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. (15) In class we discussed a way to make Python threads essentially non-preemptible. The key element of this would involve calling which function?

2. (20) Write a Perl module **Mean.pm** which will serve as a class to be used with **tie**. Its main use is to store an integer-valued variable, but on request, which is signaled by a nominal assignment of 0, the mean of all values assigned to the variable so far will be printed out. (The 0 itself is not actually assigned, i.e. the value of the variable will not become 0.) For example:

```
# this is TestMean.pl
use Mean;
...
$x = 1;
$y;
tie $y,'Mean';
$y = 6;
$y = $y + 2;
$y = $x;
$y = 0;  # prints out 5
$y = 10;
print $y, "\n";  # prints 10
$y = 0;  # prints out 6.25
...
```

Write the entire code for the module **Mean.pm**. **BE LEGIBLE.**

3. Consider the SimPy program **MachRep3.py**.

(a) (10) Suppose we were to add a class variable **TotWaitToCallRepair** to the class **Machine**, which would represent the total time that all machines have waited for a call to the repairperson. If for instance machine 1 goes down while machine 0 is up, machine 1 must wait for machine 0 to fail before the repairperson is called. We are interested in the total of all such wait times, and will print that total out when the simulation is finished. Add exactly two (2) lines, no more, to the code in **Machine.Run()** which will be used to compute that total. **TotWaitToCallRepair** is "given," and thus a line "declaring" it would not count, but "declarations" for any other variables would count. Note that your answer consists both of the two lines to be added AND specification of WHERE in the existing code the two new lines should be inserted.

(b) (10) (Do not assume anything from part (a).) Here we want to find the total amount of time during which both machines are up simultaneously. Add just three (3) lines (where for instance **if a: x = 0** counts as one line) to accomplish this, showing both the lines to be added and where to add them. Note that any "declaration" lines count.

(c) (15) (Do not continue to assume anything from parts (a) and (b).) Consider a variation in which we have **Machine.NMach** machines, and the repairperson is called whenever at least **Machine.MinCall** machines are down. We'll have a variable **Machine.WaitingCall**, which will be a list of IDs of the machines currently down, waiting for the repairperson to be called (waiting due to there being fewer than **Machine.MinCall** machines down). Rewrite the large **if-elif** section in **Machine.Run()** (starting with the **if** and ending just before **yield request...**) to reflect these changes. Do not make use of **RepairPerson.n**. Be Pythonic. **BE LEGIBLE.**

4. This question compares PerlDSM to the material we learned about Python threads.

(a) (10) Fill in the blank with a term from our course: In PerlDSM's server code, the variable **@LockQueue** is used to avoid the problem we termed _____ in our unit on Python threads.

(b) (10) One of the PerlDSM examples, **Primes.pl**, used basically the same algorithm as the prime-finding example in our unit on Python threads. Note for example that the variable **$NextI** in **Primes.pl** corresponds to the variable **nexti** in our Python example. Both variables are guarded with locks. (Note: Your answer will not depend on whether the PerlDSM or Python version is run.) Suppose we had forgotten to put in the "unlock" operation, e.g. **nextilock.release()** in the Python case. Then which of the following would occur? (i) The program may produce wrong answers. (ii) The program will produce correct answers, but possibly with some wasted effort. (iii) The program will produce correct answers, with no wasted effort. (iv) The program will hang, i.e. never finish.

Solutions:

1. sys.setcheckinterval()

2.

```
# Mean.pm

package Mean;

sub TIESCALAR {

   my $class = @_[0];
   my $r = {value=>0, n=>0, sum=>0};
   bless $r, $class;
   return $r;
```

```
}

sub FETCH {
   my $r = shift;
   return $r->{value};
}

sub STORE {
   my $r = shift;
   my $newval = shift;
   if ($newval != 0) {
      $r->{value} = $newval;
      $r->{n}++;
      $r->{sum} += $newval;
   }
   else  {
      print $r->{sum}/$r->{n}, "\n";
   }
}

1;
```

3.a.

```
if Machine.NUp == 1:
   StartWait = now()
   yield passivate,self
   Machine.TotWaitToCallRepair += now() - StartWait
```

3.b.

```
(Total2Up, Start2Up) = (0.0,0.0)
...
      if Machine.NUp == 1:
         Machine.Total2Up += now() - Machine.Start2Up
         yield passivate,self
...
      Machine.NUp += 1
      if Machine.NUp == 2: Machine.Start2Up = now()
      yield release,self,RepairPerson
```

3.c.

```
if Machine.NMach - Machine.NUp < Machine.MinCall:
   Machine.WaitingCall.append(self.ID)
   yield passivate,self
else:
   for Mch in Machine.WaitingCall:
      reactivate(M[Mch])
   Machine.WaitingCall = []
```

4.a. busy wait

4.b. (iv)