

Lab 5 overview

- Add to kernel:
 - Semaphore functions
 - YKSEM* **YKSemCreate**(int initialValue)
 - void **YKSemPost**(YKSEM *semaphore)
 - void **YKSemPend**(YKSEM *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
Description: Application code for ECE425 lab 5 (Semaphores) */

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

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

int TaskWSHk[TASK_STACK_SIZE]; /* stacks for each task */
int TaskSSHk[TASK_STACK_SIZE];
int TaskPSHk[TASK_STACK_SIZE];
int TaskStatSk[TASK_STACK_SIZE];
int TaskPRMSk[TASK_STACK_SIZE];

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



425 Lab 5.2

Three tasks that generate output

```

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

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

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

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("!win");
        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 (j = curval; j < endval; curval += 2)
        {
            flag = 0;
            for (i = 3; (i*i) < curval; i += 2)
            {
                if (curval % i == 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 *) &TaskPRMSk[TASK_STACK_SIZE], 32];
    YKNewTask[TaskWord, (void *) &TaskWSHk[TASK_STACK_SIZE], 10];
    YKNewTask[TaskSpace, (void *) &TaskSSHk[TASK_STACK_SIZE], 11];
    YKNewTask[TaskPunc, (void *) &TaskPSHk[TASK_STACK_SIZE], 12];
    while (1) {
        YKDelayTask(20);
        YKEnterMutex();
        switchCount = YKCntSwCount; idleCount = YKIdleCount;
        YKExitMutex();
        printf("==== Context switches: ");
        printf("%d\n", switchCount);
        printf("CPU usage: ");
        tmp = (int) (idleCount/max);
        printf("%d\n", tmp);
        printf("====\n");
        YKEnterMutex();
        YKCntSwCount = 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 *) &TaskStatSk[TASK_STACK_SIZE], 30];
    YKRun();
}
    
```



425 Lab 5.6

Output without pressing 'p'	Output after pressing 'p'
<pre> --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 15-- "Hey, it works!" ... --TICK 26-- <<<<< Context switches: 54, CPU usage: 5% >>>>> ... </pre>	<pre> --TICK 36-- 3659 3671 3673 3677 3691 3697 3701 3709 3719 3721 3723 3725 3729 --TICK 37-- "Hey, it works!" 3761 3767 3769 3779 3793 3797 3803 3824 3826 3833 --TICK 38-- 3847 3851 3853 3863 3877 3891 3899 3907 3911 3917 3919 3923 3929 3931 --TICK 39-- 3943 3947 3967 3989 4001 4003 4007 4013 4019 4021 4027 --TICK 40-- 4049 4051 4057 4073 4079 4091 4093 4099 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 4339 4359 --TICK 43-- "Hey, it works!" 4349 4357 4363 4373 4391 4397 4409 4421 4423 --TICK 44-- 4441 4447 4451 4457 4463 4461 4483 4489 4493 --TICK 45-- --TICK 46-- --TICK 47-- <<<<< Context switches: 39, CPU usage: 83% >>>>> </pre>

YKSEM

- YKSEM is the [struct for a semaphore](#)
- Defined in "yakk.h" (you define and use this struct 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.
- YKSEM struct 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.