Global Memory

Brief Description:

This Global memory used by UWIN is the shared memory where the process information, file descriptors etc are stored.

When is it created?

Basically this memory is mapped to every process that is running under UWIN. It is created when the first UWIN process starts. This process internally calls init.exe. This init.exe creates the global memory (using the init_sharemem() function) and gets a handle to it. Other processes use this handle. Init.exe has to be present before any process comes up. Init basically does 2 things.

1) Does all the clean up activities.
2) Passes the open socket handles between processes.

When subsequent processes are created a check is performed if shared memory is initialized and then the handle is returned to the process. Init basically contains the handle, which it gets when the first UWIN process is created. The processes created are mapped on to the shared memory. The Map File Name that provides a mapping between the process and the shared memory does this. There is another API Create File Mapping that actually creates the memory.

When is it Destroyed?

This Global memory is finally destroyed when the last UWIN process is terminated. This is generally the service associated with UWIN.

How is it stored?

It is stored using named windows memory mapped files. This name is declared in xinit.c. The named of the shared memory is stored in the variable SHMEM_NAME and is initialized to UWIN.vi. Whenever there is a requirement for using another shared memory simultaneously, another one can be created with a different name. This of course makes the shared memory mutually exclusive.

What is stored?

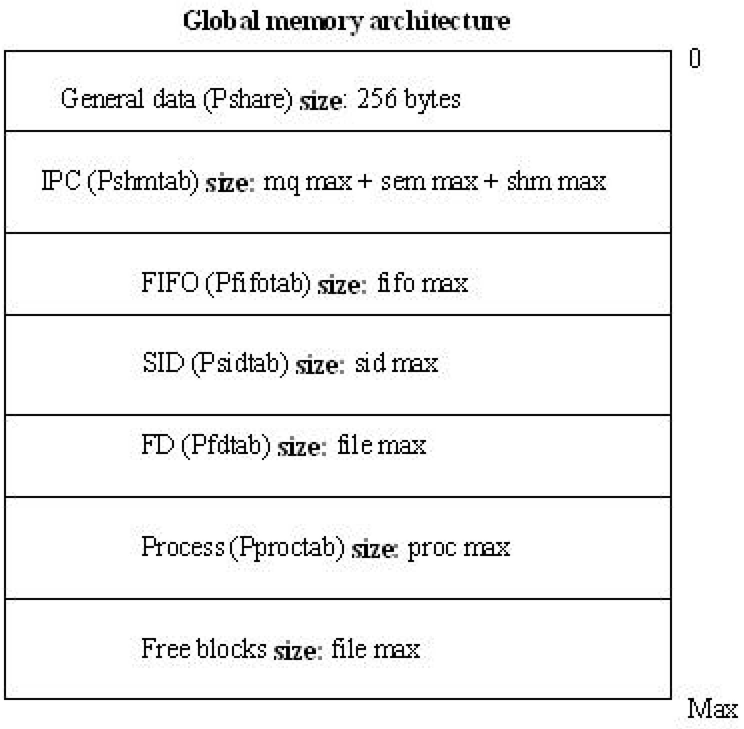
Every process has the read and write access to this shared memory. The shared memory contains the following information to be accessed by all processes running under UWIN.

* General -- This contains a structure called **Pshare**. This contains all the generic information like the device information, version of windows etc.
* IPC -- This contains a structure **Pshmtab**. This contains information about memory queues, inter-process communication, semaphores etc.
* FIFO -- This contains a structure called **Pfifotab**. This contains information about first in first out details of the queue.
* Security Identifiers – This contains a structure called Psidtab. This contains all the security identifier information.
* File Descriptor – This contains a structure called **Pfdtab**. This contains information about the file descriptors like fd number, fd type etc. If any process requires a file descriptor it gets it from here.
* Process (Pproctab) – This contains a structure called Pproctab. This contains all the process related information like pid, ppid, created files, deleted files, process name, threads associated with it etc. This is implemented using a hash table. This implementation speeds up the time required to obtain the data related to the processes.
* Free blocks – This is required for allocation and reallocation of blocks. Basically when a device is opened it obtains a free block and all the device related information is written into this block by the processes.

The size of the shared memory is determined by the 7 components mentioned above. But this is tunable by the user. Each of the components have a default size and this can be changed suitably using the applet provided or directly changing the registry contents.

The size of the global memory is the sum of sizes of the individual components. The size of the free block pool is determined by number of files the system can have. Basically the function init_sharemem() mentioned earlier calculates the size of the global memory and passes the value to the Create File Mapping that actually creates the memory.

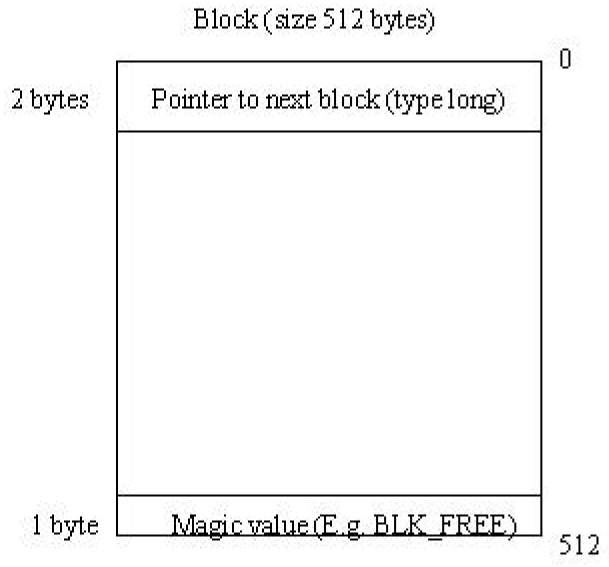
Architecture of the shared memory


Global memory architecture
0
General data (Pshare) size: 256 bytes
IPC (Pshmtab) size: mq max + sem max + shm max
FIFO (Pfifotab) size: fifo max
SID (Psidtab) size: sid max
FD (Pfdtab) size: file max
Process (Pproctab) size: proc max
Free blocks size: file max
Max

Maintenance of free blocks

Free Block structure

The free block pool is basically divided into a set of 512 bytes block. The 512-byte block structure is indicated in the figure below. The value 512 byte is used here with the assumption that all structures are of size 512 bytes. This is defined by a macro called BLOCK_SIZE. If a block is free the first 2 bytes in the figure points to the next block that is free. If it is not free it will have a flag saying that it is not free. The last byte or the magic byte represents what type of information is present in the block.(for Ex. Device information). This is used for debugging purpose. The structure Share contains a field called top that contains a pointer to the first free block in the free block pool. So as and when a block is allocated its value gets changed.


Block (size 512 bytes)
0
2 bytes
Pointer to next block (type long)
1 byte
Magic value (E.g. BLK_FREE)
512

X -→ X → Null.

There are 2 functions to allocate and de-allocate the free blocks. They are block_alloc and block_free respectively. P and V signals are used in their implementation to prevent any problem regarding synchronization.

Functions and Macros of Interest

1. init_sharedmem()	:This creates the memory mapped file (shared memory) and maps this into the processes of UWIN.
2. block_alloc()	:This allocates a block from a free pool.
3. block_free()	:This adds the block into the free pool.
4. block_magic()	:Puts the passed value into the block's last byte address.
5. block_is_free()	:Checks if the block is free.
6. block_ptr()	:Returns the pointer address for the slot provided. The slot described here represents the offset value of the block from the starting block in the free pool. For ex a slot value of 5 represents an offset of 5*512 bytes from the starting position (block pool). Thus in actual Implementation no actual addresses are stored.
7. block_slot()	:Returns the slot number for the block address provided.