Name: _____

Directions: **Work only on this sheet** (on both sides, if needed); do not turn in any supplementary sheets of paper. There is actually plenty of room for your answers, as long as you organize yourself BEFORE starting writing. In order to get full credit, **SHOW YOUR WORK**.

1. (25) Write a single line of code which could be added to the main **while** loop in **DSMSrvr.pl**, which would print to the screen the names and values of all the shared variables. Note that different elements of the same array are considered different variables.

Solution:

```
print %SharedVars
```

2. (25) The following function calculates the minimum of a variable number of arguments. Fill in the blanks.

```
sub min {
   $m = _____________
   for $x (____________) {
      if ($x < $m) {$m = $x};
   }
   return $m
}
```

Solution:

```
sub min {
   $m = shift;
   for $x (@_) {
      if ($x < $m) {$m = $x};
   }
   return $m
}
```

3. (25) Add code to **pth.py** which will do profiling, i.e. keep track of how many times each of the action **'pause'**, **'wait'**, **'set'**, **'set_all'**, **'set_but_stay'** and **'quit'** are executed, over all threads. You will have class variables **prf** and **profiling**. The former is a dictionary with keys equal to the action names and values equal to the counts, while the latter is a boolean, with True meaning that profiling is on. Add two contiguous lines for data definition/initialization, and at most three contiguous lines for the maintaining of the counts. In both cases, state in between which two consecutive lines you will insert your code.

Solution:

```
# near the beginning of thrd:
   prof = {'pause':0, 'wait':0,'set':0, 'set_but_stay':0, \
           'set_all':0, 'quit':0}
   profiling = False
...
# after setting yv1
         if thrd.profiling:
            thrd.prof[yv1] += 1
```

4. (25) Use Perl's **tie()** function to create read-only scalars, which also keep track of how many times such a scalar is read, in a variable named **accesscount**.

In **readonlyscalar.pm**, each blank may be filled in by between zero and four lines (right-semicolons) of code. In **testro.pl**, fill in blanks within a line by an expression, and fill in the fully blank line with zero or one line of code.

readonlyscalar.pm:

```
package readonlyscalar;
-----------------------------------
sub TIESCALAR {
}
sub FETCH {
   -----------------------------------
}
sub STORE {
   -----------------------------------
}
1;
```

testro.pl:

```
tie $x,'ReadOnlyScalar','$x',8;
$x = 168;  # prints out an error message
$y = $x;
print $x,"\n";  # prints 8
______________________;  # just one ";"
print _____________, "accesses so far\n";  # prints 2
```

Solution:

```
sub TIESCALAR {
   my $class = @_[0];  # we'll create a class to be named after
                       # our tied variable
   my $r = {Name=>@_[1],Value=>@_[2],AccessCount=>0};
   bless $r, $class;
   return $r;
}

sub FETCH {
   my $r = shift;
   $r->{AccessCount}++;
   return $r->{Value};
}
sub STORE {
   print "write attempt for ",$r->{Name},"\n";
}
...
tie $x,'ReadOnlyScalar','$x',8;

$x = 168;  # prints out an error message
print $xobj->{AccessCount},"\n";  # prints 0
$y = $x;
print $x,"\n";  # prints 8
$xobj = tied $x;
print $xobj->{AccessCount},"\n";  # prints 2
```