Take-home exam - DO YOUR OWN WORK! (Due July 16)

1. The following C function is passed two shared arrays and a pointer to a shared integer for the result. Assuming that the threads have already been created, give pthreads code to make it execute efficiently in parallel using contiguous allocation of loop indices. The results produced by the parallel version must be the same as the original sequential version. 20 points

```
minloc(doubles,ints,result)
double doubles[100000];
int ints[100000],*result;
{
int i,k;

k=0;
for (i=1; i<100000; i++)
   if (doubles[i] < doubles[k])
     k=i;
   else if (doubles[i] == doubles[k] && ints[i] < ints[k])
     k=i;
(*result)=k;
}
```

2. Adapt your solution to problem 1 to work with arrays `doubles` and `ints` allocated contiguously across several processes under MPI. The number of processes will divide evenly into `100000`. At termination of `minloc()`, all processes will need to know the global subscript `k`. DO NOT RELOCATE THE ENTIRE ARRAY TO ANY PROCESS! 25 points

3. Suppose you have approximately $131,072 to spend on a cluster of PCs to run a variety of existing applications written in C using MPI. Your job is to prepare a budget for the cluster, along with a list of design decisions encountered, a list of WWW URLs accessed (e.g. vendors and installed clusters), and a critique of your design (i.e. features and deficiencies). You may suggest alternative configurations. 40 points

Constraints on your design:

a. All hardware must be commercially available. Used, salvage, discontinued, stolen, homebuilt cards, etc. are not allowed.
b. You do not need to budget for service contracts, installation, support, etc.
c. Expect to spend about 20% of your budget for interprocessor communication.

4. The following task graph shows dependencies among sections of code to be included in a single function that will be coded as a single concurrent function. Each section of code will be modified by being executed across the participating threads. Thus, if there is a path from node x to node y, there must be a barrier in the code to assure that all threads have completed the processing for node x before the processing for node y commences. Show how the code should be ordered to minimize the number of barriers. Be sure to indicate the positions of the barriers. 15 points