

## Lab 6 overview

- You add
  - YKQCreate()
  - YKQPend()
  - YKQPost()
- You modify tick handler so that it posts one message to a queue each time it runs. (The queue is emptied by a task.)
- Higher priority task can hog CPU for 5-cycle intervals, letting messages queue up.
  - Task execution triggered by flag set in keypress ISR by any key.



©J Archibald

425 Lab 6.1

```

/* File: lab6defs.h */
#define MSGARRAYSIZE 20
struct msg
{
    int tick;
    int data;
};

/*
File: lab6app.c
Revision date: 4 November 2009
Description: Application code for EE 425 lab 6 (Message queues)
*/

#include "clib.h"
#include "yakk.h"
#include "lab6defs.h"

#define TASK_STACK_SIZE 512 /* stack size in words */
#define MSGQSIZE 10

struct msg MsgArray[MSGARRAYSIZE]; /* buffers for message content */

int ATaskStk[TASK_STACK_SIZE]; /* a stack for each task */
int BTaskStk[TASK_STACK_SIZE];
int STaskStk[TASK_STACK_SIZE];

int GlobalFlag;

void *MsgQ[MSGQSIZE]; /* space for message queue */
YKQ *MsgQPtr; /* actual name of queue */
    
```

Big enough that queue should not overflow.



©J Archibald

425 Lab 6.2

```

void ATask(void) /* processes data in msgs */
{
    struct msg *tmp;
    int min, max, count;

    min = 100;
    max = 0;
    count = 0;

    while (1)
    { /* get next msg */
        tmp = (struct msg *) YKQPend(MsgQPtr);

        /* check sequence count in msg;
        were msgs dropped? */
        if (tmp->tick != count+1)
        {
            printf(" Dropped msgs: tick ", 21);
            if (tmp->tick - (count+1) > 1) {
                printf(count+1);
                printfChar("-");
                printfInt(tmp->tick-1);
                printfNewLine();
            }
            else {
                printfInt(tmp->tick-1);
                printfNewLine();
            }
        }
    }
}
    
```

```

/* update sequence count */
count = tmp->tick;

/* process data; update statistics
for this sample */
if (tmp->data < min)
    min = tmp->data;
if (tmp->data > max)
    max = tmp->data;

/* output min, max, tick values */
printf("Ticks: ", 7);
printf(count);
printf("l", 1);
printf("min: ", 5);
printf(min);
printf("l", 1);
printf("Max: ", 5);
printf(max);
printfNewLine();
}
    
```

This task empties queue of messages generated by the tick handler, one message per tick.



©J Archibald

425 Lab 6.3

BTask is a CPU hog that keeps ATask from running. Its busy bursts are triggered by the press of any key and last for five clock ticks each.

```

void BTask(void) /* saturates CPU for 5 ticks */
{
    int busycount, curval, j, flag, chcount;
    unsigned tickNum;

    curval = 1001;
    chcount = 0;
    while (1)
    {
        YKDelayTask(2);
        if (GlobalFlag == 1)
        { /* flag set - loop for 5 ticks */
            YKEnterMutex();
            busycount = YKTickNum;
            YKExitMutex();
        }
    }
}
    
```

```

while (1) {
    YKEnterMutex();
    tickNum = YKTickNum;
    YKExitMutex();

    if (tickNum >= busycount + 5) break;

    curval += 2; /* evaluate next number */
    flag = 0;
    for (j = 3; (j) < curval; j += 2) {
        if (curval % j == 0) {
            flag = 1;
            break;
        }
    }
    if (!flag) {
        printfChar("."); /* output marker for each # */
        if (++chcount > 75) {
            printfNewLine();
            chcount = 0;
        }
    }
    printfNewLine();
    chcount = 0;
    GlobalFlag = 0; /* clear flag */
}
}
}
    
```



©J Archibald

425 Lab 6.4

```

void STask(void) /* tracks statistics */
{
    unsigned max, switchCount, idleCount;
    int tmp;

    YKDelayTask(1);
    printfString("Welcome to the YAK kernel\n");
    printfString("Determining CPU capacity\n");
    YKDelayTask(1);
    YKIdleCount = 0;
    YKDelayTask(5);
    max = YKIdleCount / 25;
    YKIdleCount = 0;
    YKNewTask(BTask, (void *) &BTaskStk[TASK_STACK_SIZE], 10);
    YKNewTask(ATask, (void *) &ATaskStk[TASK_STACK_SIZE], 20);
    while (1) {
        YKDelayTask(20);
        YKEnterMutex();
        switchCount = YKCtxSwCount;
        idleCount = YKIdleCount;
        YKExitMutex();
        printfString("<<<<< Context switches: ");
        printfInt(switchCount);
        printfString(", CPU usage: ");
        tmp = (int) idleCount/max;
        printfInt(100-tmp);
        printfString("%>>>>\n");
        YKEnterMutex();
        YKCtxSwCount = 0;
        YKIdleCount = 0;
        YKExitMutex();
    }
}
    
```

The stat task, just like lab 5. Starts tasks A and B. Prints statistics every 20 ticks.



©J Archibald

425 Lab 6.5

```

void main(void)
{
    YKInitialize();

    /* create queue, at least one user task, etc. */
    GlobalFlag = 0;
    MsgQPtr = YKQCreate(MsgQ, MSGQSIZE);
    YKNewTask(STask, (void *) &STaskStk[TASK_STACK_SIZE], 30);

    YKRun();
}
    
```



©J Archibald

425 Lab 6.6

## YAK functions for Lab 6

### YKQ \*YKQCreate(void \*\*start, unsigned size)

- Must be called exactly once per message queue, ideally in `main()`.
- Initializes data structure used to maintain queue; returns pointer to it.
- YKQ is typedef defined in kernel header file; defines struct that records
  - the base address of the queue,
  - the number of entries in the queue,
  - the next empty slot to use,
  - the next message to remove, etc.
- Parameters:
  - **start**: specifies base address of the queue itself, an array of void pointers.
  - **size**: specifies number of entries in queue, size of the array.
- Each queue entry is a void pointer
  - User code can use in arbitrary way, but also must manage what they point to.



425 Lab 6.7

## YAK functions for Lab 6

### void \*YKQPend(YKQ \*queue)

- If message queue non-empty, removes and returns the oldest message.
- If message queue empty, calling task is suspended by the kernel until a message is placed in queue.
- Called only by tasks, never by interrupt code.



425 Lab 6.8

## YAK functions for Lab 6

### int YKQPost(YKQ \*queue, void \*msg)

- Inserts a message (pointer) into a message queue.
- Parameters:
  - **queue**: pointer to queue to use.
  - **msg**: pointer to the message.
- Return value:
  - 1 if queue is not full and message insertion was successful.
  - 0 if queue is full and message was not inserted.
- If suspended tasks are waiting for a message from this queue, the highest priority waiting task is made ready to run.
- This function may be called from both task and interrupt code.
  - If called from interrupt code, *it must not call the scheduler.*



425 Lab 6.9

## Lab 6: Changes to interrupt handlers

- Keypress handler
  - No longer generates output; just sets **GlobalFlag** to 1.
- Reset handler
  - No modification from previous labs.
- User code tick handler
  - No longer generates "--TICK X--" output; instead, puts a message into the message queue each time it runs.
  - You are given the code for this. (See next slide.)



425 Lab 6.10

```

/* File: lab6inth.c
Revision date: 4 November 2009
Description: Sample interrupt handler code for EE 425 lab 6 (Message queues) */

#include "lab6defs.h"
#include "yak.h"
#include "clib.h"

extern YKQ *MsgQPtr;
extern struct msg MsgArray[];
extern int GlobalFlag;

void myreset(void) {
    exit(0);
}

void mykeybrd(void) {
    GlobalFlag = 1;
}

void mytick(void) {
    static int next = 0;
    static int data = 0;
    /* create a message with tick (sequence #) and pseudo-random data */
    MsgArray[next].tick = YKTickNum;
    data = (data + 89) % 100;
    MsgArray[next].data = data;
    if (YKQPost(MsgQPtr, (void *) &(MsgArray[next])) == 0)
        printf("TickISR: queue overflow\n");
    else if (++next >= MSGARRAYSIZE)
        next = 0;
}
    
```

```

/* File: lab6defs.h */
#define MSGARRAYSIZE 20

struct msg
{
    int tick;
    int data;
};
    
```



425 Lab 6.11

Welcome to the YAK kernel

Determining CPU capacity

Ticks: 1	Min: 89	Max: 89
Ticks: 2	Min: 78	Max: 89
Ticks: 3	Min: 67	Max: 89
Ticks: 4	Min: 56	Max: 89
Ticks: 5	Min: 45	Max: 89
Ticks: 6	Min: 34	Max: 89
Ticks: 7	Min: 23	Max: 89
Ticks: 8	Min: 12	Max: 89
Ticks: 9	Min: 1	Max: 89
Ticks: 10	Min: 1	Max: 90
Ticks: 11	Min: 1	Max: 90
Ticks: 12	Min: 1	Max: 90
Ticks: 13	Min: 1	Max: 90
Ticks: 14	Min: 1	Max: 90
Ticks: 15	Min: 1	Max: 90
Ticks: 16	Min: 1	Max: 90
Ticks: 17	Min: 1	Max: 90
Ticks: 18	Min: 1	Max: 90
Ticks: 19	Min: 1	Max: 91
Ticks: 20	Min: 1	Max: 91

Ticks: 21 Min: 1 Max: 91

Ticks: 22 Min: 1 Max: 91

Ticks: 23 Min: 1 Max: 91

Ticks: 24 Min: 1 Max: 91

Ticks: 25 Min: 1 Max: 91

Ticks: 26 Min: 1 Max: 91

Ticks: 27 Min: 1 Max: 91

<<<< Context switches: 62, CPU usage: 11% >>>>

Ticks: 28 Min: 1 Max: 92

Ticks: 29 Min: 1 Max: 92

Sample output:  
default tick rate

### Requirements:

- No messages dropped at default tick rate.
- Stat task output is as expected.

Key is pressed,  
Keypress ISR sets GlobalFlag,  
BTask hogs CPU for 5 tick intervals



425 Lab 6.12

Ticks: 175 Min: 0 Max: 99  
 Ticks: 176 Min: 0 Max: 99  
 Ticks: 177 Min: 0 TickISR: queue overflow!  
 Max: 99  
 I Dropped msgs: tick 178  
 Ticks: 179 Min: 0 Max: TickISR: queue overflow!  
 99  
 I Dropped msgs: tick 180  
 Ticks: 181 Min: 0 Max: TickISR: queue overflow!  
 99  
 I Dropped msgs: tick 182  
 Ticks: 183 Min: 0 Max: 99  
 Ticks: 184 Min: 0 Max: 99  
 TickISR: queue overflow!  
 I Dropped msgs: tick 185  
 Ticks: 186 Min: 0 Max: 99  
 Ticks: 187 Min: 0 Max: TickISR: queue overflow!  
 99  
 I Dropped msgs: tick 188  
 Ticks: 189 Min: 0 Max: TickISR: queue overflow!  
 99  
 I Dropped msgs: tick 190  
 Ticks: 191 Min: 0 Max: 99  
 Ticks: 192 Min: 0 Max: 99  
 Ticks: 193 Min: 0 Max: TickISR: queue overflow!  
 Max: 99  
 I Dropped msgs: tick 194  
 Ticks: 195 Min: 0 Max: TickISR: queue overflow!  
 99  
 I Dropped msgs: tick 196

Sample output:  
 750 instr/tick

**Requirements:**

- At 750 instr/tick, code doesn't crash
- Task still gets *some* entries from queue if CPU hog not running, but some entries may be dropped.
- May not see stat task output

BYU ©J Archibald 425 Lab 6.13

