190158

1.

a. Minimum = 1, as all the pids() are odd the number of processes does not change in each iteration.

Maximum = 2^5 = 32, as if pid is even for all processes that are created then number of processes increases by a factor of 2 on each iteration.

b. Minimum = 6, as the increment is one on each iteration when all the pids are odd.

maximum =112

2.

3.

4.

(a) Max virtual address space for each process = 2^53.

(b level 1: 2^43 bytes

level 2: 2^33 bytes

level 3 : 2^23 bytes Depending on the number of bits left after each level.

(c) 1 PB = 2^50 bytes = 2^50 bytes / 8kb PNFs = 2^37 PNFs

=> 37 bits for PNF , 15 bits for access reservation

Remaining = 64 – 37- 15 = 12 bits

(d) 0 bytes free memory increase as none of the unmapped addresses may be mapped . => Min = 0

Maximum - Number of entries in Level-4-2^32/8KB = 2^19

Maximum number of level-4 pages to be freed = 2^19/2^10 = 2^9

Maximum number of level-1,2,3 pages to be freed = 1 each

⇨ Max. = 8GB + 515*8Kb

5. T1 is executing first and both threads call lock function.

As flag[0] = 0, it breaks out of while loop => acquires lock.

Now, as T0 starts executing=> the 2nd condition evaluates to false, because turn = 0 and id^1 = 1.

Hence, it also breaks out of while loop as, both threads are out of lock function simultaneously, they are violating mutual exclusion.

6. there is a problem with this implementation that if the context switch happens after mutex_unlock(Q2->M)then any thread can manipulate Q2.

Therefore correct one will be as follows….

```
void AtomicMove (Queue *Q1, Queue *Q2) {
        mutex_lock(Q1->M);
        Item E = Q1->Dequeue();
        mutex_unlock(Q1->M);
        mutex_lock(Q2->M);
        Q2->Enqueue(E);
        mutex_unlock(Q2->M);
}
```

7.) a. (8 + 2*1024 + 8* 1024 * 1024) * (4096/256) here 256 byte per entry

b. Minimum= (5 + 1), We find the directory entry in the first block of the directory at each level , one more for file data access

Maximum = 5*(4*1024*1024+4+4*1024+4+8) +1

At each level of directory, the entry is found in the last block

c. minimum = 1MB + 4KB

maximum = 1MB + 16KB