Tutorial-4: Real-Time Computer Systems and Architecture

A 1. **Pre-emptive scheduling** is based on timer interrupts, where a running thread may be interrupted by the OS and switched to the ready state at will (usually if something more important comes through) or when it has exceeded its timing allocation.

Non-preemptive scheduling means once a thread is in the running state, it continues until it completes, or gives up the CPU voluntarily.

Non-preemptive threads are likely to monopolies the CPU until it given up, thus all threads must work co-operably to not hold the CPU unnecessarily so that other threads can have the cpu time needed. Another issue is such kind of scheduling cannot ensure the response time of each thread. But this type of scheduling has minimum contact switching and does not face deadlock problem.

A 2. **Deadlock** is the phenomenon that arises when a number of concurrent processes all become blocked and cannot proceed without a resource that another holds, and all cannot release any resources that it is currently holding.

Starvation is the phenomenon which arises when a process is never given a resource that it requires to proceed (includes CPU time), even if it repeatedly becomes available, as it is always allocated to another waiting process (higher priority).

Processes halt in deadlock because they cannot proceed and the resources are never made available. Therefore, no progress can proceed. Where starvation, progress is made overall at the expense of a particular process or processes, which consistently miss out on being allocated their requested resource

A 3. a) EDF = earliest deadline first (EDF). Earliest deadline first schedules tasks by those whose deadline is the earliest. The following tasks are reorder according to their decreasing priorities.

Task	Processing Time (ms)	Period (ms)	Deadline (ms)
T4	10	30	30
T2	7	40	40
T3	10	60	60
T1	25	150	150

b)

$$U = \sum_{i=1}^{4} u_i = \frac{10}{30} + \frac{7}{40} + \frac{10}{60} + \frac{25}{150} = 0.333 + 0.175 + 0.1667 + 0.1667 = 0.842$$

The sufficiency condition is given by $U \leq 1$

Therefore, $0.84 \leq 1$

Condition is satisfied, thus according to EDF it is schedulable.

A 4. A superscalar processor is one in which multiple independent instruction pipelines are used. Each pipeline consists of multiple stages, so that each pipeline can handle multiple instructions at a time. Multiple pipelines introduce a new level of parallelism, enabling multiple streams of instructions to be processed at a time.

A 5. Superpipelining exploits the fact that many pipeline stages perform tasks that require less than half a clock cycle. Thus, a doubled internal clock speed allows the performance of two tasks in one external clock cycle

A 6. **True data dependency:** A second instruction needs data produced by the first instruction.

Procedural dependency: The instructions following a branch (taken or not taken) have a procedural dependency on the branch and cannot be executed until the branch is executed.

Resource conflicts: A resource conflict is a competition of two or more instructions for the same resource at the same time.

Output dependency: Two instructions update the same register, so the later instruction must update later.

Antidependency: A second instruction destroys a value that the first instruction uses.

A 7.
1) Instruction fetch strategies that simultaneously fetch multiple instructions, often by predicting the outcomes of, and fetching beyond, conditional branch instructions. These functions require the use of multiple pipeline fetch and decode stages, and branch prediction logic.
2) Logic for determining true dependencies involving register values, and mechanisms for communicating these values to where they are needed during execution.
3) Mechanisms for initiating, or issuing, multiple instructions in parallel.
4) Resources for parallel execution of multiple instructions, including multiple pipelined functional units and memory hierarchies capable of simultaneously servicing multiple memory references.
5) Mechanisms for committing the process state in correct order.

A 8. **Single instruction, single data (SISD) stream:** A single processor executes a single instruction stream to operate on data stored in a single memory.

Single instruction, multiple data (SIMD) stream: A single machine instruction controls the simultaneous execution of a number of processing elements on a lockstep basis. Each processing element has an associated data memory, so that each instruction is executed on a different set of data by the different processors.

Multiple instruction, multiple data (MIMD) stream: A set of processors simultaneously execute different instruction sequences on different data sets.

A 9.

1) There are two or more similar processors of comparable capability.
2) 2.These processors share the same main memory and I/O facilities and are interconnected by a bus or other internal connection scheme, such that memory access time is approximately the same for each processor.
3) All processors share access to I/O devices, either through the same channels or through different channels that provide paths to the same device.
4) All processors can perform the same functions (hence the term symmetric).
5) The system is controlled by an integrated operating system that provides interaction between processors and their programs at the job, task, file, and data element levels.

A 10. **Performance:** If the work to be done by a computer can be organized so that some portions of the work can be done in parallel, then a system with multiple processors will yield greater performance than one with a single processor of the same type.

Availability: In a symmetric multiprocessor, because all processors can perform the same functions, the failure of a single processor does not halt the machine. Instead, the system can continue to function at reduced performance.

Incremental growth: A user can enhance the performance of a system by adding an additional processor.

Scaling: Vendors can offer a range of products with different price and performance characteristics based on the number of processors configured in the system.

A 11. **Simultaneous concurrent processes:** OS routines need to be reentrant to allow several processors to execute the same IS code simultaneously. With multiple processors executing the same or different parts of the OS, OS tables and management structures must be managed properly to avoid deadlock or invalid operations.

Scheduling: Any processor may perform scheduling, so conflicts must be avoided. The scheduler must assign ready processes to available processors.

Synchronization: With multiple active processes having potential access to shared address spaces or shared I/O resources, care must be taken to provide effective synchronization. Synchronization is a facility that enforces mutual exclusion and event ordering.

Memory management: Memory management on a multiprocessor must deal with all of the issues found on uniprocessor machines, as is discussed in Chapter 8. In addition, the operating system needs to exploit the available hardware parallelism, such as multiported memories, to achieve the best performance. The paging mechanisms on different processors must be coordinated to enforce

consistency when several processors share a page or segment and to decide on page replacement.

Reliability and fault tolerance: The operating system should provide graceful degradation in the face of processor failure. The scheduler and other portions of the operating system must recognize the loss of a processor and restructure management tables accordingly.

A 12. Software cache coherence schemes attempt to avoid the need for additional hardware circuitry and logic by relying on the compiler and operating system to deal with the problem. In hardware schemes, the cache coherence logic is implemented in hardware.

A 13. If the L1 cache uses a write-through policy, as is done on the S/390 described in Section 17.2, then the L1 cache does not need to know the M state. If the L1 cache uses a write-back policy, then a full MESI protocol is needed between L1 and L2

A 14. **Pipelining:** Individual instructions are executed through a pipeline of stages so that while one instruction is executing in one stage of the pipeline, another instruction is executing in another stage of the pipeline.

Superscalar: Multiple pipelines are constructed by replicating execution resources. This enables parallel execution of instructions in parallel pipelines, so long as hazards are avoided.
Simultaneous multithreading (SMT): Register banks are replicated so that multiple threads can share the use of pipeline resources.

A 15. **Multi-threaded native applications:** Multi-threaded applications are characterized by having a small number of highly threaded processes. Examples of threaded applications include Lotus Domino or Siebel CRM (Customer Relationship Manager).

Multi-process applications: Multi-process applications are characterized by the presence of many single-threaded processes. Examples of multi-process applications include the Oracle database, SAP, and PeopleSoft.

Java applications: Java applications embrace threading in a fundamental way. Not only does the Java language greatly facilitate multithreaded applications, but the Java Virtual Machine is a multithreaded process that provides scheduling and memory management for Java applications. Java applications that can benefit directly from multicore resources include application servers such as Sun's Java Application Server, BEA's Weblogic, IBM's Websphere, and the opensource Tomcat application server. All applications that use a Java 2 Platform, Enterprise Edition (J2EE platform) application server can immediately benefit from multicore technology.

Multi-instance applications: Even if an individual application does not scale to take advantage of a large number of threads, it is still possible to gain from

multicore architecture by running multiple instances of the application in parallel. If multiple application instances require some degree of isolation, virtualization technology (for the hardware of the operating system) can be used to provide each of them with its own separate and secure environment.

A 16.

- The number of core processors on the chip
- The number of levels of cache memory
- The amount of cache memory that is shared

A 17.

1) Constructive interference can reduce overall miss rates. That is, if a thread on one core accesses a main memory location, this brings the frame containing the referenced location into the shared cache. If a thread on another core soon thereafter accesses the same memory block, the memory locations will already be available in the shared onchip cache.
2) A related advantage is that data shared by multiple cores is not replicated at the shared cache level.
3) With proper frame replacement algorithms, the amount of shared cache allocated to each core is dynamic, so that threads that have a less locality can employ more cache.
4) Interprocessor communication is easy to implement, via shared memory locations.
5) The use of a shared L2 cache confines the cache coherency problem to the L1 cache level, which may provide some additional performance advantage.