1. What are the differences between the short-term, medium-term, and long-term scheduler? (10)
2. What are the differences between the remote procedure calls and sockets? (5)
3. How can we measure the length of the next CPU burst? (5)
4. The major problem with priority scheduling algorithms is indefinite blocking. How can you solve the problem? (5)
5. Why the modern operation systems adopt the multilevel feedback queue scheduling algorithm? (5)
6. In thread scheduling, what are the differences between the process-contention scope and system-contention scope, and how can we set these contention scope policies in Pthread? (10)
7. What are the differences between the spinlock and wait-signal semaphore? Why we need the monitors in the modern operation systems? (10)
8. How can you design a mechanism to solve the deadlock problem in the modern operation systems? (10)
9. How can you design a memory management mechanism in a computer architecture, which supports segmentation with paging? (10)
10. Demand paging can significantly affect the performance of a computer system. How can we measure the effective access time for a demand-paged memory, and how can you improve it? (10)
11. How can the Windows NT and UNIX manage the allocated space in a disk? (10)
12. Please describes that how can the following program work? (10)

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>

pthread_mutex_t count_mutex     = PTHREAD_MUTEX_INITIALIZER;
pthread_cond_t  condition_var   = PTHREAD_COND_INITIALIZER;

void *functionCount1();
void *functionCount2();
int count = 0;
#define COUNT_DONE  10
#define COUNT_HALT1  3
#define COUNT_HALT2  6

main()
{
   pthread_t thread1, thread2;

   pthread_create( &thread1, NULL, &functionCount1, NULL);
   pthread_create( &thread2, NULL, &functionCount2, NULL);

   pthread_join( thread1, NULL);
   pthread_join( thread2, NULL);

   printf("Final count: %d\n",count);

   exit(0);
}
```

```c
void *functionCount1()
{
   for(;;)
   {
      // Lock mutex and then wait for signal to relase mutex
      pthread_mutex_lock( &count_mutex );

      // Wait while functionCount2() operates on count
      // mutex unlocked if condition varialbe in functionCount2() signaled.
      pthread_cond_wait( &condition_var, &count_mutex );
      count++;
      printf("Counter value functionCount1: %d\n",count);

      pthread_mutex_unlock( &count_mutex );

      if(count >= COUNT_DONE) return(NULL);
    }
}
```

```c
void *functionCount2()
{
    for(;;)
    {
       pthread_mutex_lock( &count_mutex );

       if( count < COUNT_HALT1 || count > COUNT_HALT2 )
       {
          // Condition of if statement has been met.
          // Signal to free waiting thread by freeing the mutex.
          // Note: functionCount1() is now permitted to modify "count".
          pthread_cond_signal( &condition_var );
       }
       else
       {
          count++;
          printf("Counter value functionCount2: %d\n",count);
       }

       pthread_mutex_unlock( &count_mutex );

       if(count >= COUNT_DONE) return(NULL);
    }

}
```