**System call:** programming interface to the services provided by the OS. You use it to interact with the kernel.

**Layered:** The OS is divided into several layers, each built on top of lower layers. The bottom layer is the hardware. The higher is the user interface.

**Adv:** Easy debugging; as the high layer does not need to know how the lower layer is implemented and we can check the program layer by the layer to find out in which layer the problem occurs.

**Disadv:** layer approach is less efficient the monolithic because a program has to go through every layer, even the layer that are not needed, and each layer adds overheads to the system call.

**Microkernel:** Making the kernel as small as possible by moving all the non-essentials into the user space: uses object-oriented approach and divides the kernels into modules. Each core component is separate. **Communication between each module are done through message passing.**

**Adv:** Easies to extend microkernel, to port the OS to new architectures, more reliable and secure. It is easy to construct and debug as we can test each module to know in which the problem occurs. When a module is updated, we would only need to update/recompile programs that use that module.

**(Memory): If the service crush they don’t crus the whole system.**

**(structure): recompile, portable, secure, and reliable.**

**Disadv:** There are lots of performance overhead for the user space to kernel space communication. All modules are leaded even when they are not required.

**Loadable modules:** Just like microkernel, however each module would only be loaded within the kernel when it is required.

**Adv:** The kernel becomes smaller then microkernel because we would only load modules that required. Communicates through function calls.

**Faster b/c communication is faster. No hierarchical restrictions. Any module can talk to any other module.**

**Monolithic:** Consists of 2 separate parts, the system programs and kernels. Kernel is located between the system call interface and the hardware.

**Disadv:** Because all kernel functionality is stores in one level, we must compile all procedures used into one file: making it hard to implement and maintain. We need to keep recompiling the whole program when the kernel or the program is modified. There is no information hiding, every procedure can call all other procedures.

|  |  |
| --- | --- |
| New: the process is being created  Running: instructions are being executed on CPU  Waiting: the process is waiting for some event to occur  Ready: the process is waiting to be assigned to a processor  Terminated: the process has finished execution. |  |

**Process Control Block (PCB): Save CPU. WHY?** B/c restore the process that use the CPU.

**Time Quantum (Slice): slow:** the user feels the delay. **Fast:** the user will feel fast. The time spent in kernel will be more.

**OS is interrupt driven**

1. I/O Completion: request from the kernel
2. Time interrupt (time slice quantum): Timer to prevent infinite loop / process hogging resources.
3. Exceptions: Software error (division by zero, invalid memory access).
4. System calls: request for operating system service.