Range 95 to 23
Median and mean – 63

89-95 A+
76-86 A
66-73 A-
63-65 B+
58-62 B
47-57 C+
35-45 C’s and C-'s
Below 35 – “in big trouble”

Concurrent programming
Problem (#2) -- 15 and above--OK

1) (concurrency and synchronization)
The problem with the solution to the right is:
(a) Race conditions
(b) Deadlock
(c) Starvation
(d) Potential for priority inversions
(e) Missed wakeup.
(f) None of the above.

/*turn initially = DUM*/
void Tweedledee()
{
    while (turn == DUM)
        currentThread->Yield();
    while(1) {
        mutex->Acquire();
        x = Quarrel(x);
        turn=DUM;
        cond->Signal(&mutex);
        cond->Wait(&mutex);
        mutex->Release();
    }
}

void Tweedledum()
{
    while(1) {
        mutex->Acquire();
        turn=DEE;
        cond->Wait(&mutex);
        x = Quarrel(x);
        cond->Signal(&mutex);
        mutex->Release();
    }
Repeat for the solution below:

(a) Race conditions
(b) Deadlock
(c) Starvation
(d) Potential for priority inversions
(e) Missed wakeup.
(f) None of the above.

/* turn initially = DEE;
semaphores dee and dum initially 0 */

void Tweedledee()
{
    void(1) {
        while(1) {
            mutex->Acquire();
            x = Quarrel(x);
            turn=DUM;
            dum->V();
            while(turn==DUM)
                dum->P();
            mutex->Release();
        }
    }
}

void Tweedledum()
{
    while(1) {
        mutex->Acquire();
        while(turn==DEE)
            dum->P();
        x = Quarrel(x);
        turn=DEE;
        dum->V();
        while(turn==DEE)
            dum->P();
        mutex->Release();
    }
}

Lock mutex:
Condition prof, student,
int maxStudents = 0;

while(TRUE) {
    mutex.Acquire();
    profBusy = false;
    IdleProf:
    if(maxStudents == 0)
        prof.Wait(mutex);
    else
        if(profBusy = true)
            profSignal(mutex);
        else
            maxStudents++;
            student.Wait(mutex);
            student.Signal(mutex);
    mutex.Release();
}

ArrivingStudent() {
    mutex.Acquire();
    if(profBusy && maxStudents == 0)
        profSignal(mutex);
    mutex.Release();
}

b) Implement IdleProf and ArrivingStudent using semaphores. You may assume that semaphores are fair (e.g., FIFO).

Semaphore student = 0; // student "resources"
Semaphore prof = 0; // professor "resources"

while(TRUE) {
    IdleProf();
    prof.V(); // 1 prof avail.
    student.V(); // "consume" 1 student
}

ArrivingStudent() {
    student.V(); // 1 student avail.
    prof.P(); // "consume" 1 prof
}
a. \[ \frac{T}{T+S} \]

b. \[ \frac{T}{(TS)/Q + T} = \frac{Q}{S+Q} \]

c. approaches 0

d. 3 5 6 9 Shortest Job First

X goes in between depending

• Break circular waiting – lock in order of acct numbers
  – Use banker’s algorithm – max claims issue
• Break no preemption – lock first, if can’t get second roll-back (unlock first), try again
  – Starvation issue
• Break hold-and-wait – lock both at once
  – Starvation issue
  – Global mutex lock on entire database – performance bottleneck