EE458 - Embedded Systems
Introduction to uC/OS

• Outline
  – Introduction to uC/OS

• References
  – uC/OS RTOS Library Reference
Introduction to uC/OS

Background

- The source code to uC/OS was published in Embedded Systems Programming magazine in 1992 by Jean Labrosse. The version of uC/OS supplied with the Netburner is based on the original version. uC/OS evolved into uC/OS-II which is maintained by Micrium Inc.

- uC/OS is very efficient and has a small footprint. It runs nicely on ________ or small memory systems.
Introduction to uC/OS

uC/OS Tasks

- uC/OS task priorities can range from 1 (highest) to _______ (lowest). Each task MUST have a different priority.

- UserMain starts at priority 10 but by convention should change its priority to MAIN_PRIO (50). User tasks should be assigned priorities relative to MAIN_PRIO (for example MAIN_PRIO+1). User tasks should use priorities in the range 46 to 62 (MAIN_PRIO-4 to MAIN_PRIO+12).
Introduction to uC/OS

uC/OS Tasks

• There are several system tasks that run when networking is used. You must not use a priority assigned to one of these system tasks (refer to constants.h for the list).

• You (the programmer) must allocate stack space for each task that you create. This is usually done by defining a ____________:

```c
DWORD
    task_stacks[NTASKS][STKSIZE]
    __attribute__((aligned(4)));
```
Introduction to uC/OS

uC/OS Tasks

- The **OSTaskCreate()** function is used to create (and automatically start) new tasks:

  ```c
  BYTE OSTaskCreate(taskcode, taskdata, stack_top_ptr, stack_bot_ptr, priority);
  ```

- **taskcode** is a function pointer to the task function. **taskdata** is a 32-bit data type that is passed as an __________ to the task. **stack_top_ptr** and **stack_bot_ptr** are the addresses of the top and bottom of the stack. **priority** is the task priority.
Introduction to uC/OS
uC/OS Tasks

```c
void TaskCode(void *pd) { ...

#define STKSZE USER_TASK_STK_SIZE
DWORD
TaskStacks[2][STKSIZE] __attribute__((aligned(4)));

void UserMain(void *pd) {
    status = OSTaskCreate(TaskCode, NULL,
        &TaskStacks[0][STKSIZE], &TaskStacks[0][0],
        MAIN_PRIO+1);
    if(status != OS_NO_ERR) { ...
```
Introduction to uC/OS

uC/OS Tasks

- **OSSimpleTaskCreate()** is a ___________ that takes only the task name and priority as arguments. There is no return value that can be checked for errors. It auto creates a stack of size USER_TASK_STK_SIZE:

  OSSimpleTaskCreate(TaskCode, MAIN_PRIO+1);

- A task can delete itself with **OSTaskDelete()**. A task can change its own priority with **OSChangePrio(newprio)**.
Introduction to uC/OS

uC/OS Tasks

- Use `OSTimeDly(ticks)` to delay for a number of ticks. The macro `TICKS_PER_SECOND` is used in seconds to ticks conversions:
  
  ```c
  // Sleep for 5 seconds
  OSTimeDly(5 * TICKS_PER_SECOND);
  ```

- `OSLock()` and `OSUnlock()` can be used to protect ________________ by disabling and enabling task switching.
You must allocate and _______ a semaphore data structure for each semaphore:

// Create the semaphore data structure
// This must be visible to all tasks using the sem.
// and so is usually defined outside any function.
OS_SEM sync_sem;

// In UserMain, initialize the semaphore.
// Here it is initialized to zero.
e = OSSemInit(&sync_sem, 0);
Introduction to uC/OS

uC/OS Semaphores

• Use `OSSemPost()` to release (increment) the semaphore:

```c
e = OSSemPost(&sync_sem);
```

• Use `OSSemPend()` to acquire (decrement) the semaphore. This will block if the semaphore is zero. A time-out value (in ticks) is the second argument. A 0 time-out value implies that we will wait __________:

```c
e = OSSemPend(&sync_sem, 0);
```
Introduction to uC/OS

uC/OS Mailboxes

• A mailbox can be used to pass a single ______ data item between tasks.  (The item could be a pointer to an area of memory.) Similar to semaphores, you must allocate and initialize a mailbox data structure for each mailbox:

   OS_MBOX mboxes[NUMBOX];

// In UserMain (or other initialization) code
   e = OSMBoxInit(&mboxes[0], 0);
   e = OSMBoxInit(&mboxes[1], 0);
Introduction to uC/OS

uC/OS Mailboxes

- Use `OSMboxPost()` to place a message in the mailbox (does not block if full – check return status). Only one item may be in the mailbox at a time:

  ```
e = OSMboxPost(&mboxes[0], (void *)value);
  ```

- `OSMboxPend()` returns the item:

  ```
  // A return argument is used for __________
  value = (int)OSMboxPend(  
      &mboxes[0], timeout, &e);
  ```