Massachusetts Institute of Technology
Department of Electrical Engineering and Computer Science
6.826 Principles of Computer Systems

SOLUTIONS TO PROBLEM SET 4

Problem 1: Bounding Mean Execution Time (20 points)

A requirement of this problem is that if N tasks begin without completing, the clock cannot subsequently tick indefinitely. Some solutions failed to accomplish this.

```
CONST N : Int := 10

TYPE TaskId = Int

VAR now : Time
    startTimes : TaskId -> Time
    endTimes : TaskId -> Time
    endedTasks : SET TaskId
    runningTasks : SET TaskId
    nextTask : TaskId := 0

APROC NewTaskId() = << nextTask + := 1; RET nextTask >>

FUNC MeanTime() = VAR t |
                  endedTasks * (\ e | t + := endTimes(e) - startTimes(e));
                  runningTasks * (\ e | t + := now - startTimes(e));
                  RET t / (endedTasks.size + runningTasks.size);

THREAD Clock() = DO endedTasks.size < N \/ MeanTime() < T => now + := 1
                 [] SKIP OD

PROC TimedP() = VAR id : TaskId
  << id := NewTaskId(); runningTasks := runningTasks + {id};
     startTimes(id) := now >>;
  P();
  << endTimes(id) := now; runningTasks := runningTasks - {id};
     endedTasks := endedTasks + {id}; RET >>
```

Problem 2: Reliable FIFO Buffer (15 points)

a) The problem was somewhat ambiguous about the definition of "buffer throughput": that term could credibly bound either the runtime of a single (say `put`) operation, or it could bound the rate at which data moves through the buffer. Either definition was all right (and they're closely related as long as the buffer is bounded). In the solutions, I use the former definition, because it is easier to work with.

Another issue to deal with is to choose an appropriate time window for which to compute throughput. For instance, a throughput bound of 250MB/s might be enforced by keeping track of the number of operations over the past second (leading to 250MB buffers), or it might be enforced by keeping track of the number of operations over the past millisecond (leading to 250KB buffers). As long as a reasonable time window was chosen, it was fine. To shrink the window down to a single operation (effectively turning the upper bound on throughput into a lower bound on latency), however, was not reasonable. In this solution, we keep track of operations over the past millisecond.

Our `Clock` ticks every nanosecond (a 1GHz clock), so that a time bound of 10ms corresponds to 10^7 clock ticks.

```
VAR now : Time,
    deadlines : SET Time,
    counts : SEQ Int := {}

FUNC msToTick(m : Int) -> Time = RET m * 1000000

THREAD Clock() = DO now < deadlines.min =>
                 now + := 1;
                 IF now // 1000000 = 0 => counts = (0 + counts).sub(0, 1000) FI
                 [] SKIP OD

MODULE BoundedBuffer[M] EXPORT put, get =
    TYPE Q = SEQ M

    VAR q := {}

    PROC put(m) = VAR t : Time
    << now < t /\ t < now + msToTick(10) /\ ~ t in deadlines =>
       deadlines := deadlines + {t} >> ;
    << q := q + {m} >>;
    << deadlines := deadlines - {t}; counts(0) + := 4;
       (+ : counts <= 250000000) => SKIP; % fails if too much throughput
       RET >>

    PROC get() -> M =
    << now < t /\ t < now + msToTick(10) /\ ~ t in deadlines =>
       deadlines := deadlines + {t} >>;
    << VAR m | m = q.head => q := q.tail >>;
    << deadlines := deadlines - {t}; RET m>>
END Buffer
```

b) The abstraction function simply throws away the irrelevant state.

```
   AF Buffer.q = BoundedBuffer.q
```

It's clear that our implementations have traces that are a subset of the `Buffer` specification.

Problem 3: Widgets (15 points)

(a) The processor speed permits a transaction rate of

$$\frac{1.2 \times 10^9 \text{ instr.}}{1 \text{ sec.}} \cdot \frac{1 \text{ trans.}}{2 \times 10^5 \text{ instr.}} = 6000 \text{ trans./sec.}$$

A set of D balanced disks permits a transaction rate of

$$D \cdot \left(\frac{6 \times 10^{-3} \text{ sec.}}{1 \text{ trans.}} + \frac{4 \times 10^3 \text{ bytes}}{1 \text{ trans.}} \cdot \frac{1 \text{ sec.}}{5 \times 10^7 \text{ bytes}} \right)^{-1} = \frac{D}{6.08 \times 10^{-3}} \text{ trans./sec.}$$

Equating the two gives $D = 36.48 \approx 36$ disks.

(b) This time, the disks permit a transaction rate of

$$D \cdot \left(\frac{6 \times 10^{-3} \text{ sec.}}{10 \text{ trans.}} + \frac{4 \times 10^3 \text{ bytes}}{1 \text{ trans.}} \cdot \frac{1 \text{ sec.}}{5 \times 10^7 \text{ bytes}} \right)^{-1} = \frac{D}{6.8 \times 10^{-4}} \text{ trans./sec.}$$

Equating the disk transaction rate with the processor transaction rate gives $D = 4.08 \approx 4$ disks.

(c) The modified web server can handle $\frac{1}{6.8 \times 10^{-4}} \approx 1471$ transactions per second.

Problem 4: Resource Allocation (20 points)

Average running time after upgrading the disk is:

$$\frac{7 + 3(1 - y)}{10} = 1 - 0.3y$$

while average running time after upgrading the CPU is:

$$\frac{5(1 - x) + 5}{10} = 1 - 0.5x$$

(a) Upgrading the CPU is better if $x > \frac{3}{5}y$; upgrading the disk is better if $x < \frac{3}{5}y$; and they are equally bad for $x = \frac{3}{5}y$.

(b) Note that upgrading the CPU gives a mean time between 0.5 and 1, while upgrading the disk gives a mean time between 0.7 and 1. If we upgrade the CPU to give a mean time of 0.7, i.e. $0.7 = 1 - 0.5x \Rightarrow \frac{3}{5} = x$, then no disk upgrade (value of y) gives better performance. There is no value of y that ensures that upgrading the CPU can never be better than upgrading the disk.