1.2.2 Multiprogrammed Systems

The most important aspect of job scheduling is the ability to multiprogram. A single user cannot keep either the CPU or the I/O devices busy at all times. Multiprogramming increases CPU utilization by organizing jobs so that the CPU always has one to execute.

The idea is as follows: The operating system keeps several jobs in memory simultaneously (Figure 1.3). This set of jobs is a subset of the jobs kept in the job pool-since the number of jobs that can be kept simultaneously in memory is usually much smaller than the number of jobs that can be in the job pool. The

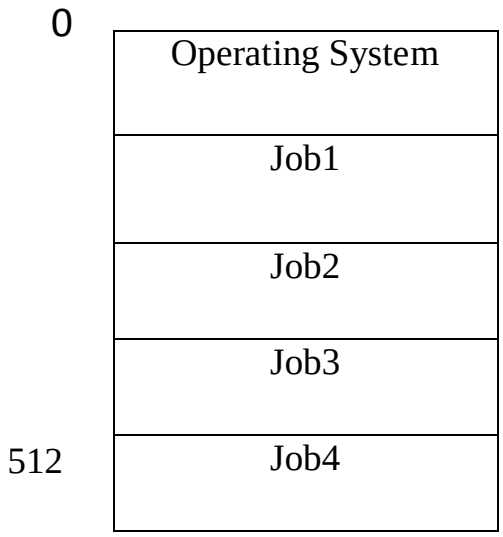

Fig. 1.3 Memory layout for a multiprogramming system.

operating system picks and begins to execute one of the jobs in the memory. Eventually, the job may have to wait for some task, such as an I/O operation, to complete. In a non-multiprogrammed system, the CPU would sit idle. In a multiprogramming system, the operating system simply switches to, and executes, another job. When *that* job needs to wait, the CPU is switched to *another* job, and so on.

Multiprogramming is the first instance where the operating system must make decisions for the users. All the jobs that enter the system are kept in the job pool. This pool consists of all processes residing on disk awaiting allocation of main memory. If several jobs are ready to be brought into memory, and if there is not enough room for all of them, then the system must choose among one them. Making this decision is *job scheduling*. When the operating system selects a job from the job pool, it loads that job into memory for execution. Having several programs in memory at the same time requires some form of memory management. In addition, if several jobs are ready to run at the same time, the system must choose among them. Making this decision is *CPU scheduling*. Finally, multiple jobs running concurrently require that their ability to affect one another be limited in all phases of the operating system, including process scheduling, disk storage, and memory management.

1.2.3 Time-Sharing Systems

Multiprogrammed, batched systems provided an environment where the various system resources (for example, CPU, memory, peripheral devices) were utilized effectively, but it did not provide for user interaction with the computer system. Time sharing (or multitasking) is a logical extension of multiprogramming. The CPU executes multiple jobs by switching among them, but the switches occur so frequently that the users can interact with each program while it is running.

An interactive (or hands-on) computer system provides direct communication between the user and the system. The user gives instructions to the operating system or to a program directly, using a keyboard or a mouse, and waits for immediate results. Accordingly, the response time should be short typically within one second or so. A time-shared operating system allows many users to share the computer simultaneously. Since each action or command in a time-shared system

tends to be short, only a little CPU time is needed for each user. As the system switches rapidly from one user to the next, each user is given the impression that the entire computer system is dedicated to her use, even though it is being shared among many users.

A time-shared operating system uses CPU scheduling and multiprogramming to provide each user with a small portion of a time-shared computer. Each user has at least one separate program in memory. A program loaded into memory and executing is commonly referred to as a process. Since interactive I/O typically runs at "people speeds," it may take a long time to complete. Rather than let the CPU sit idle when this interactive input takes place, the operating system will rapidly switch the CPU to the program of some other user.

Time-sharing operating systems are even more complex than multiprogrammed operating systems. In both, several jobs must be kept simultaneously in memory, so the system must have memory management and protection. To obtain a reasonable response time, jobs may have to be swapped in and out of main memory to the disk that now serves as a backing store for main memory. A common method for achieving this goal is virtual memory, which is a technique that allows the execution of a job that may not be completely in memory. The main advantage of the virtual-memory scheme is that programs can be larger than **physical memory.**

Time-sharing systems must also provide a file system, a mechanism for concurrent execution, mechanisms for job synchronization and communication and it may ensure that jobs do not get stuck in a deadlock, forever waiting for one another.

Accordingly, multiprogramming and time sharing are the central themes of modern operating systems.

1.3 Desktop Systems

Personal computers PCs appeared in the 1970s. During their first decade, the CPUs in PCs lacked the features needed to protect an operating system from user programs. PC operating systems therefore were neither multiuser nor multitasking. However, the goals of these operating systems have changed with time; instead of maximizing CPU and peripheral utilization, the systems opt for maximizing user convenience and responsiveness.

Operating systems for these computers have benefited in several ways from the development of operating systems for mainframes. Microcomputers were immediately able to adopt some of the technology developed for larger operating systems. On the other hand, the hardware costs for microcomputers are sufficiently low that individuals have sole use of the computer, and CPU utilization is no longer a prime concern. Thus, some of the design decisions made in operating systems for mainframes may not be appropriate for smaller systems. Other design decisions still apply. For example, file protection was, at first, not necessary on a personal machine. However, these computers are now often tied into other computers over local-area networks or other Internet connections. When other computers and other users can access the files on a PC, file protection again becomes a necessary feature of the operating system. The lack of such protection has made it easy for malicious programs to destroy data on systems such as MS-DOS and the Macintosh operating system. These programs may be self-replicating, and may spread rapidly via **worm** or **virus** mechanisms and disrupt entire companies or even worldwide networks. Advanced timesharing features such as protected memory and file permissions are not enough, on their own, to safeguard a system from attack.

1.4 Multiprocessor Systems

Most systems to date are single-processor systems; that is, they have only one main CPU. However, **multiprocessor systems** (also known as **parallel systems** or **tightly coupled systems)** are growing in importance. Such systems have more than one processor in close communication, sharing the computer bus, the clock, and sometimes memory and peripheral devices. Multiprocessor systems have three main advantages.

1. Increased throughput. By increasing the number of processors, we hope to get more work done in less time. The speed-up ratio with N processors is not N; rather, it is less than *N*. When multiple processors cooperate on a task, a certain amount of overhead is incurred in keeping all the parts working correctly. This overhead, plus contention for shared resources, lowers the expected gain from additional processors. Similarly, a group of *N* programmers working closely together does not result in N times the amount of work being accomplished.

2. Economy of scale. Multiprocessor systems can save more money than multiple single-processor systems, because they can share peripherals, mass storage, and power supplies. If several programs operate on the same set of data, it is cheaper to store those data on one disk and to have all the processors share them, than to have many computers with local disks and many copies of the data.

3. Increased reliablility. If functions can be distributed properly among several processors, then the failure of one processor will not halt the system, only slow it down. If we have ten processors and one fails, then each of the remaining nine processors must pick up a share of the work of the failed processor. Thus, the entire system runs only 10 percent slower, rather than failing altogether. This ability to continue providing service proportional to the level of surviving

hardware is called **graceful degradation.** Systems designed for graceful degradation are also called **fault tolerant.**

Continued operation in the presence of failures requires a mechanism to allow the failure to be detected, diagnosed, and, if possible, corrected. The Tandem system uses both hardware and software duplication to ensure continued operation despite faults. The system consists of two identical processors, each with its own local memory. The processors are connected by a bus. One processor is the primary and the other is the backup. Two copies are kept of each process: one on the primary processor and the other on the backup. At fixed checkpoints in the execution of the system, the state information of each job including a copy of the memory image-is copied from the primary machine to the backup. If a failure is detected, the backup copy is activated and is restarted from the most recent checkpoint. This solution is expensive, since it involves considerable hardware duplication.

The most common multiple-processor systems now use **symmetric multiprocessing** (SMP), in which each processor runs an identical copy of the operating system, and these copies communicate with one another as needed. Some systems use **asymmetric multiprocessing,** in which each processor is assigned a specific task. A master processor controls the system; the other processors either look to the master for instruction or have predefined tasks. This scheme defines a master-slave relationship. The master processor schedules and allocates work to the slave processors.

SMP means that all processors are peers; no master-slave relationship exists between processors. Each processor concurrently runs a copy of the operating system.

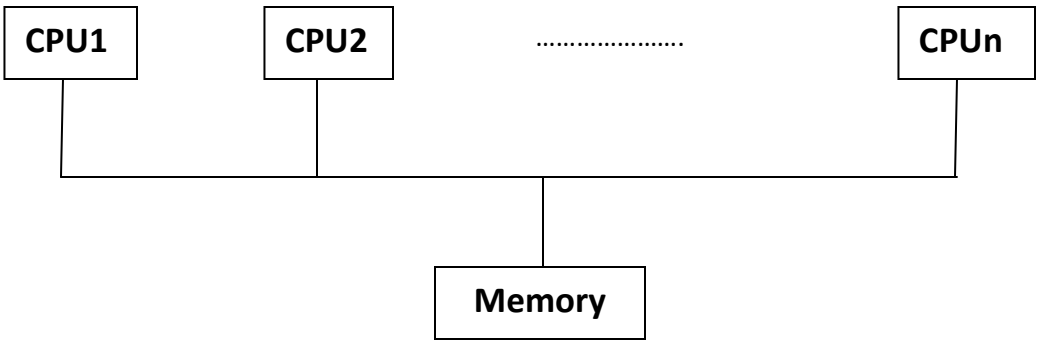

Fig 1.4 illustrates a typical SMP architecture.

The difference between symmetric and asymmetric multiprocessing may be the result of either hardware or software. Special hardware can differentiate the multiple processors, or the software can be written to allow only one master and multiple slaves.