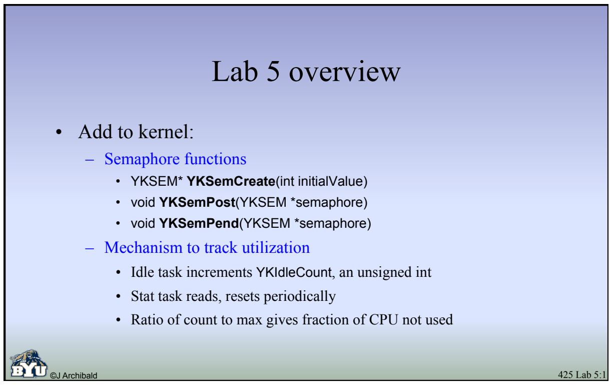
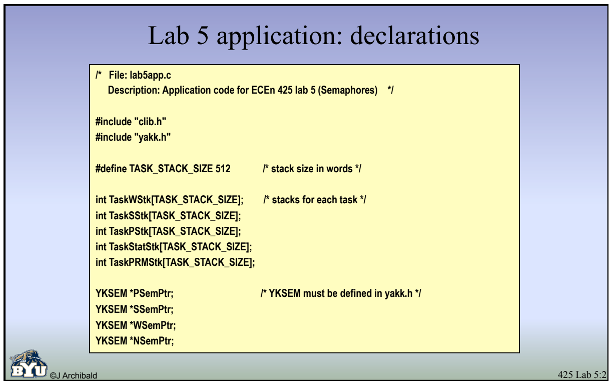


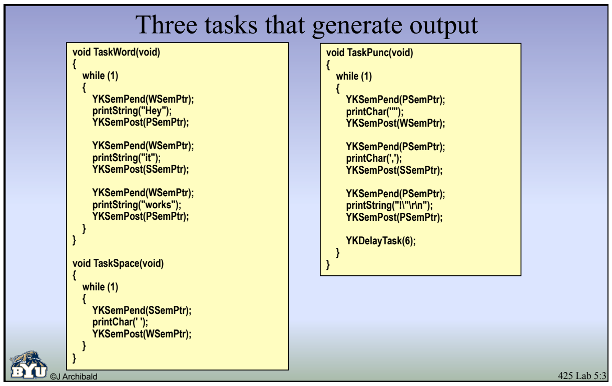
Lab 5 overview

- 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



Lab 5 application: declarations

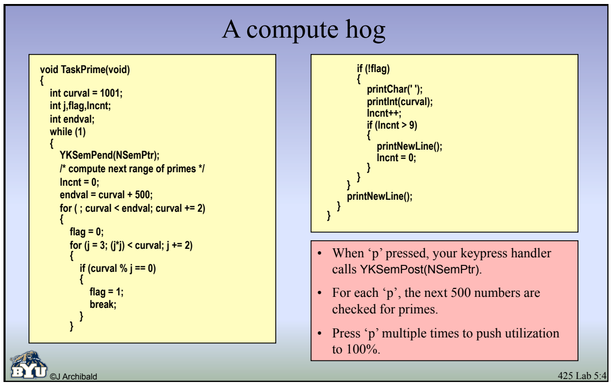
```
/*   File: lab5app.c
     Description: Application code for ECEn 425 lab 5 (Semaphores)    */

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

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

int TaskWStk[TASK_STACK_SIZE];        /* stacks for each task */
int TaskSStk[TASK_STACK_SIZE];
int TaskPStk[TASK_STACK_SIZE];
int TaskStatStk[TASK_STACK_SIZE];
int TaskPRMStk[TASK_STACK_SIZE];

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



Three tasks that generate output

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

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

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

void TaskSpace(void)
{
   while (1)
   {
      YKSemPend(SSemPtr);
      printChar(' ');
      YKSemPost(WSemPtr);
   }
}
```

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

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

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

      YKDelayTask(6);
   }
}
```

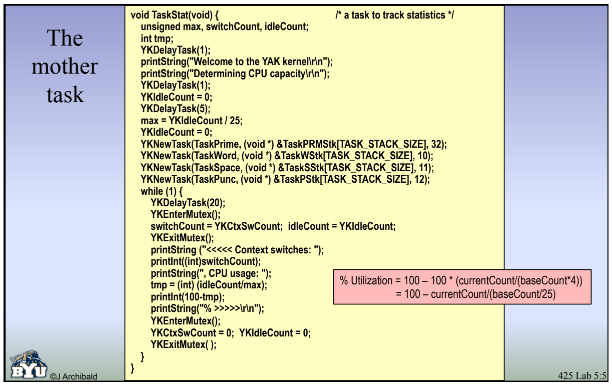


A compute hog

```
void TaskPrime(void)
{
   int curval = 1001;
   int j,flag,lncnt;
   int endval;
   while (1)
   {
      YKSemPend(NSemPtr);
      /* compute next range of primes */
      lncnt = 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)
         {
            printChar(' ');
            printInt(curval);
            lncnt++;
            if (lncnt > 9)
            {
               printNewLine();
               lncnt = 0;
            }
         }
      }
      printNewLine();
   }
}
```

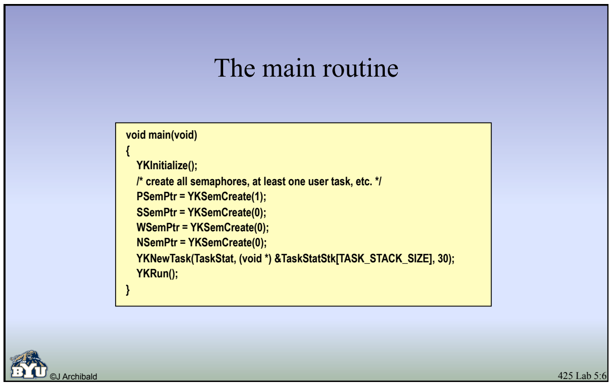
- When ‘p’ pressed, your keypress handler calls YKSemPost(NSemPtr).
- For each ‘p’, the next 500 numbers are checked for primes.
- Press ‘p’ multiple times to push utilization to 100%.



The mother task

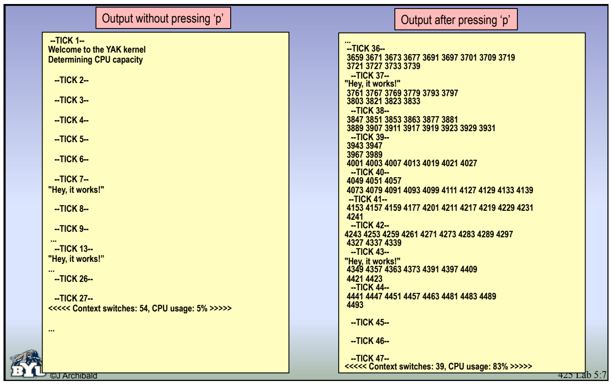
```
void TaskStat(void) {                          /* a task to track statistics */
   unsigned max, switchCount, idleCount;
   int tmp;
   YKDelayTask(1);
   printString("Welcome to the YAK kernel\r\n");
   printString("Determining CPU capacity\r\n");
   YKDelayTask(1);
   YKIdleCount = 0;
   YKDelayTask(5);
   max = YKIdleCount / 25;
   YKIdleCount = 0;
   YKNewTask(TaskPrime, (void *) &TaskPRMStk[TASK_STACK_SIZE], 32);
   YKNewTask(TaskWord, (void *) &TaskWStk[TASK_STACK_SIZE], 10);
   YKNewTask(TaskSpace, (void *) &TaskSStk[TASK_STACK_SIZE], 11);
   YKNewTask(TaskPunc, (void *) &TaskPStk[TASK_STACK_SIZE], 12);
   while (1) {
      YKDelayTask(20);
      YKEnterMutex();
      switchCount = YKCtxSwCount;  idleCount = YKIdleCount;
      YKExitMutex();
      printString ("<<<<< Context switches: ");
      printInt((int)switchCount);
      printString(", CPU usage: ");
      tmp = (int) (idleCount/max);
      printInt(100-tmp);
      printString("% >>>>>\r\n");
      YKEnterMutex();
      YKCtxSwCount = 0;  YKIdleCount = 0;
      YKExitMutex( );
   }
}
```

% Utilization = 100 – 100 * (currentCount/(baseCount*4))
= 100 – currentCount/(baseCount/25)



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();
}
```

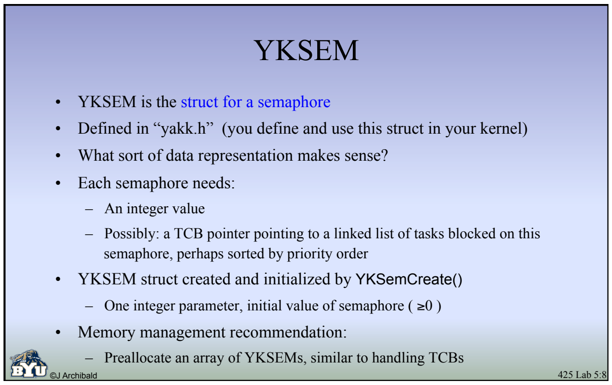


Output without pressing ‘p’

```
 --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 13--
"Hey, it works!”
...
  --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
 3721 3727 3733 3739
  --TICK 37--
"Hey, it works!"
 3761 3767 3769 3779 3793 3797
 3803 3821 3823 3833
  --TICK 38--
 3847 3851 3853 3863 3877 3881
 3889 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 4337 4339
  --TICK 43--
"Hey, it works!"
 4349 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% >>>>>
```



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