Interprocess communication

Chapter 6.4

Interprocess communication

- Interprocess communication (IPC): OS provides mechanisms so that processes can pass data.
- Two types of semantics:
  - blocking: sending process waits for response;
  - non-blocking: sending process continues.

Interprocess communication mechanisms

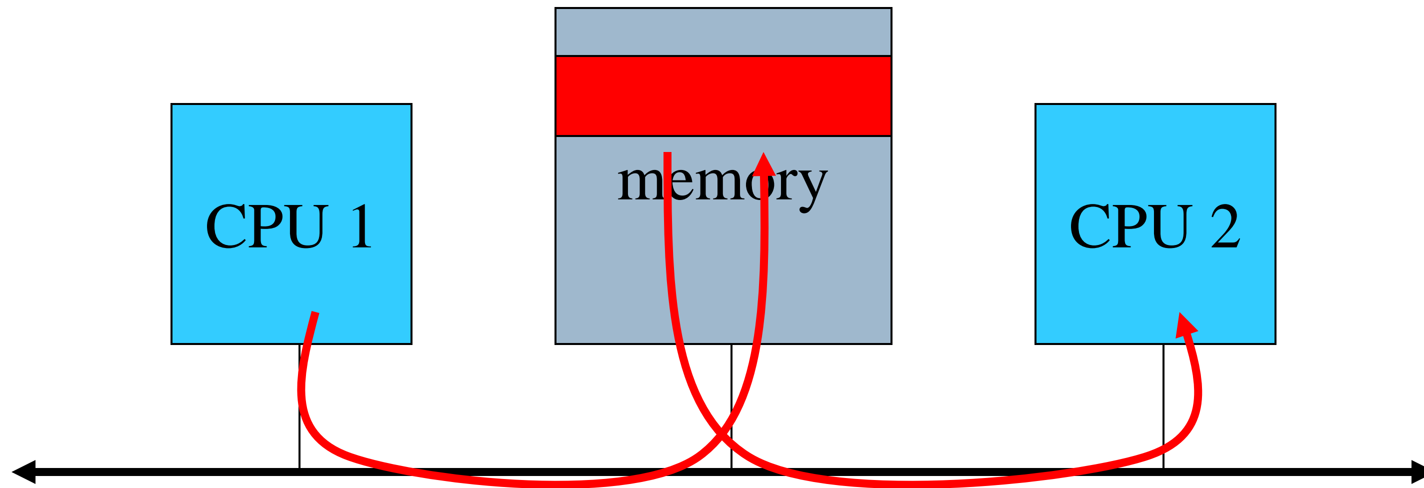
- Semaphores
  - Binary
  - counting
- Signals
- Mail boxes
- Queues
- Pipes

IPC styles

- Shared memory:
  - processes have some memory in common;
  - must cooperate to avoid destroying/missing messages.
- Message passing:
  - processes send messages along a communication channel---no common address space.
  - comm. channel may be physical or virtual

Shared memory

- Shared memory on a bus:



Race condition in shared memory

- Problem when two CPUs try to write the same location:
  - CPU 1 reads flag and sees 0.
  - CPU 2 reads flag and sees 0.
  - CPU 1 sets flag to one and writes location.
  - CPU 2 sets flag to one and overwrites location.

Atomic test-and-set

- Problem can be solved with an atomic test-and-set:
  - single bus operation reads memory location, tests it, writes it.
- ARM test-and-set provided by SWP (swap):

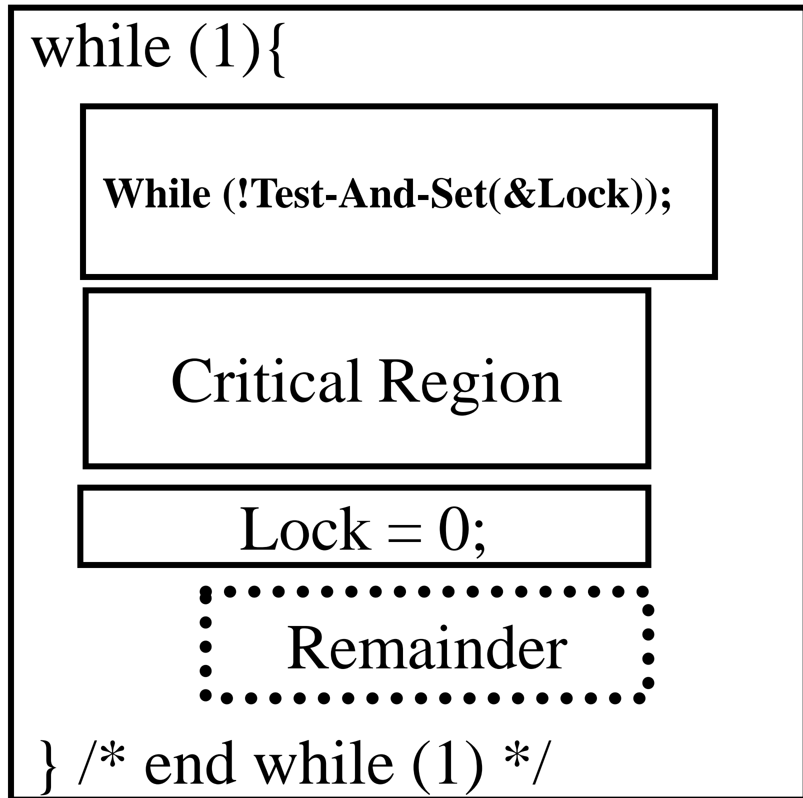
```
          ADR   r0,SEMAPHORE
          LDR   r1,#1
GETFLAG   SWP   r1,r1,[r0]
          BNZ   GETFLAG
```

Critical regions

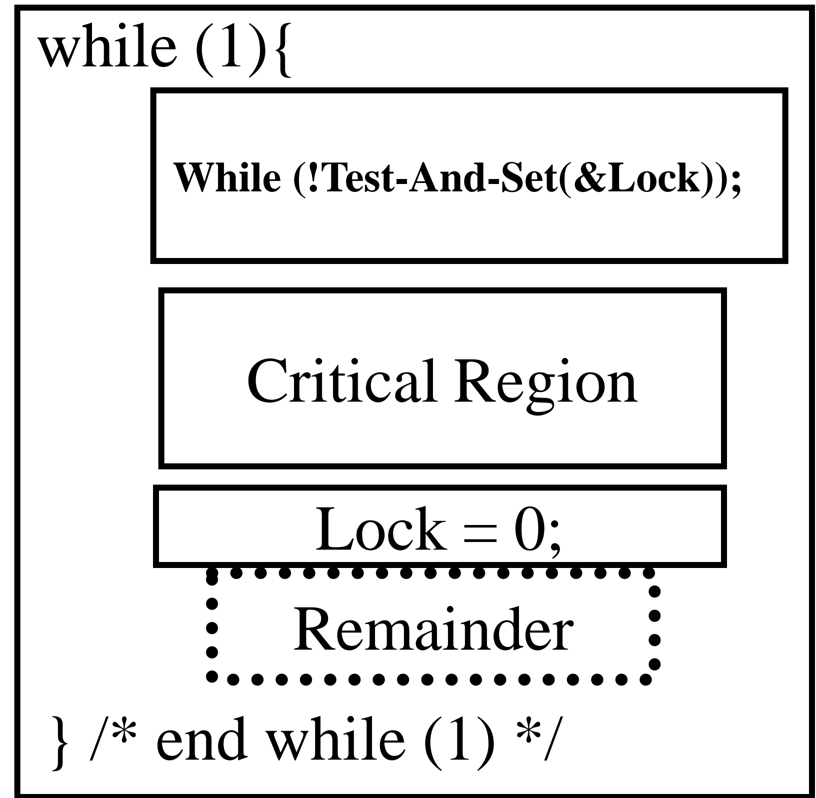
- Critical region: section of code that cannot be interrupted by another process.
- Examples:
  - writing shared memory;
  - accessing I/O device.

Mutual Exclusion Example

System variables: Lock = 0;



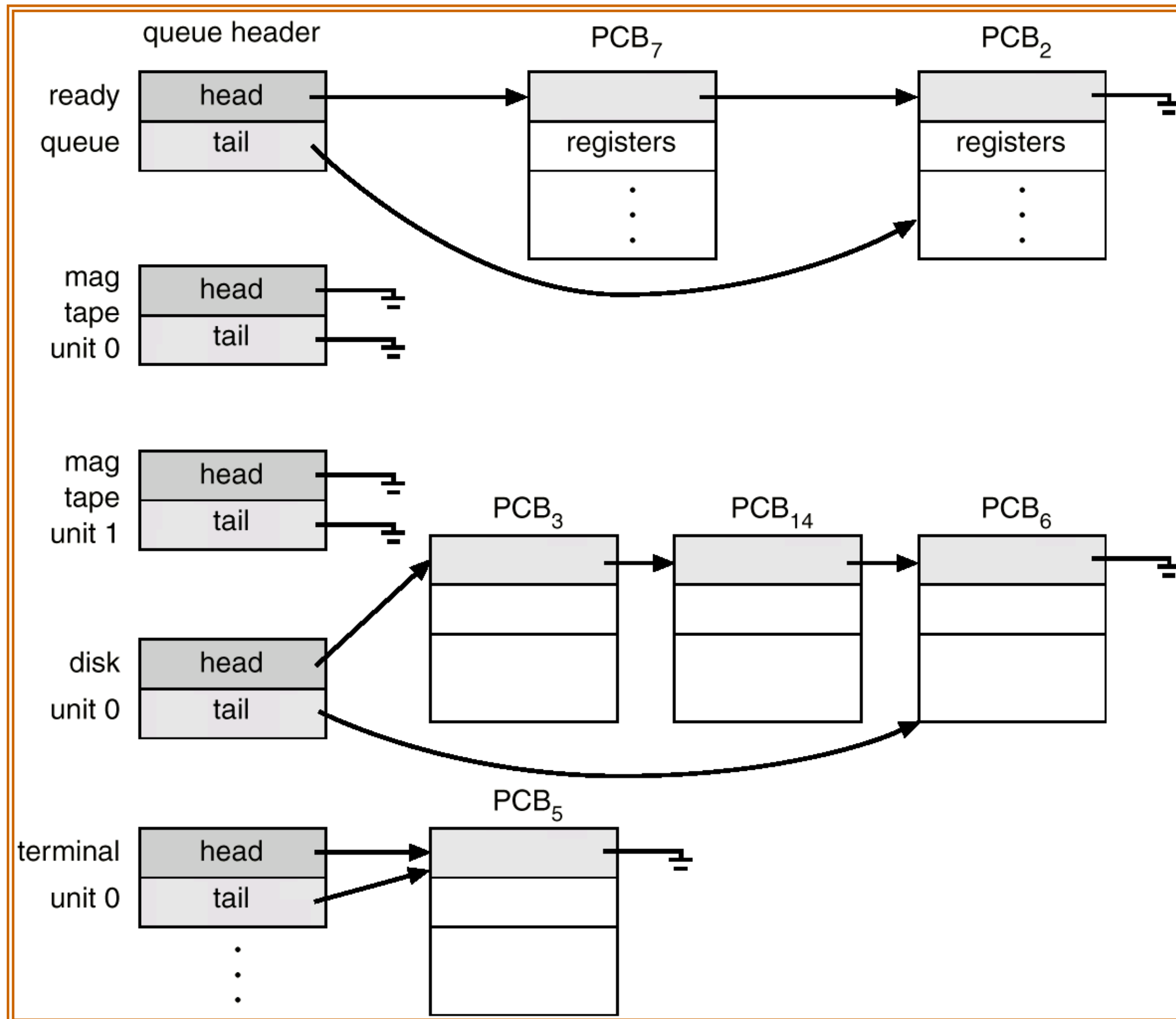
Process 1



Process 2

Task and Device Queues

Processes queued for shared device access



Semaphores

- Semaphore: OS primitive for controlling access to critical regions.
- Protocol:
  1. Get access to semaphore with P().
     (Dutch "Proberen" – to test)
  2. Perform critical region operations.
  3. Release semaphore with V().
     (Dutch "Verhogen" – to increment)

Binary semaphore

- Semaphore S values
  - S=1 : resource in use
  - S=0 : resource not in use
- Semaphore S actions
  - wait(&S) : test & set (read S, set S=1)
    use resource if read 0, o/w wait
  - signal(&S) : write S=0 to free up resource

Example

- Access critical region

```
wait(&S);        //continue if read S=1, o/w wait
//execute "critical region"
signal(&S);      //free the resource
```

- Task synchronization

```
Task1                 Task2
signal(&S1)           signal(&S2)
wait (&S2)            wait(&S1)
tasks synchronize at this point
```

Counting semaphore

- Semaphore S values
  - S=1 : resource free
  - S=0 : resource in use, no others waiting
  - S<0 : resource in use, others waiting
- Semaphore S actions
  - wait(&S) : S--, use resource if S=0, o/w wait
  - Signal(&S) : S++, wake up other task if S<1

Also use for access to N copies of a resource
– semaphore indicates number of copies free

Potential deadlock

- Tasks 1 and 2 each require two resources, R1 and R2, with access controlled by S1 and S2, respectively

Task1	Task2
wait(&S1)	wait(&S2)
//have R1	//have R2
wait (&S2)	wait(&S1)
//wait for R2	//wait for R1

DEADLOCK

POSIX semaphores

- POSIX supports counting semaphores with _POSIX_SEMAPHORES option.
  - Semaphore with N resources will not block until N processes hold the semaphore.
- Semaphores are given name:
  - Example: /sem1
- P() is sem_wait()
- V() is sem_post()

Semaphore example

```
int i, oflags;
sem_t *my_semaphore; //descriptor for sem.

//create the semaphore
my_semaphore = sem_open("/sem1",oflags);
  /* do useful work here */

//destroy the semaphore
i = sem_close(my_semaphore);
```

Semaphore example

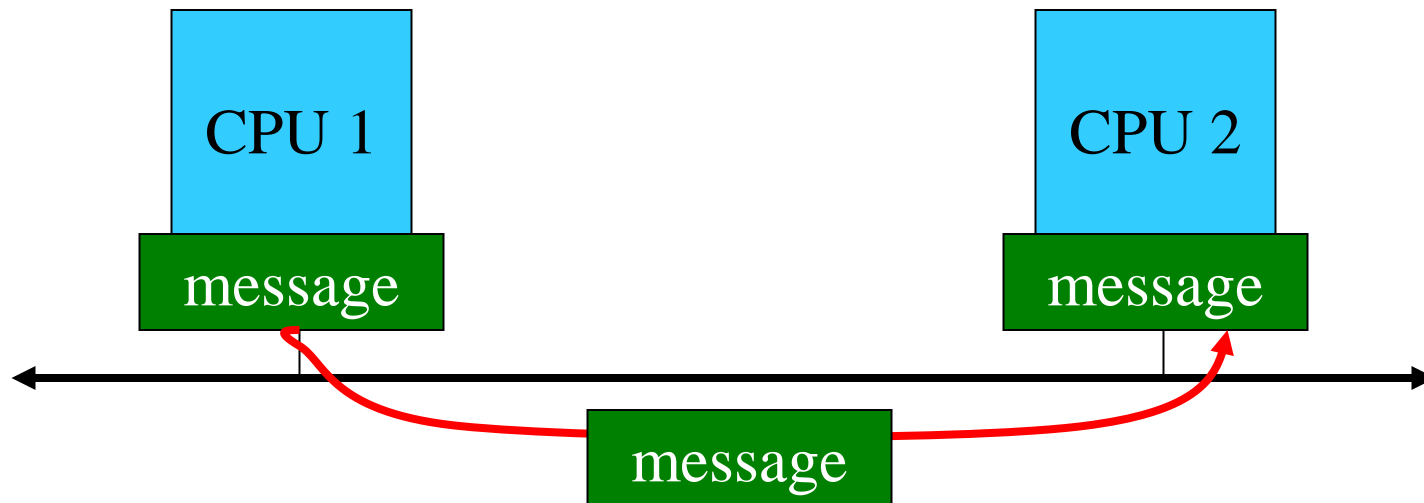
```
int i;
i = sem_wait(my_semaphore);    // P()
 // wait for semaphore, block if not free
 // now do useful work
i = sem_post(my_semaphore);    // V()



// test without blocking
i = sem_trywait(my_semaphore);
```

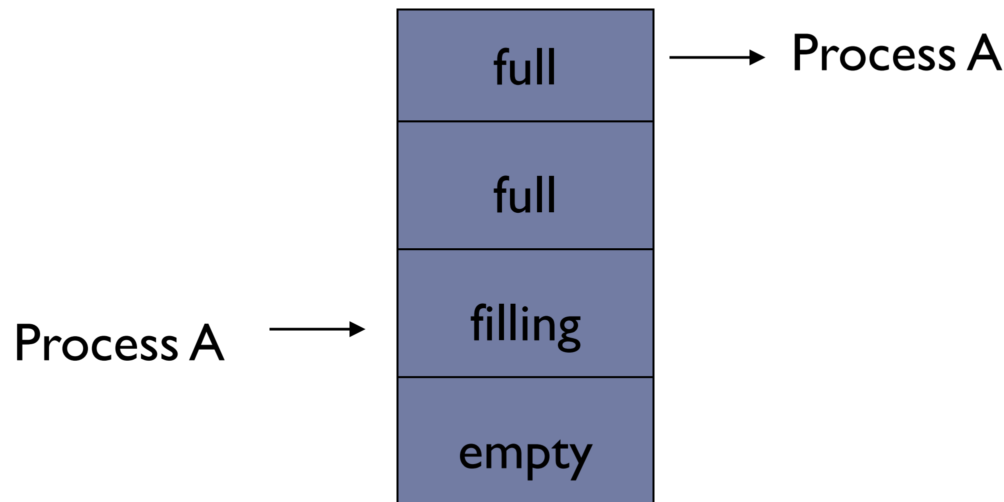
Message passing

- Message passing on a network:



Message passing via mailboxes

- Mailbox = message buffer between two processes (FIFO)



Use semaphore to lock buffer during read/write

MicroC/OS-II Mailboxes

- OSMboxCreate(msg)
  - create mail box & insert initial msg
- OSMboxPost(box, msg)
  - add msg to box
- OSMboxAccept(box)
  - get msg if there, o/w continue
- OSMboxPend(box,timeout)
  - get msg if there, o/w wait up to timeout
- OSMboxQuery(box,&data)
  - return information about the mailbox

Signals

- A Unix mechanism for simple communication between processes.
- Analogous to an interrupt---forces execution of a process at a given location.
  - But a signal is caused by one process with a function call.
- No data---can only pass type of signal.

POSIX signal types (partial list)

- SIGABRT: abort process
- SIGTERM: terminate process
- SIGFPE: floating point exception
- SIGILL: illegal instruction
- SIGKILL: unavoidable process termination
- SIGALRM: real-time clock expired
- SIGUSR1, SIGUSR2: user defined

POSIX signals

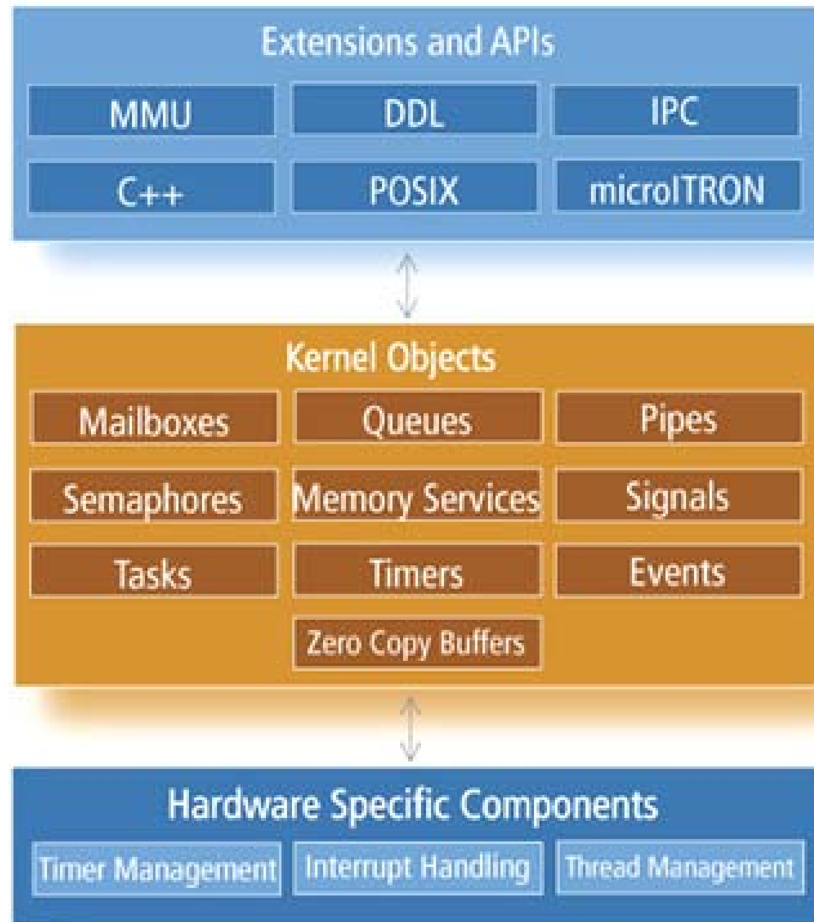
- Must declare a signal handler for the process using sigaction().
  - what to do when signal received
  - handler is called when signal is received

```
retval=sigaction(SIGUSR1,&act,&oldact);
```

- Send signal with sigqueue():

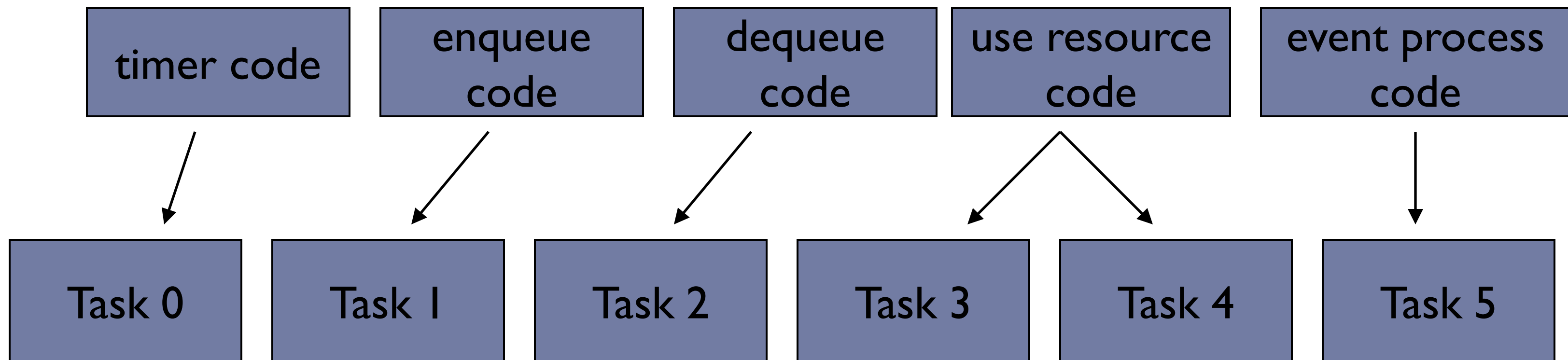
```
sigqueue(destpid,SIGRTMAX-1,sval);
```

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++;
     }
  }
}
```