**Assignment 2**

**Operating Systems**

1. Consider the following program:

**boolean** blocked [2];

**int** turn;

**void** P (int id)

{

**while** (true) {

blocked[id] = true;

**while** (turn != id) {

**while** (blocked[1-id])

/\* do nothing \*/;

turn = id;

}

/\* critical section \*/

blocked[id] = false;

/\* remainder \*/

}

}

**void** main()

{

blocked[0] = false;

blocked[1] = false;

turn = 0;

**parbegin** (P(0), P(1));

}

The above is a proposed solution to the mutual exclusion problem for two processes. Find a counterexample that demonstrates that this solution is incorrect.

1. *N* processes share *M* resource units that can be reserved and released only one at a time. The maximum need of each process does not exceed *M*, and the sum of all maximum needs is less than *M* + *N*. Show that a deadlock cannot occur. [Hint: a deadlock can occur only if all resources are occupied and all the processes involved in deadlock have pending requests. What does this imply?)
2. Suppose that there are two types of philosophers. One type always picks up his left fork first (a “lefty”), and the other type always picks up his right fork first (a “righty”). Prove the following:
3. Any seating arrangement of lefties and righties with at least one of each avoids deadlock (hint: prove by contradiction; assume there is a deadlock, then what can you say about a philosopher’s neighbour)
4. Any seating arrangement of lefties and righties with at least one of each prevents starvation
5. Consider the following program

**const int** n = 50;

**int** tally;

**void** total()

{

**int** count;

**for** (count = 1; count <= n; count++)

tally++;

}

**void** main()

{

tally = 0;

**parbegin** (total (), total ()**);**

write (tally);

}

Determine the proper lower bound and upper bound on the final value of the shared variable *tally* output by this concurrent program. Assume processes can execute at any relative speed and that a value can only be incremented after it has been loaded into a register by a separate machine instruction.

1. Consider a paging system with a page table stored in memory.

**a.** If a memory reference takes 200 nanoseconds, how long does a paged reference take (reference is retrieved from page table in memory and then program data is retrieved from memory) Assume all pages are in memory?

**b.** If we add associative registers (TLB), and 75 percent of all page-table references

are found in the associative registers, what is the effective memory reference time on average? (Assume that finding a page table entry in the associative registers takes 10 nanoseconds, if the entry is there.) Assume all pages are in memory.

**c.** What is the effective memory reference time if the hit ratio for the associative registers is 70%, the hit ratio for memory is 20%, and 10% of the time we must go out to the disk to load the faulted page in memory, where context switching, disk transfer time, etc takes 100 m seconds.).

1. Consider a code containing loops of the following form:

for(i=0; i<n; i++)

for(j=0; j<n; j++)

a[j][i] += b[j][i];

The inner addition loop is observed to be trashing the TLB much more than it should. Reason this out and give a fix. What do you expect to happen to the paging performance?

1. A FIFO page replacement algorithm will replace a page that was first referenced the longest time ago. Under a workload with locality of reference, choosing to replace a page accessed a long time ago seems like a good idea since it is not likely to be within the current locality being accessed. Does this mean that a FIFO replacement algorithm

can approximate a LRU algo+rithm for workloads with strong locality? Justify.