1. Concept checkers

(1) What is the principal motivation for introducing the process model?

(2) What is the main difference between a thread and a process?

(3) What major advantage do operating systems have over user-level applications when implementing threads?

(4) Since threads share a process' address space, each thread can modify others' stack and state. Should there be protections between threads? Should there be protections between processes?

(5) Threads in a process use the same physical registers like EAX, EBX, …. Will modification of these registers by one thread ruin the results stored there by another thread? How are the limited physical registers properly shared by unlimited number of threads?

(6) Which of the following items are shared by all threads and which are private to each thread?
①address space; ②global variables; ③stack; ④thread state;
⑤program counter; ⑥registers; ⑦open files.

(7) What states can a process/thread be in? Describe all possible transitions between the states

2. The following are two code fragments in a linked list implementation:

Insert a link:	Remove a link:

tmp=new node(val);
successor=current->next;
current->next=tmp;
tmp->next=successor;
	current=predecessor->next;
	predecessor->next=current->next;
	delete current;

(1) Give an execution sequence where the above program will fail when multiprogrammed.
(2) Does the following version fixes the synchronization problem?

Insert a link:	Remove a link:
cur->next=new node(val, cur->next)	pred->next=pred->next->next;

3. Below are iterations of a fish feeding demonstration program. The goal is to ensure that the goldfish is fed once and only once? Does each version prevent fish from starvation and overeaten (as you know, goldfish eats whatever amount you feed them)? If not, give a counter-example execution sequence where the fish starves/overeats. Finally, implement a properly synchronized version of the program using locks.

(1)

Peter:	Tracy:

if (noFeed) {
  feed fish
}
	if (nofeed) {
	  feed fish
	}

(2)

Peter:	Tracy:

if (noNote) {
  leave note
  if (noFeed) {feed fish}
}
	if (noNote) {
	  leave note
	  if (noFeed) {feed fish}
	}

(3)

Peter:	Tracy:

leave notePeter
if (no noteTracy) {
  if (noFeed) {feed fish}
}
remove notePeter
	leave noteTracy
	if (no notePeter) {
	  if (noFeed) {feed fish}
	}
	remove noteTracy

(4)

Peter:	Tracy:

leave notePeter
while (no noteTracy) { do nothing }
if (noFeed) {feed fish}
remove notePeter
	leave noteTracy
	if (no notePeter) {
	  if (noFeed) {feed fish}
	}
	remove noteTracy

4. Below is an implementation of consumer-producer problem using sleep & wakeup primitives. Is it ensured that the inventory is properly maintained and the two threads are deadlock free (i. e. threads cannot be both blocked)?

```
int N=100;     /*inventory capacity*/
int count=0;   /*# of goods in stock*/
```

```
void producer(){
   while(true) {
      int item=produce_item();
      if (count==N) sleep();
      insert_item(item);
      count=count+1;
      if (count==1) wakeup(consumer);}}
```

```
void consumer(){
   while(true) {
      if (count==0) sleep();
      int item=remove_item();
      count=count-1;
      if (count==N-1) wakeup(producer);
      consume_item(item);}}
```

5. The following program is a sketch for the famous readers and writers problem, which models access to a database. In this model, multiple threads are allowed to read the database simultaneously, but only one thread is allowed to write the database at a time; besides, other threads will not be even allowed to read the database if one process is writing it. Modify the program to meet the synchronization requirements.

```
void reader(){




      read_database();




      use_data_read();


}
```

```
void writer(){


      think_up_data();




      write_database();


}
```

(1) Use a lock to ensure that no two threads can access the database simultaneously (even between readers).
(2) Introduce an integer to count the number of threads reading the database. Increase the counter by one when a reader is about to read the database and decrease it by one when the reader finishes.
(3) Modify the reader() function such that new readers can access the database as long as some threads are already reading it.
(4) Use another lock to prevent simultaneous access to the counter and to ensure atomicity of the operation of increasing/decreasing the counter and acquiring/releasing the lock of the database.
(5) Assume your reader-writer program is applied to a news website with great publicity, where thousands of audiences are browsing posts at any second and editors add or modify the posts once in an hour. What's the problem with the current implementation in this case? Can you fix it?

6. There is a single-log bridge over the river whose width allows only one man to cross. If an eastward travelling person and a westward travelling person cross the bridge at the same time, they will end up getting stuck in the middle (we assume no one is willing to jump to the water and swim across the river). However, multiple persons travelling in the same direction could cross the bridge simultaneously by one following another. The following program is a sketch that simulates scenario. Use locks to ensure that no one get suck infinitely and people travelling in both direction have a chance to cross the bridge even if there is a steady stream in the other direction.

void travel_east(){
 	
cross_bridge();
 	
}
	void travel_west(){
	
	cross_bridge();
	
	}

7. Assume LSRC has a single bathroom on the first floor which is not gender-segregated. When a woman is in a bathroom, other women may enter, but no men, and vice versa. A sign with a sliding marker on the door of the bathroom indicates which of three possible states it is currently in: Empty, Women present, and Men present. Complete the following procedures with proper synchronization to ensure the order of the bathroom.

void woman_wants_to_enter(){
 	
state=Women_present;
 	
}
	void man_wants_to_enter(){
	
	state=Men_present;
	
	}

void woman_leaves(){
 	
state=Empty;
 	
}
	void man_leaves(){
	
	state=Empty;
	
	}