1-4, everything in class/notes

disable/enable timer / smart interrupt

two privileged instructions — can have CPU forever / does it interrupt all CPU's

```
disable();
enable();
```

```
P1:
while(1){
    disable();
    x++;
    enable();
}
```

```
P2:
while(1){
    disable();
    x--;
    enable();
}
```

```
test-and-set(bool &lock){
    bool initial = lock;
    lock = true;
    return initial;
}
```

```
P1:
.....                    lock = false;
while(1){
    .....
    while(test-and-set(lock));   // busy waiting checking  { entry section }
    x++;                         { critical section }
    lock = false;                { exit section }
```

..... { remainder section }

}

P2:

.....

while(1){

.....

while (test-and-set(lock)); ← busy waiting checking { entry section }

x--; { critical section }

lock = false; { exit section }

..... { remainder section }

}

(1) mutually exclusive

(2) deadlock free

(3) starvation free

(4) low coupling/dependence

Can't assume P1 or P2 is faster

software solution pass 3 / fail 1

try 1:

int turn = 1;

Cobegin P1 || P2 Coend ← run in parallel

P1: P2:

....

```
while(1){                                   while(1){
  while(turn==2);        entry                while(turn==1);
    X++;                 Critical               X++;
    turn=2;              exit                   turn=1;
  }.....                 remainder            }.....
}                                           }
```

try2:

```
int c1=0, c2=0;
```

```
P1:                                         P2:
  .....                                       .....
  while(1){                                   while(1){
a   c1=1;                  entry            c   c2=1;
b   while(c2);                              d   while(c1);
      X++;                 Critical               X--;
      c1=0;                exit                   c2=0;
    .....                                       .....
  }                                           }
```

a→c→b→d Deadlock

Correct/peterson? solution

```
int c1=0, c2=0, will_wait;
```

```
P1:                                         P2:
  .....                                       .....
  while(1){                                   while(1){
1   c1=1;                                       c2=1;
```

will_wait = 1;	entry	will_wait = 2;
while(c2 && will_wait == 1);		while(c1 && will_wait == 2);
x++;	critical	x--;
c1 = 0;	exit	c2 = 0;
.....		.....
}		}

Semaphore

integer (4)

queue

wait: P (-3)

signal: S