**CEG 4350/6350 Final Exam. Summer 2020**

Open Book, notes, and handouts (9:50 – 11:55 am)

Alex Reigle

**1. (15 points)**

Suppose there are two types of resources – resource type *R* and resource type *S*. Resource type *R* has *m* identical instances, and resource type *S* has *n* identical instances. A number of processes can share these instances as follows:

1. Each instance (of resource types *R* and *S*) can be used by only one process at a time.
2. Each process needs to acquire and use any one instance of *R* and any one instance of *S* at the same time. Thus, if there is no instance of resource type *R* or *S* (or both) available for a requesting process, the process has to wait until both instances become available.

Design a monitor implementing Acquire() and Release() functions that behave as follows (in atomic manner); and declare and initialize as many variables as needed.

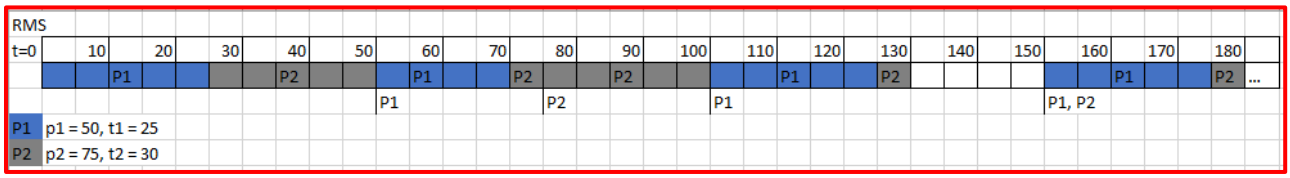
**1.**

```
Monitor ResourceAllocate{
//Monitiors by definition satisfy mutual exclusion
Semaphore semS;
Semaphore semR;
semS.S = n;     //Initialize to max available number of resources/instances
semR.R = m;

Acquire () {
   if (semR.R == 0) {
   block(semR); //block this process in a queue
    }
   if(semS.S == 0){
   block(semS); //block this process in a queue
    }

   /*Wait until there is a resource available and
   then use that resource*/

   semR.R--;
   semS.S--;

   // return the ids of two available resource instances (note: this comment doesn't need to be
   // implemented)
}

Release () {
  // mark two released resource instances as available (note: this comment doesn't need to be
// implemented)
  if(semR.R <m ){
  wakeup(semR); //wake up a blocked process (if there is any);
  semR.R++;}

  if(semS.S <n ){
  wakeup(semS); //wake up a blocked process (if there is any);
  semS.S++;}

  /* If the semaphore values are equal to their max (m or n) then there are no resources to
release and the program should exit with no action*
 }

}
```

**2. (15 points)**

Consider two periodic processes, P1 and P2, whose period and processing time are (p1=50, t1=25) and (p2=75, t2=30), respectively. Note: As both P1 and P2 are periodic processes, each process will start only once within each of its period.

(1) Can these two processes be scheduled using the rate-monotonic scheduling (RMS)? Justify your answer using a Gantt chart (i.e., a timing diagram).

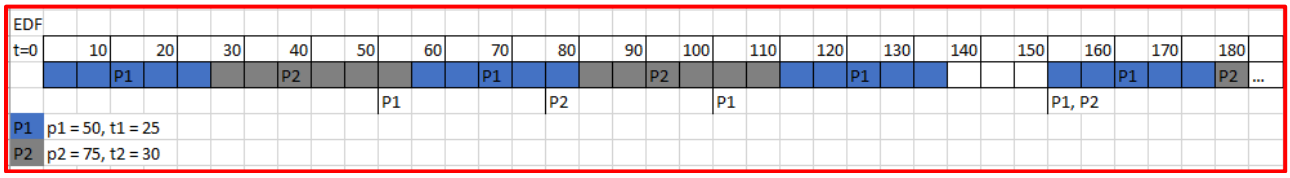



Yes, RMS can be used to schedule these processes. However, the RMS is not guaranteed to work based on:

$$\sum_{i=1}^{n} \frac{ti}{pi} \leq n(2^{1/n} - 1)$$

Where the sum of $t_i/p_{pi}$ = 0.9 which is greater than 0.83 and therefor does not satisfy the guarantee condition.

(2) Can these two processes be scheduled using the earliest-deadline-first (EDF) scheduling? Justify your answer using a Gantt chart (i.e., a timing diagram).




Yes, EDF can be used to schedule these processes.

**3. (15 points)**

Suppose that 4 processes, denoted by P1, P2, P3 and P4, are sharing the memory that has total 16 frames (numbered from 0 to 15), and currently some of their virtual pages are loaded into the frames as follows:

- Process P1's virtual pages 100, 101, 102, and 103 are loaded at frames 4, 5, 6, and 7, respectively.
- Process P2's virtual pages 200, 201, 202, and 203 are loaded at frames 0, 1, 2, and 3, respectively.
- Process P3's virtual pages 300, 301, 302, and 303 are loaded at frames 12, 13, 14, and 15, respectively.
- Process P4's virtual pages 400, 401, 402, and 403 are loaded at frames 8, 9, 10, and 11, respectively.

We also assume that an **inverted page table (IPT)** is used for address translation.

(a) Show the structure (i.e., entry numbers and column names) and the current content (i.e., current values in each entry) of the **inverted page table**.

| INVERTED PAGE TABLE | | | |
|---|---|---|---|
| Frame # | Page # | RWX | Dirty Bit |
| 0 | 200 | | |
| 1 | 201 | | |
| 2 | 202 | | |
| 3 | 203 | | |
| 4 | 100 | | |
| 5 | 101 | | |
| 6 | 102 | | |
| 7 | 103 | | |
| 8 | 400 | | |
| 9 | 401 | | |
| 10 | 402 | | |
| 11 | 403 | | |
| 12 | 300 | | |
| 13 | 301 | | |
| 14 | 302 | | |
| 15 | 303 | | |

(b) Show the structure and current content of the **hash table** created additionally, in order to speed up the searching of the above inverted page table. We assume the hash function used is: virtual_page_number *modulo* 10 (i.e., virtual_page_number % 10).

| HASH TABLE | |
|---|---|
| Frame modulo(10) | Frames |
| 0 | 0 --> 4 --> 8 --> 12 |
| 1 | 1 --> 5 --> 9 --> 13 |
| 2 | 2 --> 6 --> 10 --> 14 |
| 3 | 3 --> 7 --> 11 --> 15 |
| 4 | |
| 5 | |
| 6 | |
| 7 | |
| 8 | |
| 9 | |