Question I. Introduction to operating systems.

1. Explain the difference between hard real-time scheduling and soft real-time scheduling.

- *Real-time scheduling imposes a maximal response time to start or to complete a real-time process.*
- *With hard real-time, the maximal response time must always be guaranteed or the real time processing is simply cancelled because of security reasons.*
- *With soft real-time, the maximal response time should be respected as much as possible, but whatever the real-time processing can still be executed. The only consequence of a non-respect of a maximal response time is that the results of the real-time processing could be inaccurate.*
- *In conclusion, we can say that hard real-time scheduling is more rigorous than soft real-time scheduling for the respect of the maximal response time.*

2. Can an operating system implement the two operational modes (user vs system) if the hardware does not support a mode bit? Explain.

The operating system could use a software mode bit to implement the dual-operation modes, but that would be very inefficient compared to a hardware mode bit.

3. Give an example of an operating system that does not implement the dual-mode protection.

The DOS operating system initially ran on an 8086 CPU that did not have a hardware mode bit. DOS did not have good protection mechanisms and a faulty user program could easily bring the whole system to a halt.

4. Explain the main difference that exists between multiprogramming and multitasking.

Multiprogramming refers to the ability to load multiple programs at the same time in memory for later execution. Multitasking also requires multiple programs in memory, but in addition it allows the programs to run concurrently.

5. Among symmetrical and asymmetrical multiprocessing, which one provides a better performance in general.

- *With symmetrical multiprocessing, the OS kernel can execute on any CPU, but each CPU is independent and has its waiting queue. Sometimes, the CPUs can communicate with each other when needed.*
- *With asymmetrical multiprocessing, the OS kernel runs only on a dedicated CPU that handles the dispatching of user programs on the other CPUs.*
- *Symmetrical multiprocessing offers better overall performances because a process can be instantly scheduled on any available CPU, whereas with asymmetrical*

multiprocessing a CPU may have to wait to be assigned work by the master CPU. Also, the master CPU may be underutilized because it only runs the operating system.

- *A problem with symmetrical multiprocessing is that the workload of the CPUs can become unbalanced, whereas with asymmetrical multiprocessing the master CPU has the possibility to dispatch an equitable amount of work on each CPU.*

6. Explain an advantage that a microkernel has over a general kernel structure.

A microkernel is a kernel that some nonessential components have been moved to user space, and the core components (e.g. basic process and memory management, communication, and protection) that remain run in system space. The microkernel occupies less memory space, and the user space components run more efficiently than in system space and can be easily extended without affecting the microkernel.

Question II. Process management.

1. When a process moves to the running state, the register context of that process must be restored on the CPU before it begins execution.

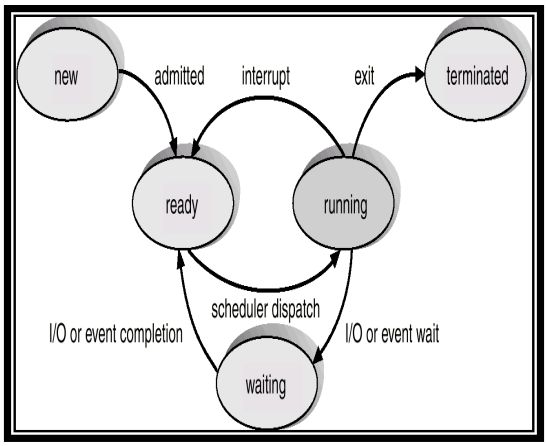
(a) Why is it necessary to restore the register context for the process?

The register context contains the CPU information that is required to execute the process. If the register context is not restored than the process will execute with an inconsistent CPU state that may cause it to crash.

(b) Why are all registers restored before the program-counter (i.e. PC) register?

When the program counter register is restored, execution control is immediately transferred to the user process. Thus, the dispatcher stops executing and there is no way to restore the other registers.

2. The following process state diagram applies to a pre-emptive multitasking (e.g. Windows, Linux, MaxOS X) operating system.



(a) Modify the above process state diagram so that it applies to a cooperative multitasking (MacOS 9) operating system.

We simply need to replace the edge labeled interrupt by a new edge labeled yield because a cooperative tasks relinquishes the CPU only when it wants.

(b) Modify the above process state diagram so that it applies to a multiprogrammed batch operating system.

We should remove the interrupt edge because a job can not be interrupted by a timer event. Therefore, a job can not move directly from the running state to the ready state. A job leaves the running state only when it terminates or when it starts an I/O operation.

3. The combined threads approach provides the benefits of both user and kernel threads. Explain what benefits that this approach inherits from (a) user threads and (b) kernel threads.

(a)In the combined threads approach, the threads are mostly managed in user space which provides a better performance level.

(b) In the combined threads approach, the threads are dispatched mostly in the kernel which provides a better concurrency level.

Question III. Process synchronization and communication.

1. A TestAndSet instruction is a machine instruction that is used in synchronization and that guarantees the atomicity of a group of statements. A programmer needing such synchronization mechanism must use a high-level instruction in his program.

- Would that high-level instruction be implemented as part of the compiler (e.g. instruction) or as part of the kernel (e.g. system call)?

A TestAndSet instruction is not privileged and could either be implemented in user or system mode. The TestAndSet instruction is used mostly to ensure atomicity at the entry section of a synchronization code, and is therefore mostly used in a monitor (compiler) or in a semaphore (kernel). Therefore, the programmer will indirectly use the TestAndSet instruction by calling the appropriate monitor or semaphore routines.

2. Dijkstra has proposed multiple software solutions to the critical section problem. However, these solutions seem to be incorrect according to Dijkstra.

- Explain why the following proposed solution is effectively incorrect. Specify which critical section requirements are violated.

(a)
```
int turn = 1;
proc(int i)
{  while (TRUE)
   {   compute;
       while (turn != i);
       <critical section>
       turn = (i+1) mod 2;
   }
}
```

The bounded waiting requirement is not satisfied because a process that doesn't need the critical section when its turn comes will block the other process. This is due to the fact that the access to the critical section uses strict alternations.
The progress requirement is also not satisfied because a process is selected to enter in its critical section even if it did not request permission.

3. Explain which critical section requirement that the following synchronization code violates.

```
…
wait(mutex);
x = x + 1;
cout << x << endl;
signal(mutex);
…
```

The mutual exclusion requirement is violated because a process executes in its critical section code that is not related to that critical section.