

Lab 5 overview

- Add to kernel:
 - Semaphore routines
 - YKSemCreate: create and initialize semaphore
 - YKSemPost: post to semaphore
 - YKSemPend: pend on semaphore
 - Mechanism to track utilization
 - Idle task increments YKIdleCount, an unsigned int
 - Stat task reads, resets periodically
 - Ratio of count to max gives fraction of CPU not used.



425 Lab 5.1

Lab 5 application: declarations

```

/* File: lab5app.c
Revision date: 13 November 2009
Description: Application code for EE 425 lab 5 (Semaphores) */

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

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

int TaskWSM[TASK_STACK_SIZE]; /* stacks for each task */
int TaskSSM[TASK_STACK_SIZE];
int TaskPSM[TASK_STACK_SIZE];
int TaskStatStk[TASK_STACK_SIZE];
int TaskPRMSM[TASK_STACK_SIZE];

YKSEM *PSemPtr; /* YKSEM must be defined in yakk.h */
YKSEM *SSemPtr;
YKSEM *WSemPtr;
YKSEM *NSemPtr;
    
```



425 Lab 5.2

Three tasks

```

void TaskWord(void)
{
    while (1)
    {
        YKSemPend(WSemPtr);
        printf("Hey");
        YKSemPost(WSemPtr);

        YKSemPend(WSemPtr);
        printf("ll");
        YKSemPost(WSemPtr);

        YKSemPend(WSemPtr);
        printf("works");
        YKSemPost(WSemPtr);
    }
}

void TaskSpace(void)
{
    while (1)
    {
        YKSemPend(SSemPtr);
        printf(" ");
        YKSemPost(WSemPtr);
    }
}
    
```

```

void TaskPunc(void)
{
    while (1)
    {
        YKSemPend(PSemPtr);
        printf("!");
        YKSemPost(WSemPtr);

        YKSemPend(PSemPtr);
        printf("!");
        YKSemPost(SSemPtr);

        YKSemPend(PSemPtr);
        printf("!");
        YKSemPost(PSemPtr);

        YKDelayTask(6);
    }
}
    
```



425 Lab 5.3

A compute hog

```

void TaskPrime(void)
{
    int curval = 1001;
    int j, flag, incnt;
    int endval;
    while (1)
    {
        YKSemPend(NSemPtr);
        /* compute next range of primes */
        incnt = 0;
        endval = curval + 500;
        for ( ; curval < endval; curval += 2)
        {
            flag = 0;
            for (j = 3; (j*j) < curval; j += 2)
            {
                if (curval % j == 0)
                {
                    flag = 1;
                    break;
                }
            }
        }
    }
}
    
```

```

if (!flag)
{
    printf(" ");
    printf("%d", curval);
    incnt++;
    if (incnt > 9)
    {
        printf("\n");
        incnt = 0;
    }
}
printf("\n");
}
}
    
```

- When 'p' pressed, your keypress handler calls YKSemPost(NSemPtr).
- For each 'p', the next 500 numbers are checked for primes.
- Press 'p' multiple times to push utilization to 100%.



425 Lab 5.4

The mother task

```

void TaskStat(void) /* a task to track statistics */
{
    unsigned max, switchCount, idleCount;
    int tmp;
    YKDelayTask(1);
    printf("Welcome to the YAK kernel!\n");
    printf("Determining CPU capacity!\n");
    YKDelayTask(1);
    YKIdleCount = 0;
    YKDelayTask(5);
    max = YKIdleCount / 25;
    YKIdleCount = 0;
    YKNewTask[TaskPrime, (void *) &TaskPRMSM[TASK_STACK_SIZE], 32];
    YKNewTask[TaskWord, (void *) &TaskWSM[TASK_STACK_SIZE], 10];
    YKNewTask[TaskSpace, (void *) &TaskSSM[TASK_STACK_SIZE], 11];
    YKNewTask[TaskPunc, (void *) &TaskPSM[TASK_STACK_SIZE], 12];
    while (1) {
        YKDelayTask(20);
        YKEnterMutex();
        switchCount = YKCtxSwCount; idleCount = YKIdleCount;
        YKExitMutex();
        printf("==== Context switches: ");
        printf("%d\n", switchCount);
        printf("CPU usage: ");
        tmp = (int) (idleCount/max);
        printf("%d\n", tmp);
        printf("====\n");
        YKEnterMutex();
        YKCtxSwCount = 0; YKIdleCount = 0;
        YKExitMutex();
    }
}
    
```

$\% \text{ Utilization} = 100 - 100 * (\text{currentCount}/(\text{baseCount}^4)) = 100 - \text{currentCount}/(\text{baseCount}/25)$



425 Lab 5.5

The main routine

```

void main(void)
{
    YKInitialize();
    /* create all semaphores, at least one user task, etc. */
    PSemPtr = YKSemCreate(1);
    SSemPtr = YKSemCreate(0);
    WSemPtr = YKSemCreate(0);
    NSemPtr = YKSemCreate(0);
    YKNewTask[TaskStat, (void *) &TaskStatStk[TASK_STACK_SIZE], 30];
    YKRun();
}
    
```



425 Lab 5.6

What is output?

```

10 void TaskWord(void)
{
  while (1)
  {
    YKSemPend(WSemPtr);
    printString("Hey");
    YKSemPost(PSemPtr);

    YKSemPend(WSemPtr);
    printString("H");
    YKSemPost(SSemPtr);

    YKSemPend(WSemPtr);
    printString("works");
    YKSemPost(PSemPtr);
  }
}

11 void TaskSpace(void)
{
  while (1)
  {
    YKSemPend(SSemPtr);
    printChar(" ");
    YKSemPost(WSemPtr);
  }
}

```

```

12 void TaskPunc(void)
{
  while (1)
  {
    YKSemPend(PSemPtr);
    printChar("");
    YKSemPost(WSemPtr);

    YKSemPend(PSemPtr);
    printChar(",");
    YKSemPost(SSemPtr);

    YKSemPend(PSemPtr);
    printString("!\\n");
    YKSemPost(PSemPtr);
  }
}

```

Semaphore status:

PSem: 1

SSem: -1

WSem: -1

Output:

"Hey, it works!"

425 Lab 5.7

Output without pressing 'p'

```

--TICK 1--
Welcome to the YAK kernel
Determining CPU capacity
--TICK 2--
--TICK 3--
--TICK 4--
--TICK 5--
--TICK 6--
--TICK 7--
"Hey, it works!"
--TICK 8--
--TICK 9--
--TICK 10--
--TICK 11--
"Hey, it works!"
--TICK 12--
--TICK 13--
"Hey, it works!"
--TICK 14--
--TICK 15--
--TICK 16--
--TICK 17--
--TICK 18--
--TICK 19--
--TICK 20--
--TICK 21--
--TICK 22--
--TICK 23--
--TICK 24--
--TICK 25--
--TICK 26--
--TICK 27--
<<<<< Context switches: 54, CPU usage: 5% >>>>>

```

Output after pressing 'p'

```

--TICK 36--
3659 3671 3673 3677 3691 3697 3701 3709 3719
3741 3773 3775
--TICK 37--
"Hey, it works!"
3763 3767 3769 3779 3793 3797
3803 3807 3823 3833
--TICK 38--
3847 3851 3853 3863 3877 3881
3889 3907 3911 3917 3919 3923 3929 3931
--TICK 39--
3943 3947
3967 3969
4002 4003 4007 4013 4019 4021 4027
--TICK 40--
4048 4051 4057
4073 4079 4081 4083 4089 4111 4127 4129 4133 4139 --TICK 41--
4153 4157 4159 4177 4201 4211 4217 4219 4229 4231
4241
--TICK 42--
4243 4253 4259 4261 4271 4273 4283 4289 4297
4327 4331 4339
--TICK 43--
"Hey, it works!"
4348 4357 4363 4373 4391 4397 4409
4421 4423
--TICK 44--
4441 4447 4451 4457 4463 4481 4483 4489
4493
--TICK 45--
--TICK 46--
--TICK 47--
<<<<< Context switches: 39, CPU usage: 83% >>>>>

```

425 Lab 5.8

YKSEM

- YKSEM is the struct for a semaphore
- Defined in "yakk.h" (you define this structure in your kernel)
- What sort of data representation makes sense?
- Each semaphore needs:
 - An integer value.
 - Possibly: a TCB pointer pointing to a linked list of tasks blocked on this semaphore, perhaps sorted by priority order.
- Created and initialized by YKSemCreate.
 - One integer parameter, initial value of semaphore (≥ 0)
- Memory management recommendation:
 - Preallocate an array of YKSEMs, similar to handling TCBS.

425 Lab 5.9