CS744 (Autumn 2017)

Midsem Review: Solutions

1. (a) Semaphores variables:

```
pt_waiting = 0
treatment_done = 0
doc_avlbl = 1
```

   (b) Patient process:

```
down(doc_avlbl)
consultDoctor()
up(pt_waiting)
down(treatment_done)
noteTreatment()
up(doc_avlbl)
```

   (c) Doctor:

```
while(1) {
down(pt_waiting)
treatPatient()
up(treatment_done)
}
```

2. (a) Producer:

```
int produced = produceNext();
shptr->field1=produced;
shptr->field2 = 1; //indicating ready
while(shptr->field2 == 1); //do nothing
```

   (b) Consumer:

```
while(shptr->field2 == 0); //do nothing
consumed=shptr->field1;
consumeNext(consumed);
shptr->field2 = 0; //indicating done
```

3. **Using conditional variables.** A boolean variable `writer_present`, and two conditional variables, `reader_can_enter` and `writer_can_enter`, are used.

   Read lock:

```
lock(mutex)
while(writer_present)
  wait(reader_can_enter)
read_count++
unlock(mutex)
```

   Read unlock:

```
lock(mutex)
read_count--
if(read_count==0)
  signal(writer_can_enter)
unlock(mutex)
```

   Write lock:

```
lock(mutex)
while(read_count > 0 || writer_present)
  wait(writer_can_enter)
writer_present = true
unlock(mutex)
```

   Write unlock:

```
lock(mutex)
writer_present = false
signal(writer_can_enter)
signal(reader_can_enter)
unlock(mutex)
```

Using semaphores.

```
sem lock = 1; sem writer_can_enter = 1; int readCount = 0;

readLock:
down(lock)
readCount++
if(readCount ==1)
  down(writer_can_enter) //don't coexist with a writer
up(lock)

readUnlock:
down(lock)
readCount--
if(readCount == 0)
  up(writer_can_enter)
up(lock)

writeLock:
down(writer_can_enter)

writeUnlock:
up(writer_can_enter)
```

In the above code, the first reader entering the critical section will block future writers from entering. Further, if the first reader blocks before entering the critical section (because a writer is present), then future readers will even fail to acquire the lock required for readers, so they will not enter either. A leaving writer unblocks the first reader. Once the first blocked reader is unblocked, it releases the lock which will enable the other readers to enter the critical section.

4. The accounts must be locked in order of their account numbers. Otherwise, a transfer from account X to Y and a parallel transfer from Y to X may acquire locks on X and Y in different orders and end up in a deadlock.

```
struct account *lower = (from->accountnum < to->accountnum)?from:to;
struct account *higher = (from->accountnum < to->accountnum)?to:from;
dolock(&(lower->lock));
dolock(&(higher->lock));

from->balance -= amount;
to->balance += amount;

unlock(&(lower->lock));
unlock(&(higher->lock));
```

5. D

6. (a) 5. The value is only changed in the parent.
   (b) Yes, the file is only closed in the parent.

7. When $M < N$ and the workload to the server is CPU-bound.

8. Each PTE has frame number (21 bits) and flags (10 bits) $\approx$ 4 bytes. The total number of pages per process is 2^{20}, so total size of inner page table pages is $2^{20} \times 4$ = 4MB. With heirarchical paging, we require one outer page table per process, because one page is sufficient to hold all the PTE of inner page tables. So the total size of page tables of one process is 4MB + 4KB. For 1K process, the total memory consumed by page tables is 4GB + 4MB.

9. Size of each cache set = $4 * 64 = 2^8$ bytes. Number of cache sets = $2^{22}/2^8 = 2^{14}$. So, 6 bits form the offset, 14 bits form the index, and the remaining 32 - 6 - 14 = 12 bits form the tag.

10. $(2 + 6 + Z)/6 = 10$, so $Z = 52$.