1. **Thread:** is a lightweight process, used as a process management technique.
2. threads provide a way to improve application performance through parallelism.
3. Traditional (or heavyweight) process has a single thread of control.
4. **Single-threaded approach:** the traditional approach of a single thread of execution per process.
5. If a process has multiple threads of control, it can perform more than one task at a time.
6. **Multithreading:** the ability of an OS to support multiple, concurrent paths of execution within a single process.
7. In a single-threaded model, the representation of a process includes its PCB, user address space and user/kernel stacks to manage the call/return behavior of process execution.
8. In a multithreaded environment, there is still a single PCB and user address space. However, there are separate stacks for each thread, as well as a separate thread control block containing register values, priority, and other thread-related state information.
9. There are two broad categories of thread implementation: **user-level threads (ULTs)** and **kernel-level threads (KLTs)**.
10. User-level threads are supported above the kernel and are managed without kernel support.
11. Kernel-level threads are supported and managed directly by the operating system.
12. Any application can be programmed to be multithreaded by using a **“threads library”**.
13. **Threads library:** provides the programmer with an API for creating and managing threads.
14. The ULT adopts the **Many-to-One Model**, maps many user-level threads to one kernel thread (process).
15. A **One-to-One Model** is adopted for the KLT that maps each user thread to a kernel thread.
16. The **Many-to-Many Model** is adopted in the combined approach which multiplexes many user-level threads to a smaller or equal number of kernel threads.
17. Cooperating processes can either ***directly share a logical address space*** or ***share data only through files or messages***.
18. The basic requirement for support of concurrent processes is the ability to enforce **“mutual exclusion”**.
19. **Mutual Exclusion:** a requirement that when one process is in a **critical section** that accesses shared resources, no other process may be in a **critical section** that accesses any of those shared resources.
20. **Critical section:** a section of code within a process that requires access to shared resources and that must not be executed while another process is in a corresponding section of code.
21. Both **“processes interleaving”** in a uniprocessor multiprogramming system, and **“processes overlapping”** in a multiprocessor system, can be viewed as examples of concurrent processing.
22. Overlap processing overlaps time records. With overlap processing, a percentage of data from one time record is reused in the next time record.
23. **Processes unaware of each other (competition)**
24. **Processes indirectly aware of each other (Cooperation by Sharing)**
25. **Processes directly aware of each other (Cooperation by Communication)**
26. **Semaphore:** an integer value used for signaling among processes in order to synchronize their activities, hence achieving concurrency.
27. Semaphore **s** is accessed only through two standard operations:
28. **signal(s)**
29. **wait(s)**
30. Only three operations may be performed on a semaphore:
31. **Initialize:** to a nonnegative integer value.
32. **Decrement:** **wait(s)** decrements the semaphore value. If it becomes negative, the process executing the **wait(s)** operating is blocked, otherwise continues execution.
33. **Increment:** **signal(s)** increments the semaphore value. If it is less than or equal to zero, then a process blocked by a **wait(s)** operation is unblocked.
34. A semaphore whose definition includes the FIFO policy is called a **strong semaphore**.
35. A semaphore that does not specify the order in which processes are removed from the queue is a **weak semaphore**.
36. For mutual exclusion, **strong semaphores** guarantee freedom from starvation, while **weak semaphores** do not.
37. producer won’t try to add data into the buffer if it’s full and that the consumer won’t try to remove data from an empty buffer.
38. ***there are one or more producers generating some type of data and placing these in a buffer. There is a single consumer that is taking items out of the buffer one at a time.***
39. **Deadlock:** permanentblocking of a set of processes that either compete for system resources or communicate with each other.
40. **Why deadlock is permanent?** Because none of the events is ever triggered.
41. Two general categories of resources can be distinguished: **reusable and consumable**.
42. **Reusable resource:** one that can be safely used by a process and is not depleted by that use. Processes obtain resource units that they later release for reuse by other processes.
43. **Consumable Resource:** one that can be created (produced) and destroyed (consumed).
44. There is no limit on the number of consumable resources of a particular type.
45. **The conditions of policy must be present for a deadlock to occur:**
46. **Mutual exclusion**
47. **Hold and wait:** a process may hold allocated resources while awaiting assignment of other resources.
48. **No preemption:** no resource can be forcibly removed from a process holding it.
49. **Circular wait:** a closed chain of processes exists, such that each process holds at least one resource needed by the next process in the chain.
50. The Conditions for Deadlock
51. **Resource allocation graph:** a useful tool in characterizing the allocation of resources to processes.
52. Dealing with Deadlock
53. **Three general approaches exist for dealing with deadlock:**
54. **Deadlock prevention:** to design a system in such a way that the possibility of deadlock is excluded.
55. Deadlock prevention methods fall into two classes:
56. ***Indirect prevention:*** prevent the occurrence of one of the three necessary conditions, i.e. mutual exclusion, hold and wait, no preemption.
57. ***Direct prevention:*** prevent the occurrence of a circular wait.
58. **Deadlock avoidance:** allows the three necessary conditions ***(mutual exclusion, hold & wait, no preemption)*** but makes sensible choices to assure that deadlock point is never reached.
59. **Deadlock detection:** detect the presence of deadlock (conditions 1 through 4 hold) and take action to recover.
60. The state consists of the two vectors, **Resource** and **Available**, and the two matrices, **Claim** and **Allocation**.
61. **Safe state:** is a state in which there is “at-least” one sequence of resource allocations to processes that does not result in a deadlock (i.e., all of the processes can be run to completion).
62. **The main memory can be divided into two parts:**
63. Part for the operating system (kernel)
64. Part for the program currently being executed (user part).
65. **Memory management satisfy the following requirements:**
66. Relocation
67. Protection
68. Sharing
69. Logical organization
70. Physical organization
71. The main memory in a computer system is organized as a linear or one-dimensional address space, consisting of a sequence of **bytes or words**.
72. Secondary memory is similarly organized
73. **Physical Organization:**
74. The computer memory is organized into at least two levels, referred to as **main memory** and **secondary memory**.
75. **Frame:** A fixed-length block of main memory.
76. **Page:** A fixed-length block of data that resides in secondary memory (such as hard disk).
77. ***A page of data may temporarily be copied into a frame of main memory.***
78. **Segment:** A variable-length block of data that resides in secondary memory (such as hard disk).
79. Fixed Partitioning:
80. **For OS:** It can be assumed that OS occupies some fixed portion of main memory and that rest of main memory is available for use by multiple processes.
81. The simplest scheme for managing memory is to partition it into regions with **fixed boundaries**.
82. To overcome some of the difficulties with **fixed partitioning**, an approach known as **dynamic partitioning** was developed.
83. **External fragmentation:** In dynamic portioning, as time goes on, the memory becomes more and more fragmented and memory utilization declines.
84. **Compaction:** a technique used for overcoming external fragmentation, where from time to time, the OS shifts the processes so that they are contiguous and so that all of the free memory is together in one block.
85. The difficulty with compaction is that it is a time consuming procedure and wasteful of processor time.
86. For data exchange between processor and memory, the processor uses two internal registers:
87. Memory address register (MAR): specifies the address in memory for the next read or write.
88. Memory buffer register (MBR), contains the data to be written into memory or it receives the data read from memory.
89. Similarly, for data exchange between processor and I/O, the following registers are used:
90. I/O address register (I/OAR) specifies a particular I/O device.
91. I/O buffer register (I/OBR) is used for the exchange of data between an I/O module and the processor.
92. **Instruction cycle:** the processing required for a single instruction.
93. The actions a processor takes generally fall into four categories:
94. **Processor-memory:** Data may be transferred from processor to memory or from memory to processor.
95. **Processor-I/O:** Data may be transferred to or from a peripheral device by transferring between the processor and an I/O module.
96. **Data processing:** The processor may perform some arithmetic or logic operation on data.
97. **Control:** An instruction may specify that the sequence of execution be altered.
98. Interrupt: a mechanism by which other modules (I/O, memory) may interrupt the normal sequencing of the processor.
99. **Main benefit:** interrupts are provided primarily as a way to improve processor utilization.
100. There is a trade-off among the three key characteristics of memory: namely, capacity, access time, and cost.
101. A variety of technologies are used to implement memory systems, where the following relationships hold:
102. **Faster access time, greater cost per bit.**
103. **Greater capacity, smaller cost per bit.**
104. **Greater capacity, slower access speed.**
105. The way to meet both large capacity and performance is to employ a memory hierarchy.
106. Going down the hierarchy, the following occurs:
107. **Decreasing cost per bit.**
108. **Increasing capacity.**
109. **Increasing access time.**
110. **Decreasing frequency of access to the memory by the processor.**
111. **Instruction set architecture (ISA):** The ISA defines the range of machine language instructions that a computer can follow. This interface is the boundary between hardware and software. Note that both application programs and utilities may access the ISA directly.
112. **Application binary interface (ABI):** The ABI defines a standard for binary portability across programs.
113. **Application programming interface (API):** The API gives a program access to the hardware resources and services available in a system through the ISA supplemented with high-level language (HLL) library calls.
114. **Process:** is a program in execution *OR* an instance of a program running on a computer.
115. There are four main causes of system programming errors:
116. **Improper synchronization:** e.g. improper design of signaling.
117. **Failed mutual exclusion:** e.g. access same file simultaneously
118. **Nondeterminate program operation:** multiple programs in shared memory
119. **Deadlocks:** e.g. two I/O devices waiting for each other.
120. **Process isolation:** The OS must prevent independent processes from interfering with each other’s memory, both data and instructions.
121. **Automatic allocation and management:** Programs should be dynamically allocated across the memory hierarchy as required.
122. **Support of modular programming:** Programmers should be able to define program modules, create, destroy and alter the size dynamically.
123. **Protection and access control:** OS must allow portions of memory to be accessible in various ways (protected and shared).
124. **Long-term storage:** Many application programs require means for storing information for extended periods of time.
125. **Monolithic kernel:** the entire operating system runs as a single program in kernel mode. OS functionality is provided in these large kernels, including scheduling, file system, networking, device drivers, memory management, and more. ***E.g. Linux***.
126. **Microkernel:** putting as little as possible in kernel mode because bugs in the kernel can bring down the system instantly. ***E.g. Minix***.
127. Only few essential functions are assigned to the kernel, including address spaces, interprocess communication (IPC), and basic scheduling.
128. Other OS services are provided by processes that run in user mode and are treated like any other application (well suited to a distributed environment).
129. **Symmetric multiprocessing (SMP):** The OS of an SMP schedules processes or threads across all of the processors.
130. SMP has a number of potential advantages over uniprocessor architecture:
131. **Performance**
132. **Availability**
133. **Incremental growth**
134. **Multithreading:** is a technique in which a process is divided into threads that can run concurrently.
135. **Distributed operating system:** provides the illusion of single main/secondary memory space, plus other unified access facilities, such as a distributed file system.
136. **Object-oriented design:** lends discipline to the process by adding modular extensions to a small kernel.