Hash table Implementation

Brief Description:

This is basically used to store and retrieve the information related to a process that comes up in UWIN. For ex. Parent child related information, group information etc.

What is the need?

1. Performance: To speed up the process of information retrieval.
2. Ease of use: Maintain the internal UWIN process table information effectively and perform creation/deletion easily.
3. Accessibility: Make the information accessible to all processes. This can be done because of the use of Hash values Unlike the absolute addresses.

How is it done?

All the information about a process that UWIN needs to maintain is stored in a process table data structure called **Pproc_t**. (For ex fd's associated with the process, parent process etc.). Hashing is done using process "pid". (*This pid is same as ntpid except in one case where the process does an exec. The basic problem is that NT doesn't provide an exec feature. So to implement it a new process has to be created which will obviously contain a pid different from the process that 'exec'ed, unlike Unix where both the pid's are the same. This inconsistency can be resolved by having separate fields for pid and ntpid. The values of which will determine if the process has 'exce'ed or not. Care is taken so that the handle of the parent process is not closed till the child process exited to avoid any inconsistency relating to the parent pid being assigned to some other process and the 'exec'ed process trying to access the parent process. The handle is stored in a field called phandle defined in the struct pproc_t. The wait thread is responsible for preventing the handle from closing).*
This process table data structure is maintained in the shared memory accessible to all UWIN processes and will be mapped to virtual memory of different process, since pointers across processes will not make sense.

What information is maintained?

Following linked lists are maintained to avoid linear scans.

- Processes that share a common hash code.

When 2 or more processes have the same hash code a linked list containing the processes of the same hash code is maintained. Thus when there is a clash one has to traverse through this list to get the process.

- Process that share a common parent.
  This makes the searching of the children easier, especially when waiting for large number of children to complete.

- Processes that share a common group.
  This makes the searching of processes of same group easier.

Process table entries

```
Typedef struct Pproc_t
{
...
Short parent;  /* Points to parent process */
Short child; /* Points to latest child */
Short group; /* Points to one of the processes in the group */
Short sibling; /* Points to the process that is at the same level as it */
Short next; /* Points to next process in the same hash slot */
...
} Pproc_t;
```

Taking care of clashes in the hash code

If an entry for process P2 has to be inserted into the slot where, say entry for P1 is already present, then the entry for P1 is stored in 'P2->next' and the pbq (pointer to the beginning of the queue) for that slot is filled with entry for P2. This basically is referred to as the slot value.

```
        next      next
    pbq  ->   P2   ->  P1
```

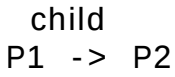
Similarly, if there is one more clash, the 'list' grows as below:

```
       Next      next       next
    pbq  ->  P3   ->   P2   ->  P1
```

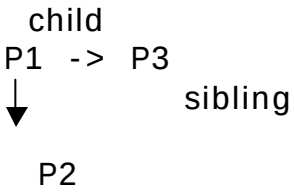
If 'next' is zero, it indicates end of 'list'. Also, if the value in the pbq for that particular slot is zero, it indicates that there are no processes in the slot.

Keeping track of Children

If a process P1 generates a child P2 (using a fork or exec), then 'P1->child‘ contains the hash table entry for process P2.



If again P1 generates a child P3, then 'P1->child' now contains hash entry for P3 and 'P3->sibling‘ contains hash entry for P2.

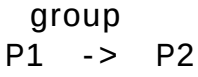


If 'sibling' is zero, it indicates that the process is at the end of the list. To walk through all siblings, start at
'Parent->child' and traverse till 'Current->sibling' is zero.
If 'child' is zero, it indicates that the process has no children

Note: Given the hash value the memory location of the process can always be found.

Tracking processes in the same group

If a process leader P1 and process P2 are of same group, then 'P1->group' contains the hash table entry for process P2.



If process P3 joins the group, then 'P1->group' now contains hash entry for P3 and 'P3->group' contains hash entry for P2.

```
          group     group
P1   ->    P3   ->   P2
```

If 'group' of a process is zero, it indicates that the process is at the end of the list. To walk through all the processes of the same group, start at the process whose 'pid' is equal to 'pgid' (i.e., process group leader according to the Unix standards), and traverse till 'Current->group' is zero.

Let us consider a scenario where a process leader dies.
Now the process group leader must not disappear for this to work. If the process group leader dies, process slot is not freed. Instead, it is marked as PROCSTATE_EXITED and kept until the last member of the group terminates. Also, reissuing of this pid must be prevented since this would create a duplicate pid situation (which is detected and warned). To do this, the exit code duplicates the process handle to the init process to keep the handle open when the process group leader is about to die (this is in DllMain ()). Thus the reference count to that process is incremented and the process (group leader) is not freed.
When the last process in the group dies, init process closes this handle. This message is given to the init process by message posting. This message is given to the wait thread of init.

Note: This scenario is only for group leader in groups.

Macros and function used

proc_hash() Generates the hash code from the pid supplied
proc_slot() Finds the block number(slot value) into which the given process
Falls.
proc_findlist() Searches the hash slot and returns the process structure whose pid matches the supplied pid.
proc_ptr() Gets the process structure for the supplied block number.

Code for generating the hash value

(Unsigned short) (pid&(Share->nproc-1))

pid refers to the pid of the process for which the hash value is to be generated.
nprocs refers to the count of number of processes in UWIN.

Code for getting the slot address given the slot number

```
(void*)((char*)Pproctab + ((slot)-1)*BLOCK_SIZE)
```

Pproctab is the pointer, which points to the starting point of the memory where the array information is stored.
BLOCK_SIZE by default is 512 bytes.

Pproctab[X] will get the process detail of the block whose slot number is X. This X is calculated by applying the hashing function on the pid.