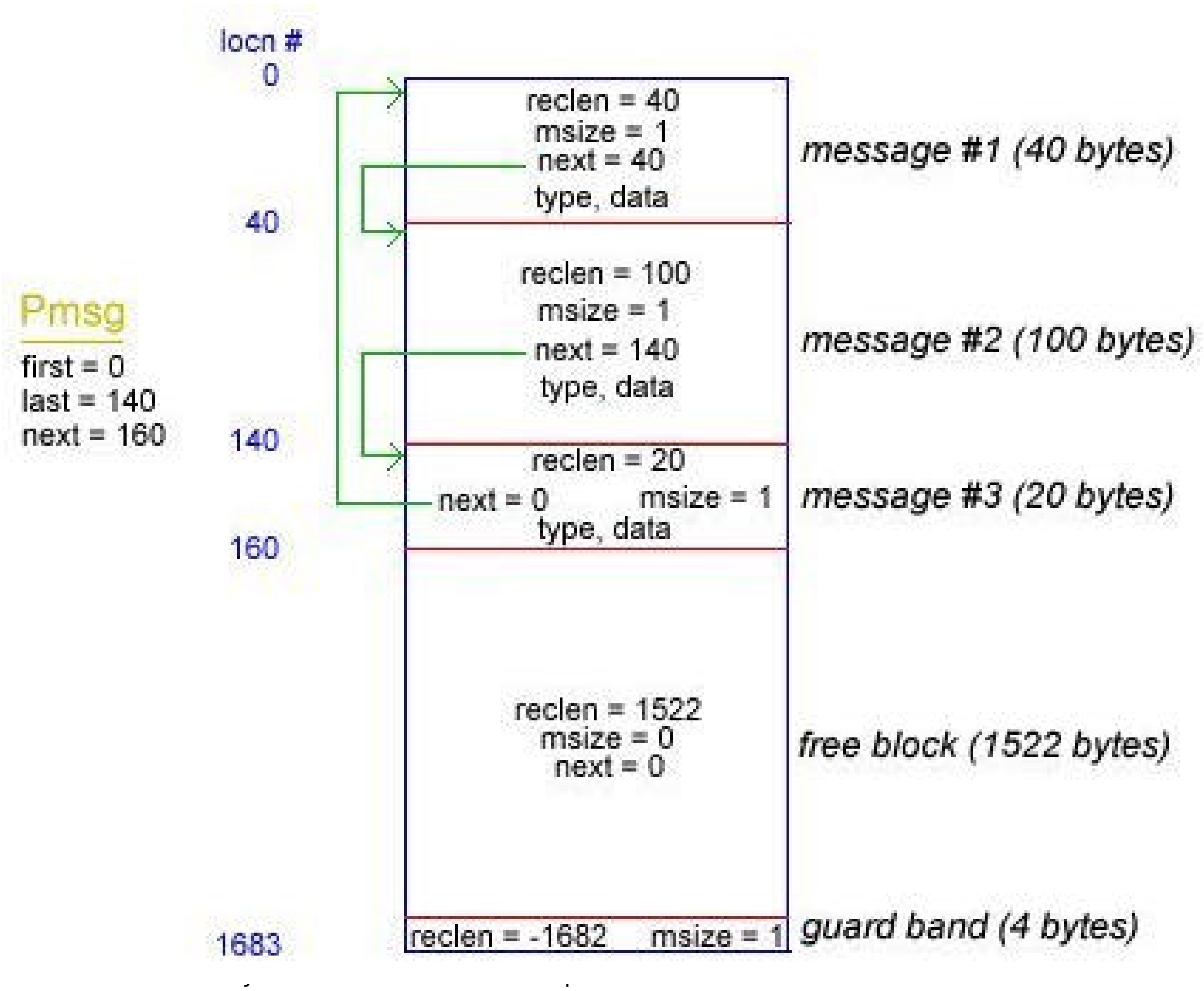
This message queue is by default 1MB.The message structure is defined as follows.

```
struct Message
{
    long    reclen;   /* number of  bytes current block can store */
    long    msize;   /* block is free (0) or allocated (1) */
    long    next; /* offset to the next msg in the queue */
    long    type; /* type of message as supplied by the application */
    char    msg[4]; /* data as supplied by the application */
};
```

When more than one consecutive block are free, they are merged into a single free block.
There can be more than one free block at any given time which are not contiguous.

The *Message* structure resides in the memory mapped file.

Layout of memory mapped file



The guard band is used to make the search circular starting from the offset present in the *next*.

Deletion in the message queue

If there are no messages in the queue, it returns error or waits till a message arrives (depending on the IPC_NOWAIT flag).

If there are messages, each of them is compared with the type to see if that is being requested by the application.

If a match is found, the message is sent to the application and the memory block is freed up.

Note
One message queue is created per id.

Functions / Macros used

msgnext() : Finds the location of the next available message in the message queue (utilizes the guard band to loop back to the beginning of the memory).