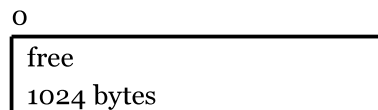


# Homework 7

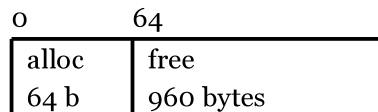
*Due: 1 May 2009*

## Problem 7.1

In this problem, you'll simulate a memory allocation algorithm, keeping track of a free chunk of memory and subdividing it as allocation requests come in. You'll start with an empty 1024-byte piece of memory (the right side of the diagram is omitted here for size, but you should draw the whole allocation):



As requests come in, you'll subdivide it, labelling each piece according to size and allocation status; for instance, after a 64-byte request, you draw:



Hand-simulate the following sequence of allocations (and deallocations), first according to the first-fit algorithm, then according to best-fit, and finally according to worst-fit.

```

a = malloc (64);
b = malloc (64);
free (a);
c = malloc (16);
d = malloc (256);
e = malloc (512);
free (d);
f = malloc (64);
g = malloc (128);
h = malloc (32);
free (f);
i = malloc (112);
free (g);
j = malloc (240);

```

**Problem 7.2**

Consider the code we wrote in class on Monday (essentially same as on p127 of the MOS book), and the modifications we made to it Wednesday (to be more like p130).

- a. Write out a series of instructions that would cause the non-semaphore version to break; there should be two columns of instructions, one for the producer and one for the consumer, and since we're modelling a single-processor system, there should be only one instruction per line (so both columns aren't executing at once). For instance, the diagram might begin like this:

<b>Producer</b>	<b>Consumer</b>
<code>produce_item()</code>	
store result in <code>item</code>	
get <code>count</code>	
check if it's == N (it isn't)	check if <code>count</code> == 0 (it is)
	<code>sleep()</code>
<code>insert_item(item)</code>	

Note that you can choose how coarse- or fine-grained you write the instructions: in the left column, the fetch and the compare of `count` are written separately, while in the consumer, they're written together. You can choose the granularity to highlight the problem with the code, and you also get to choose when the context switches fall, also to highlight the problem. Add any annotations (e.g. variable values) you need to clarify why the problem occurs. You can use abbreviations as long as they're clear.

Once you've written out the table, indicate how the semaphore solution from Wednesday prevents that particular situation.

- b. Then, separately, draw another diagram of instructions that, first, includes one producer and *two* consumers, and second, highlights a *different* problem in the Monday/p127 code that is also solved by the semaphore solution. Indicate how semaphores solve this one, too.