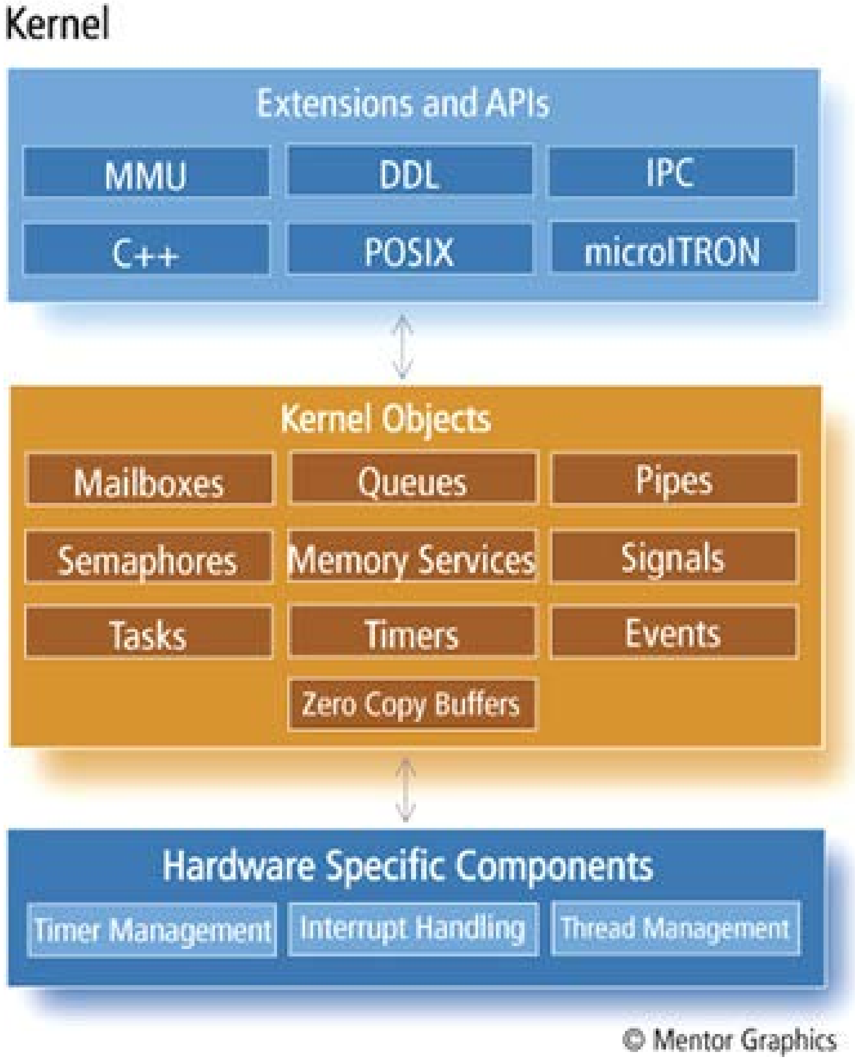
Example: Mentor Graphics POSIX Implementation (*Nucleus*)

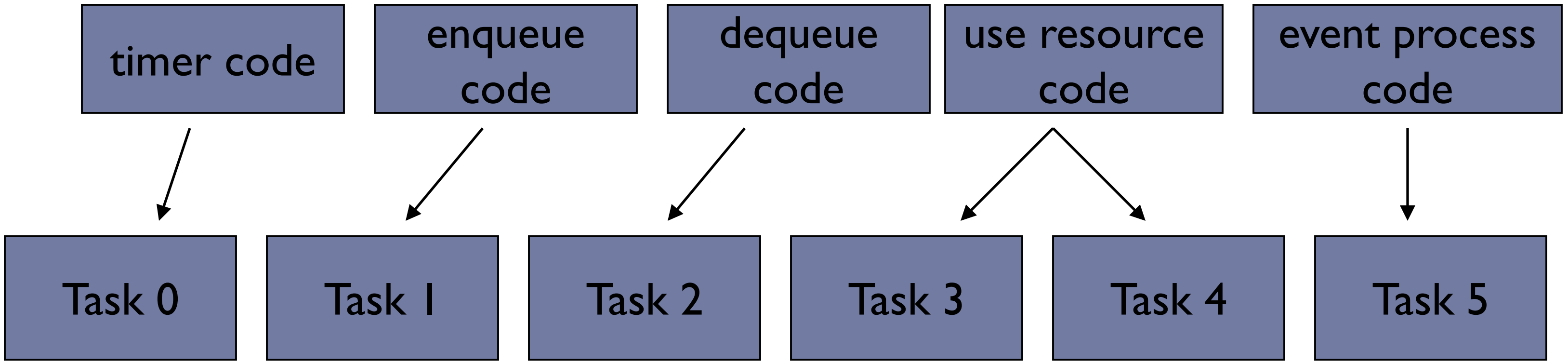
Mentor Graphics Nucleus User Guide

Nucleus POSIX Kernel



Nucleus POSIX demo program
(Mentor Graphics EDGE tools for SoC/ARM)

- Six tasks/five “functions”
  - a system timer
  - a task that queues messages
  - a task that retrieves queued messages
  - two instances of a task that uses a resource for 100 “ticks”
  - a task that waits for event signals



Demo program: Application structures

```
#include “nucleus.h”  /* OS function def’s */

/* Define Nucleus objects */
NU_TASK     Task_0;
NU_TASK     Task_1;
NU_TASK     Task_2;
NU_TASK     Task_3;
NU_TASK     Task_4;
NU_TASK     Task_5;
NU_QUEUE    Queue_0;
NU_SEMAPHORE Sempahore_0;
NU_EVENT_GROUP Event_Group_0;
NU_MEMORY_POOL System_Memory;
```

Demo program: Global variables

```
/* Define demo global variables */
UNSIGNED Task_Time;
UNSIGNED Task_1_messages_sent;
UNSIGNED Task_2_messages_received;
UNSIGNED Task_2_invalid_messages;
NU_TASK  *Who_has_the_resource;
UNSIGNED Event_Detections;
```

Nucleus POSIX process/task creation
(done by startup procedure)

```
/* Create a system memory pool to allocate to tasks */
status = NU_Create_Memory_Pool(
    &System_Memory,           //pointer to sys memory
    "SYSMEM",                 //name
    first_available_memory,   //address of 1st location
    SYSTEM_MEMORY_SIZE,       //size of allocated memory
    50,                       //minimum allocation
    NU_FIFO                   //if memory not available,
   );                         //resume tasks in FIFO order
```

Nucleus POSIX process/task creation
(done by startup procedure)

```
/* Create task 0 */
//allocates memory for task stack from memory pool
NU_Allocate_Memory(&System_Memory, &pointer,
                    TASK_STACK_SIZE, NU_NO_SUSPEND);


//create task activation record
status = NU_Create_Task(&Task_0, "TASK 0", task_0, 0,
              NU_NULL, pointer, TASK_STACK_SIZE, 1, 20,
              NU_PREEMPT, NU_START);
```

priority

time slice

Nucleus POSIX process/task creation
(done by startup procedure)

```
/* Create task 1 – Queue sending task*/
NU_Allocate_Memory(&System_Memory, &pointer,
                          TASK_STACK_SIZE, NU_NO_SUSPEND);



status = NU_Create_Task(&Task_2, "TASK 2", task_2, 0,
              NU_NULL, pointer, TASK_STACK_SIZE, 10, 5,
              NU_PREEMPT, NU_START);
```

priority

time slice

```
/* repeat for tasks 2-5 */
```

Demo program: System timer task

```
void task_0( )
{
  Task_Time = 0;
  while (1) {
    NU_Sleep (100); /*suspend for 100 ticks */

    Task_Time++;    /* increment time */

    /* set event flag to lift suspension of Task 5 */
    status = NU_Set_Events(&Event_Group_0,1,NU_OR);
  }
}
```

Demo program: Queue-sending task

```
void task_1( )
{
  Send_Message = 0;
  while (1) {
    /* queue a message */
    /* suspend if queue full or time slice */
    status = ND_Send_To_Queue(&Queue_0,
           &Send_Message, 1, NU_SUSPEND);
    Send_Message++;
  }
}
```

Demo program: Queue-receiving task

```
void task_2( )
{
  message_expected = 0;
  while (1) {
    /* retrieve a message */
    /* suspend if queue empty or time slice */
    status = ND_Receive_From_Queue(&Queue_0,
                &Receive_Message, 1, &received_size,
                NU_SUSPEND);
    message_expected++;
  }
}
```

Demo program: Use resource task

(two instances in the demo)

```
/* two tasks compete for use of a resource */
void tasks_3_and_4( )
{
  while (1) {
    /* set semaphore to lock resource */
    status = ND_Obtain_Semaphore(&Semaphore_0,
                                    NU_SUSPEND);
    if (status == NU_SUCCESS) {
       Who_has_resource = ND_Current_Task_Pointer();
       /* hold resource 100 ticks to suspend other task */
       NU_Sleep (100);
       NU_Release_Semaphore (&Semaphore_0);
     }
  }
}
```

Demo program: Wait for event to be set by Task 0

```
void task_5( )
{
  event_detections = 0;
  while (1) {
    /* wait for event and consume it */
    status = ND_Retrieve_Events(&Event_Group_0, 1,
           NU_OR_CONSUME, &event_group, NU_SUSPEND);
    if (status == NU_SUCCESS) {
       Event_Detections++;
     }
  }
}
```