

Problem 1: Definitions

Define each of the following terms:

- **Deadlock:** *when two or more threads are each waiting for each other to release a resource in a cyclical way such that none of them can make progress.*
- **Livelock:** *when a set of threads depend on each other to make progress, but are constantly changing state with regard to one another in an attempt to do so, and in the process not actually making any progress.*
- **Priority inversion:** *When a high priority thread A is blocked waiting for a resource that is held by a lower priority thread B, but in turn, thread B is not running because a thread C with priority greater than B's but lower than A's is scheduled to run. This can be seen as a special case of starvation.*
- **Working set:** *The set of memory (pages) that a program uses frequently.*

Problem 2: Deadlocks

a) Consider the following three threads executing different paths of execution. Determine if and where deadlock can occur for each thread. For example, “Thread 1 can deadlock on Line 2” would mean that Thread 1 can deadlock and wait forever to acquire mutex_e.

Thread 1	Thread 2	Thread 3
lock(mutex_d)	lock(mutex_e)	lock(mutex_f)
lock(mutex_e)	lock(mutex_f)	lock(mutex_d)
e=d*2+e	f=f+e	f=f*d
unlock(mutex_e)	unlock(mutex_f)	unlock(mutex_d)
unlock(mutex_d)	unlock(mutex_e)	unlock(mutex_f)

Complete the following, or write “deadlock cannot occur”

Thread 1 can deadlock on line Thread 1 can deadlock on line 2

Thread 2 can deadlock on line Thread 2 can deadlock on line 2

Thread 3 can deadlock on line Thread 3 can deadlock on line 2

b) Repeat part a) for the following two threads.

Thread 1	Thread 2
<pre>lock(mutex_h) lock(mutex_i) lock(mutex_j) j = i + h unlock(mutex_j) unlock(mutex_h) lock(mutex_g) g=g+i unlock(mutex_g) unlock(mutex_i)</pre>	<pre>lock(mutex_g) lock(mutex_i) i = g + g unlock(mutex_i) unlock(mutex_g)</pre>

Complete the following, or write “deadlock cannot occur”

Thread 1 can deadlock on line Thread 1 can deadlock on line 7

Thread 2 can deadlock on line Thread 2 can deadlock on line 2

c) Given the following processes and their resource allocations, use the banker’s algorithm to determine if a deadlock is inevitable; show your calculations. The system has 10 units of X and 15 units of Y.

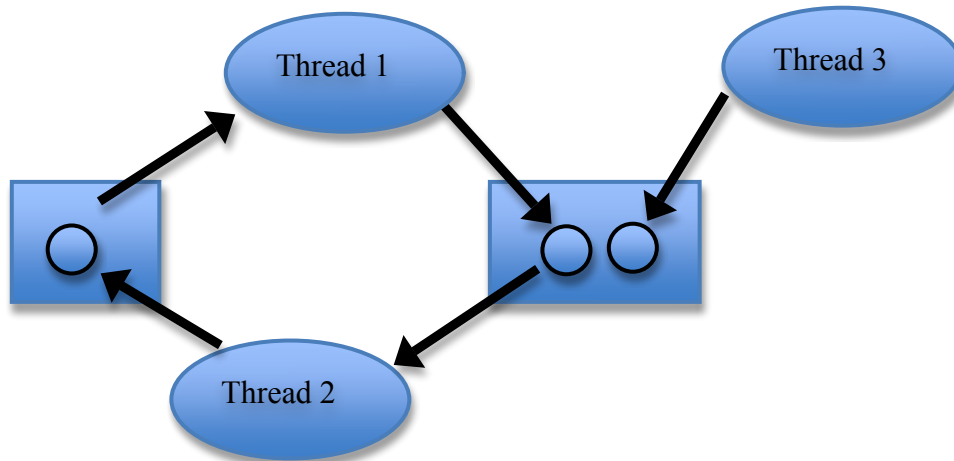
Process	has X	may need X	has Y	may need Y
1	3	10	2	4
2	0	6	6	7
3	5	5	2	6
4	1	2	5	5

Deadlock is not inevitable.

At the beginning, $avail_X = 10 - 9 = 1$, $avail_Y = 15 - 15 = 0$

There are enough available resources for process 4 to (get all that it may_need and) finish. Then process 3 can finish, then process 2, then process 1.

d) Give an example of a resource allocation graph that includes a cycle, but does not include a deadlock.



Problem 3: Address Translation

a) In a memory system that uses paging, what are the advantages and disadvantages of using larger pages over smaller pages?

The larger the page size, the more internal fragmentation (e.g. if the stack size is 8 pages, but the program actually only wanted 7.5 pages, half of a page is wasted). Also, larger page sizes means smaller page tables (which take less time to walk and use less memory). With larger pages, page faults may occur less often, improving performance, and paging to disk could be more efficient because longer sequential writes or reads will be happening.

b) Assume you have a system with 32-bit addresses, and suppose that you want to have 4 KB pages.

How many bits do you need to represent offsets within a page?

12 (4KB pages = 2^{12} addressable bytes means you need 12 bits to address them)

How many bits would you need to represent a virtual page number?

20 (i.e. the rest of the bits)

c) In the same system (with 4 KB pages and 32-bit addresses), suppose that you have a TLB of size 3 entries containing the following mappings:

Virtual page number	Physical page number	Valid bit
0x31415	0x01337	1
0xCAFE5	0xF000D	1
0xDECAF	0xCOFEE	0

For each of the following virtual addresses accessed by the program, determine whether there would be a TLB hit or miss, and for the hits, write the physical address that the virtual address translate to.

Virtual address	Hit or miss?	Physical address
0x31415926	<i>Hit</i>	<i>0x01337926</i>
0xCAFE1335	<i>Miss</i>	<i>N/A</i>
0xDECAF123	<i>Miss</i>	<i>N/A</i>
0x31415555	<i>Hit</i>	<i>0x01337555</i>

Problem 4: Short Answers

a) A hardware test-and-set instruction provides a simple way to achieve mutual exclusion using only user-level code. Why might a kernel-level solution still be preferable?

Using hardware test-and-set at the user-level would require busy waiting. A kernel-level solution which uses blocking could be more efficient.

b) Without hardware support for test-and-set, could you still implement critical sections? How would you do it?

Yes. See, e.g., Peterson's Algorithm in the textbook, which uses two variables: a Boolean, and an array of n Booleans where $n = \text{num_threads}$.

c) What is the worst possible CPU scheduling algorithm, i.e. the one that yields the worst possible average response time? Assume that the algorithm must run a thread if there is one available, that a context switch has zero overhead, and that the length of each job is known when it is submitted.

Longest job first.