University of Maryland
CMSC 412, Spring 2024
Professor Pete Keleher
May 6, 2024

Name: ____

DirectoryID:

Mid-Term Exam 1

⊕ ***Show your work or you may not be considered for partial credit.***

⊕ *We will not grade the back of the sheets.*

⊕ ***Avoid asking questions*** *- If something is unclear, document your assumptions and move on.*

⊕ *Do not forget to write your name on the first page. Initial each subsequent page.*

⊕ *Be* **neat** *and* **precise**. *We will not grade answers we cannot read.*

⊕ *If you have written something incorrect along with the correct answer, you should* **not** *expect to get all the points. I will grade based upon what you* **wrote**, *not what you* **meant**.

⊕ *You should draw simple figures if you think it will make your answers clearer.*

☐ *means "choose many"*

◯ *means "choose one"*

1. (20 pts) Short Answer

   (a) (4 pts) Explain the difference between stable and unstable queuing systems, and why this is important.
   answer: unstable means offered load is more than capacity, so queues grow exponentially

   (b) (3 pts) Discuss the problem of encapsulation with `trylock` primitives.
   answer: trylock requires you to back out of acquired locks if you fail on a later one. Encapsulation means that the prior acquired locks might be in function calls and hard to back out of, or might just be missed.

   (c) (3 pts) Discuss the usual approach to preventing the circular-wait aspects of deadlocks.
   answer: acquire locks in well-known order respected by all parties

(d) (4 pts) Explain how the GeekOS (after project 2) handles signal interrupts that have registered user-level handlers. Be specific.
answer: trampolines, etc.

(e) (3 pts) Explain how safe states differ from unsafe states in the context of deadlocks. Be explicit.

answer: In an unsafe state, converting only a single claim edge into a request edge makes deadlock

(f) (3 pts) Explain the difference between Hoare and Mesa semantics.
answer: Mesa semantics means scheduling is not tied to anything happening with synchronization, whereas Hoare semantics mean that calling `signal()` means that the scheduler immediately stops the current thread and switches to the signaled thread.

2. (10 pts) The following wait-free code adds an item to the head of a linked list. Find at least two different types of major errors that could occur, and explain how each can come about.

```
void insert(int value) {
    node_t *n = malloc(sizeof(node_t));
    n->value = value;
    do {
        n->next   = head;   // L1
        head      = n;      // L2
    } while (head != n);   // L3
}
```

answer: Could lose a mod if T1 executes L1, losts CPU, T1 comes through and does the whole thing. Then T1 finishes and T2's edit lost.
T1 could execute L1 and L2, then T2 executes everything, and then T1 resumes. T1's edit could be duplicated.

3. Queuing (20 pts)

   Consider a queuing system with three types of jobs, all of which repeat at 10 msec intervals:

   - type A, takes 6 msec, starts at time 0
   - type B, takes 3 msec, starts at time 1 msec
   - type C, takes 1 msec, starts at time 2 msec

   (a) (1 pts) What is the overall throughput?
   answer: 300 / second.

   (b) (1 pts) What is the overall utilization?
   answer: $(5 + 3 + 1)/10 = 1.0 \Longrightarrow 100\%$

   (c) (3 pts) What is the average number of jobs (both in queue and running) in the system, if we assume an overall average turnaround time of 6 msec?
   answer:
   From Little's Law: $N = \lambda \times R$, so
   $N = 300 * 6 \times 10^{-3} = 1.8$ jobs in the system.

   (d) Consider a **FIFO** queuing discipline.

   i. (2 pts) Show the running job for each of the first 10 msec.
   For example, "ABCABABAAA" would be round robin. Use '_' for no ready job.
   answer: AAAAAABBBC

   ii. (2 pts) What are the turnaround times of job types A, B, and C?
   answer: Turnaround times are 6, 8, and 8.

   iii. (3 pts) How many jobs are in the system, on average?
   answer: $N = \lambda \times R$, so $N = 300 * \frac{22}{10} \times 10^{-3} = 2.2$ jobs.

(e) Consider a preemptive shortest-job-first (**SJFP**) queuing system with the same jobs as above.

i. (2 pts) Show the running job for each of the first 10 msec.
answer: ABCBBAAAAA

ii. (3 pts) What are the turnaround times of job types A, B, and C?
answer: Turnaround times are 10, 4, and 1 $\Longrightarrow R = 5$msec.

iii. (3 pts) How many jobs are in the system, on average?
answer: $N = \lambda \times R$, so $N = 300 * 5 \times 10^{-3} = 1.5$ jobs.

4. (15 pts) Write a semaphore version of reader-writer locks synchronization, such that:

   - One or more readers OR a single writer can be in the critical section.
   - Do not spinwait, i.e. each time through a loop should have a semaphore wait.
   - Free of starvation in the following sense:
     - If a reader is leaving the critical section and there is a writer waiting, no more readers can enter the critical section until at least one writer runs.
     - Conversely, if a writer is leaving the critical section and there is a reader waiting, no more writers will enter the critical section at least one reader runs.
   - Mesa semantics

   Write initialization code, and read and write variants of both "lock" and "unlock". Call and initialize your semaphores as:

```
sem_t    aLock = sem_init(N);  // where N is an integer
sem_wait(&aLock);
sem_post(&aLock);
```

answer:

```
sem_t    readLock; = sem_init(1);  // initially 1
sem_t    writeLock = sem_init(1);  // initially 1
sem_t    mutex; = sem_init(1);  // initially 1

int      readers = 0;
int      writers = 0;
bool     readerTurn = true;


void rwlockLockRead() {
    sem_wait(&readLock);

    if (++readers == 1)
        sem_wait(&writeLock);

    while (!readerTurn && (writers > 0))
        sem_wait(&writeLock);

    sem_post(&readLock);

    // Do the read

    sem_wait(&readLock);

    readerTurn = false;
    if (--readers == 0)
        sem_post(&writeLock);
    }
    sem_post(&readLock);
}

void rwlockLockWrite() {
    mutexIncr(&writers, 1);
    do {
        sem_wait(&writeLock);
    } while (readerTurn && (readers > 0));

    // Do the write

    mutexIncr(&writers, -1);
    readerTurn = true;
    sem_post(&writeLock);


void mutexIncr(int *val, int howMuch) {
    sem_wait(&mutex);
    *val += howMuch;
    sem_post(&mutex);
}
```

5. (10 pts) Assume the extern function `TestAndSet()` below is compiled to a single atomic TestAndSet x86 instruction. Write a lock-free producer-consumer solution where the buffer holds at most a single value, and that value must be a positive integer. Add any other global variables you think you need.

```
// returns true on match
extern bool TestAndSet(int *intValue, int expected, int newValue);

int    buffer;

int func consume() {
  int val;
  do {
    val = buffer;
  } while ((val == 0) || !TestAndSet(&buffer, val, 0));
  return val;
}

func produce(int new) {
  int empty;
  do {
    ;
  } while (!TestAndSet(&buffer, 0, new));
}
```

6. (10 pts) Paging and memory systems. Assume:

- a two-level page table
- 4K byte pages
- A 1GB address space.
- PTEs and PDEs are 4 bytes each
- Used address ranges:
  - 2MB of code at location 0
  - 6 MB heap allocated at location 128MB
  - 50k stack allocated starting at location 1GB

(a) (3 pts) How many valid PDEs does the directory have?
answer:
$2^{12}/2^2 = 2^{10}$ PTEs / page of the page table, so a PDE maps $2^{10} \times 2^{12} = 2^{22} = 4$MB.
Therefore need 1 for code, two for heap, 1 for stack = 4

(b) (3 pts) How many pages does the directory use?
answer: $\frac{2^{30}}{2^{22}} = 2^8$PDEs $*$ 2^3bytes/PDE $= 2^{12}$ bytes needed. This requires only **1 page**.

(c) (3 pts) How many total physical pages does the multi-level page table use?
answer: 4 + 1 = **5**

(d) (1 pts) Would a 3-level page table consume less physical memory? Why or why not?
answer: no, only num pages matters, directory has to have at least one, and need four distinct instantiated page table pages.

7. (15 pts) Paging and memory systems.

   Assume three page frames and the following page reference stream for this question:
   1 2 3 4 1 2 5 1 2 3 4 5

   For each policy show page frame contents after each reference, marking hit with an H, and count up the total number of hits. For example:

```
              H H     H
5 5 5 3 4 4 1 2 3 4 1 2
  1 2 3 4 3 4 4 1 1 2 2       3 hits
    1 2 3 4 1 2 5 3 3
```

   (a) LRU

```
              H H
1 2 3 4 1 2 5 1 2 3 4 5
  1 2 3 4 1 2 5 1 2 3 4   2 hits
    1 2 3 4 1 2 5 1 2 3
```

   (b) MIN

```
        H H   H H     H
1 2 3 4 4 4 5 5 5 5 5 5
  1 2 2 2 2 2 2 2 2 2 2    5 hits
    1 1 1 1 1 1 1 3 4 4
```

(repeating the reference stream: 1 2 3 4 1 2 5 1 2 3 4 5)

(c) FIFO

```
                 H  H     H
1  2  3  4  1  2  5  5  5  3  4  4
   1  2  3  4  1  2  2  2  5  3  3      3 hits
      1  2  3  4  1  1  1  2  5  5
```

(d) Repeat FIFO with 4 page frames.

```
            H  H
1  2  3  4  4  4  5  1  2  3  4  5
   1  2  3  3  3  4  5  1  2  3  4      2 hits
      1  2  2  2  3  4  5  1  2  3
         1  1  1  2  3  4  5  1  2
```

(e) Anything interesting about the two FIFO cases? Explain.
answer: Fewer hits than with 3 page frames. This is Belady's anomaly, FIFO is not a "stack" algorithm.