Homework 1

Due Date: October 15, 2008 **Points:** 100

Questions

1. (*10 points*) Why was timesharing not widespread on second-generation computers? (*text*, problem 1.6)

2. (*15 points*) Some early computers protected their operating systems against destruction by placing it in memory locations from which all programs (including the operating system itself) could read, but into which no program could write. What problem does this protection scheme create?

3. (*25 points*) The classical batch processing system completely ignores the cost of increased waiting time for users. Consider a single batch characterized by the following parameters:

   - M average mounting time
   - T average service time per job
   - N number of jobs
   - S unit price of service time
   - W unit price of waiting time per user

   Show that the optimal batch size minimizing the cost of service time and waiting time per user within a single batch is $N_{opt} = \sqrt{\frac{MS}{TW}}$.

4. (*25 points*) Does the solution shown in Fig. 2-20 solve the Dining Philosophers Problem? If yes, demonstrate that neither deadlock nor starvation can occur; if no, give an example of deadlock or starvation occurring (and say which one it is).

5. (*25 points*) (The Model Builders' Problem) Consider a system with three model airplane building processes and one agent process. Each building process requires a tube of glue, a piece of newspaper, and a model kit to put a model airplane together. One of the processes has tubes of glue, another pieces of newspaper, and the third model kits. The agent has an infinite supply of all three. The agent places two of the ingredients for the model builders on the table. The process who has the remaining ingredient can then build one model airplane. It signals the agent upon completion. The agent then puts out another two of the three ingredients and the cycle repeats. Write a program to synchronize the agent and the model building processes using monitors.

Extra Credit

1. (*30 points*) Here is another proposed solution to the Dining Philosophers Problem. Does it actually solve the problem? If yes, demonstrate that neither deadlock nor starvation can occur; if no, give an example of deadlock or starvation occurring (and say which one it is). The proposed solution is on the next page.

```
 1  #define N       5       /* number of philosophers */
 2  typedef int semaphore;  /* semaphores are special kind of int */
 3  semaphore fsem[N] = { 1, 1, 1, 1, 1 };  /* one semaphore per fork */
 4
 5  void philosopher(int i)
 6  {
 7          while (1) {
 8                  think();
 9                  /* acquire the forks */
10                  if (i % 2 == 1) {
11                          down(fsem[i]);
12                          down(fsem[(i + 1) % N]);
13                  }
14                  else {
15                          down(fsem[(i + 1) % N]);
16                          down(fsem[i]);
17                  }
18                  eat();
19                  /* release forks */
20                  up(fsem[i]);
21                  up(fsem[(i + 1) % N]);
22          }
23  }
24
25  void main(void)
26  {
27          int i;   /* counter in a for loop */
28
29          for(i = 09; i < N; i++)
30                  philosopher(i);
31  }
```