CS 450 Summer 2013

Practice Final Exam

Problem 1. (- points):

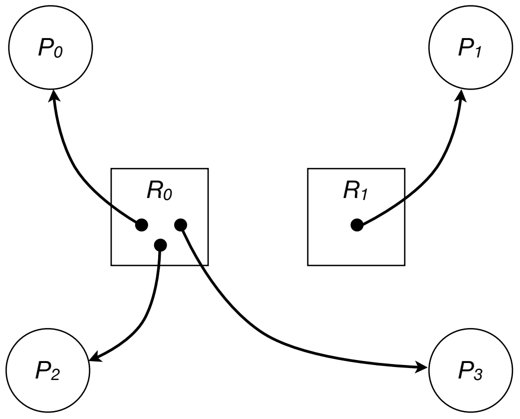
A. List and briefly describe the 4 necessary conditions for deadlock.

B. Sketch a resource allocation graph wherein the 4 necessary conditions are present, and the processes involved are deadlocked.

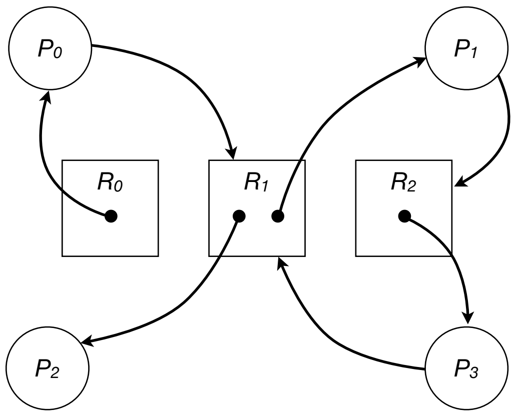
C. Sketch a resource allocation graph wherein the 4 necessary conditions are present, and there is **no** deadlock. (Each resource node may contain one or more instances.)

Problem 2. (- points):

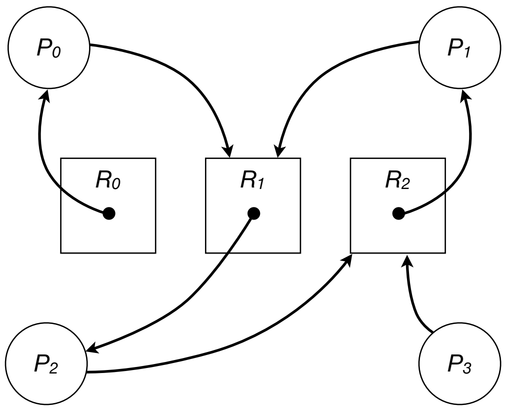
For each of the following resource allocation graphs, determine if the system is currently deadlocked. If it is **not**, add the *minimum number* of **edges** to the graph needed to create deadlock. (If the system is in deadlock to begin with, *don't modify the graph*!) Either way, list the deadlocked processes – i.e., only those processes that satisfy all 4 necessary conditions for deadlock.



Deadlocked processes: ____________________



Deadlocked processes: ____________________



Deadlocked processes: ____________________

Problem 3. (- points):

Consider the following system resource snapshot:

	Allocation			*Max*			*Available*		
	A	B	C	A	B	C	A	B	C
P_0	1	0	2	2	4	2	2	3	0
P_1	1	1	1	1	1	5			
P_2	0	1	1	3	4	4			
P_3	0	1	1	2	3	1			
P_4	1	2	0	2	2	5			

A. What are the contents of the matrix *Need*?

B. Give a process sequence that demonstrates the system is in a safe state.

C. If a request from process P_0 arrives for 1 instance of resource A, can the request be granted immediately? If not, why?

Problem 4. (- points):

Consider the following resource snapshot for a system currently in deadlock:

	Allocated				*Requested*				*Available*			
	A	B	C	D	A	B	C	D	A	B	C	D
P_0	3	0	0	0	0	2	0	0	1	2	0	2
P_1	2	5	0	0	0	0	0	3				
P_2	0	0	2	0	4	0	0	0				
P_3	4	0	3	2	0	4	0	0				
P_4	0	1	5	0	0	0	2	2				
P_5	0	2	0	6	5	0	0	0				

A. What are the processes and resources involved in the deadlock?

B. How would you recommend we get out of the current deadlock? Be specific, and justify your answer.

Problem 5. (- points):

Consider the following implementation of a reusable barrier. When put in use, `N` processes will be executing the barrier code in a loop – expecting to rendezvous at some point after the first line, and before entering the critical section.

For (global) initial values:

```
count = 0

mutex = Semaphore(1)

turnstile1 = Semaphore(0)

turnstile2 = Semaphore(0)
```

```
# rendezvous

mutex.wait()
  count += 1
  if count == N:
    turnstile1.signal()
mutex.signal()

turnstile1.wait()
turnstile1.signal()

# critical section

mutex.wait()
  count -= 1
  if count == 0:
    turnstile2.signal()
mutex.signal()

turnstile2.wait()
turnstile2.signal()
```

1. Upon testing this implementation, we find that it only works a limited number of times. In the long run, processes are no longer forced to rendezvous before proceeding through the turnstiles. Why is this happening?

2. Show how you would fix the problem – make additions/alterations as needed to the code.

Problem 6. (- points):

Due to a budget shortfall, it's become necessary for the computer science and electrical engineering departments to share conference rooms. Faculty from these two departments don't like to mix and the conference room is rather small, however, so the following rules apply:

- there cannot be CS and EE faculty in the conference room simultaneously
- there can never be more than 5 faculty members in the conference room simultaneously

Without synchronization, the body of each faculty thread would simply look like this:

```
while True:
    useConferenceRoom()
```

You are to add synchronization code to implement the rules above. Your solution should avoid both deadlock and starvation, and will consist of two separate thread bodies (one for each type of faculty) . Below are some variables you can use in your solution (note that the lightswitch construct is defined and its use illustrated on the last page of the exam):

```
empty       = Semaphore(1)
turnstile   = Semaphore(1)
CSswitch    = Lightswitch()
CSmultiplex = Semaphore(5)
EEswitch    = Lightswitch()
EEmultiplex = Semaphore(5)
```

Problem 7. (- points):

Consider the following table of block requests and arrival times:

Block #	Arrival time
6	0
15	5
30	8
10	18
45	21
5	23
60	25

The disk head is currently positioned over block #13. Seeking across N blocks requires $2 + N$ms; access times can be ignored. Complete the following table, showing the order in which block requests are serviced and the resulting seek times.

A. SSTF

From block #	To block #	Seek time
13	6	9ms
	Total:	

B. C-LOOK

From block #	To block #	Seek time
13	6	9ms
	Total:	

Problem 8. (4 points):

Consider the following parameters of an i-node based filesystem:

- Blocks are 1KB (2^{10} bytes) large
- Block pointers are 64-bits (8 bytes) wide
- Each i-node contains 32 direct pointers, 8 single indirect pointers, 4 double indirect pointers, and 1 triple indirect pointer.

What is the largest file size supported by this filesystem? (Show your work, preferably leaving your final answer in sums of powers of 2)

Problem 9. (4 points):

Consider the following parameters of a FAT based filesystem:

- Blocks are 8KB (2^{13} bytes) large
- FAT entries are 32 bits wide, of which 24 bits are used to store a block address

A. How large does the FAT structure need to be accommodate a 1GB (2^{30} bytes) disk?

B. What is the largest theoretical file size supported by the FAT structure from part (A)?

Lightswitch Pattern

```
# Class definition

class Lightswitch:
    def __init__(self):
       self.counter = 0
       self.mutex = Semaphore(1)

    def lock(self, semaphore):
       self.mutex.wait()
           self.counter += 1
           if self.counter == 1:
              semaphore.wait()
       self.mutex.signal()

    def unlock(self, semaphore):
       self.mutex.wait()
           self.counter -= 1
           if self.counter == 0:
              semaphore.signal()
       self.mutex.signal()



# e.g., Readers-Writers (no priority) solution:

## 0. Variables
readLightswitch = Lightswitch()
roomLock        = Semaphore(1)

## 1. Writers
roomLock.wait()
write()
roomLock.signal()

## 2. Readers
readLightswitch.lock(roomLock)
read()
readLightswitch.unlock(roomLock)
```

Feel free to detach this page!