**Q (2):** Operating System is?

A: Provides an environment for application

B: A set of software frameworks that provide programs to run addition services to application developers such as databases, multimedia, graphics...

C: Defines the ways in which the system

D: E and A resources are used to solve the computing problems of the users

E: Is a software that manages the computer hardware

**Q (3):** We have stressed the need for an operating system to make efficient use of the computing hardware. When is it appropriate for the operating system to forsake this principle and to waste resources?

A: All programs that are not associated with the operating system might waste CPU cycles

B: Single-user systems like mobiles maximize use of the system for the user. A GUI might waste CPU cycles, but it optimizes the user's interaction with the system

C: Allocate separate resources of the computer as needed to solve the problem given might waste CPU cycles

D: Main memory allocation and management might waste CPU cycles

Q (12): which of the following is considered as activity of an operating system with regard to secondary storage management?

**A: Disk scheduling**

**B: Decide which processes are to be loaded into memory when memory space becomes available**

**C: the creation and deletion of both user and system processes**

**D: the suspension and resumption of processes**

Q (16): How dynamically linked libraries (in Windows, DLLs) are loaded into memory

**A: Linker combines these object files into single binary executable file**

**B: Each operating system provides its //// unique system calls for that purpose**

**C: Loaded by Application Binary Interface (ABI)**

**D: Loaded once as needed, shared by all use the same version of that same libraries**

Q (18): Source code complied into object files designed to be loaded into any physical memory location, **what** is the role of **linker**?

**Q (23):** When a process creates a new process using the fork () operation, the state which be shared between the parent process and the child process?

A: stack memory segments

B: shared memory segments

C: heap memory segments

D: data memory segments

E: A&C&D

Q (26): Consider the code of the question Q (25), Explain what happened for the code of the question of Line A?

**A: Do nothing because of the NULL in parameters**

**B: After fork both child and parent will start executing simultaneously**

**C: Lists the contents of current folder**

**D: The execution of the child process will be overlapped over time**

Q (27): Consider the code of the question Q (25), after compiling and executing the program we got the following output (partially):

**This line is from parent, value = 4861**

**This line is from child, value = 4926**

**This line is from parent, value = 4862**

**This line is from child, value = 4927**

**This line is from parent, value = 4863**

**This line is from child, value = 4928**

**This line is from parent, value = 4864**

**This line is from child, value = 4929**

**The cause of the scrambled data is context switching. How to solve problem?**

**A: Add wait() system call to the Parent Process() function**

**B: Add wait system call to the Child Process() function**

**C: Use threads instead of process**

**D: Change MAX COUNT to 50**

**E: Use Garbage Collection**

Q (34): What are **Real-Time** Systems?

**A: Those systems in which the correctness of the system depends only on the logical result are computation**

**B: Those systems in which the correctness of the system depends only on the time at which the result are produced**

**C: Those systems in which the correctness of the system depends only on the logical result of computation, regardless of the time at which the results are produced**

**D: Those systems in which the correctness of the system depends not only on the logical result of computation, but also on the time at which the results are produced**

Q (36): In Real-Time System, if Tasks' character are **event** driven. What we called those tasks?

**A: Priority Tasks**

**C: Periodic Tasks**

**B: Sporadic Tasks**

**D: Aperiodic Tasks**

Q (38): Consider Task set: T; = (ci, pi= di) as T1 = (5,10), T2 = (7,15) and T3 = (3,20).
Using **RMS**, knowing that tasks arrive at t=0. **What** task will be executed at **t=10**?

**A: T3 for 3 msec**

**B: TI for 5 msec**

**C: T1 for 1 msec**

**D: T2 for 7 msec**

Q (39): Consider previous question; **what** task will be executed at t=10? using **EDF** algorithm?

**A: T1 for 5 msec**

**B: T1 for 2 msec**

**C: T3 for 3 msec**

**D: T2 for 2 msec**

Q (40): To guard against the race condition, we need to ensure that?

**A: Only one process at a time can**

**B: The code has no access to shared data manipulate the shared data**

**C: The use of optimal scheduling algorithm**

**D: The code has no global or static variables**

Q (41): What is Critical Section?

**A: Is a section of code where database**

**B: Is a section of code where the user en records are read**

**critical info. Like username and password**

**C: Is a section of code where shared data may be manipulated and a possible race condition may occur**

**D: Is a section of code where /////// memory allocation is used condition may occur**

Q (44): Many systems provide **hardware** support for implementing the critical section code. In case of uniprocessor: **What** can be done?

A: Disable Atomic Instructions

B: Disable Context Switching

C: Disable Kernel Mode

D: Disable Interrupts

Q (45): **Memory Barriers** are hardware synchronization solutions; what is Memory Barrier?

A: Where a memory modification of one processor may not be immediately visible to all other processors

B: Where a limit is forced on the usage of memory

C: An instruction that forces any change in memory to be propagated (made visible) to all other processors

D: A memory model of Uniprocessor register set

Q (48):

Consider the following C Program:

```
#include <pthread.h>
 #include...
 Long long sum= 0;
 #define NUM_LOOP 50000
Void *runner (void *param); /* the thread */
 Void main (){
pthread_t tid1;
 int offset1 =5;
 pthread_create(&tid1,NULL, runner,&offset1);
 pthread_t tid2;
 int offset2 =-5;
 pthread_create(&tid2 , NULL, runner, &offset2);
 /* now wait for the thread to exit */
 pthread_join(tid1,NULL);
 pthread_join(tid2,NULL);
printf("sum = %lld\n", sum); // *** Line 1 ***
}
void *runner(void *param)
{
int i;
 int offset = *(int *) param;
 for (i = 0; i < NUM_LOOP; i++)
sum += offset;
 pthread_exit(0);
}
```

What is the critical section of previous code?

A: for (i = 0; i < NUM_LOOP; i++)
sum += offset;

B: void *runner (void *param)
{All instructions}

C: The main () function

D: for (i = 0; i < NUM_LOOP; i++)
sum += offset;

and Line 1

Q (51): What is the role of sem_wait(), and sem_post()?

A: Each thread that wishes to use a shared resource performs a sem_post.
and sem_wait : the thread release the resource
B. Each thread that wishes to shared resource performs a sem_wait().
and sem_post: the thread release the resource
C: Both used to stop the context switching
D: Mutex synchronization tool.

Q (52): OS designers build software tools to solve critical section problem the most advanced is **Monitor**. How Monitor provides synchronization?

A: By monitoring the status of shared data
B: Is an abstract data type, in which internal variables can only be accessible by code within the procedure

C: Only one process may be active within the Monitor at a time ✓

D: B and C

Q (54): Another form of deadlock is Starvation. **What** is **Starvation**?

A: Two or more processes are waiting indefinitely for an event that can be caused by only one of the waiting processes

B: Resources needed by existed

C: A process may never be removed from the semaphore queue in which it is suspended ✓

D: Scheduling problem when high ////// process holds a lock needed priority process

Q (61): middleware is a set of software framework that provide addition services to application developers such as databases, multimedia, graphics...

A: True

B: False

Q (69): machine-language instructions are atomic; that is can be interrupt

A: True

B: False