GENERAL INFORMATICS

6.1. Information System Life Cycle

6.2. Steps in computer problem solving process

6.3. Steps for preparing a program for execution

6.4. Executing a program

Chapter 6.

Steps in Computer Problem Solving Process

Information System

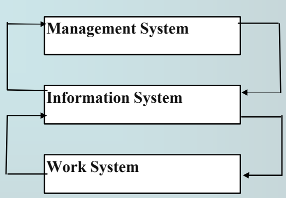

Figure 1.28. The information system

The enterprise information system is the system that includes all enterprise components whose actions are of informational type. In each information system on retrieve varies elements: decision makers, users, processing devices and system software together with application programs, communication devices etc.

The main objective of the information system is to supply, to the enterprise decision makers, the needed information to control, decide and act, that means those information that have value for decision maker. The information system capture the information that comes from the work system or from the external environment, process that information and communicate to the management system that take decisions on the basis of that information, decisions that are send in turn to the information system and this one communicate this information.

Information (Informatic) System Categories

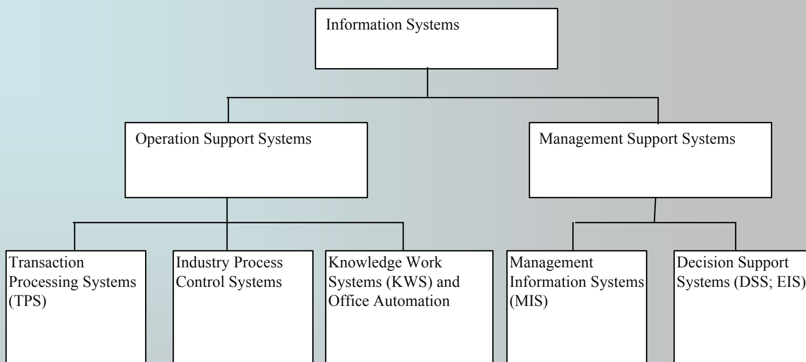

Figure 1.29. The categories of information (informatic) systems

Information System

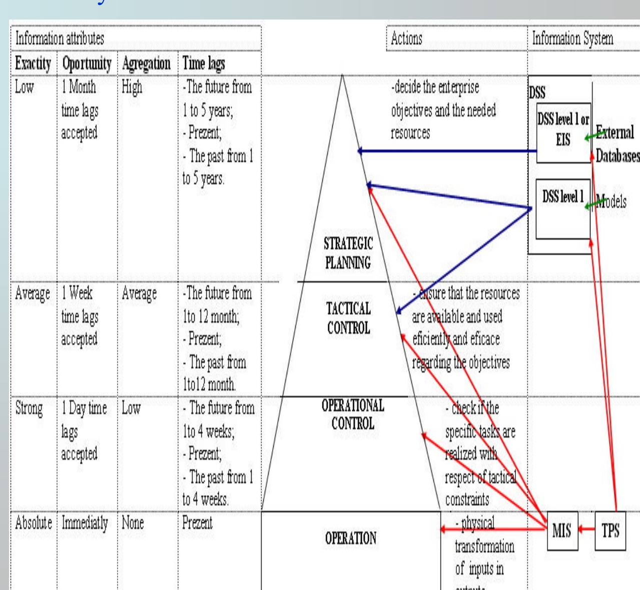

Figure 1.30. The relationship between the levels of management and the category of the informatic system

Information System

Components / Model	Data (What ?)	Processes (How ?)	Network (Where ?)	The played role viewed from the point of view of:	Summary description of the role (Who ?, When ?, Why ?)
1. Enterprise (business) area (domain)	List of main entities of the enterprise	List of the functions executed in the enterprise	List of locations of the enterprise	Owner	Offers a strategic point of view that includes: • the business dimension, • the mission, • objectives.
2. The Economic Model	Economic entities and the links among them	Function and Processes decomposition	Communication links	Architect	The presentation of economic models by specifying dimension, mission and objectives
3. Information System Model	The model and links of economic data	The flow between application processes	Distributed network	Designer	The information system design as helpful solution for the enterprise to reach his objectives
4. Technologies Model	Database Design	Processes Description	Configuration Project	Builder	Convert the informational system models designed according to the technological characteristics and constraints
5. The description of technologies	The description of database schema and database sub schemas	The program and control blocks code	Configuration description	Builder	Convert the technological models in instructions for informational structure generation
6. Information System	Data and Information	Application Programs	System Configuration	User	Administer, use and ensure the functionality of the whole system

Figure 1.31. The general frame for analyzing information system

Information (Informatic) System Life Cycle

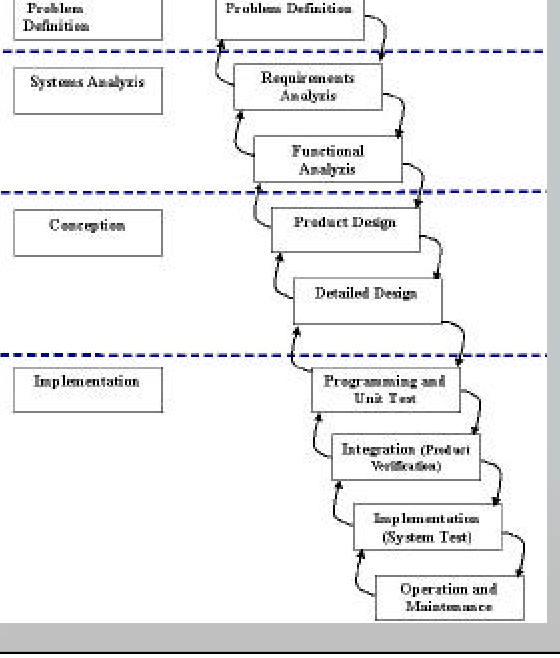

Figure 1.32. The waterfall lifecycle

Information System Life Cycle

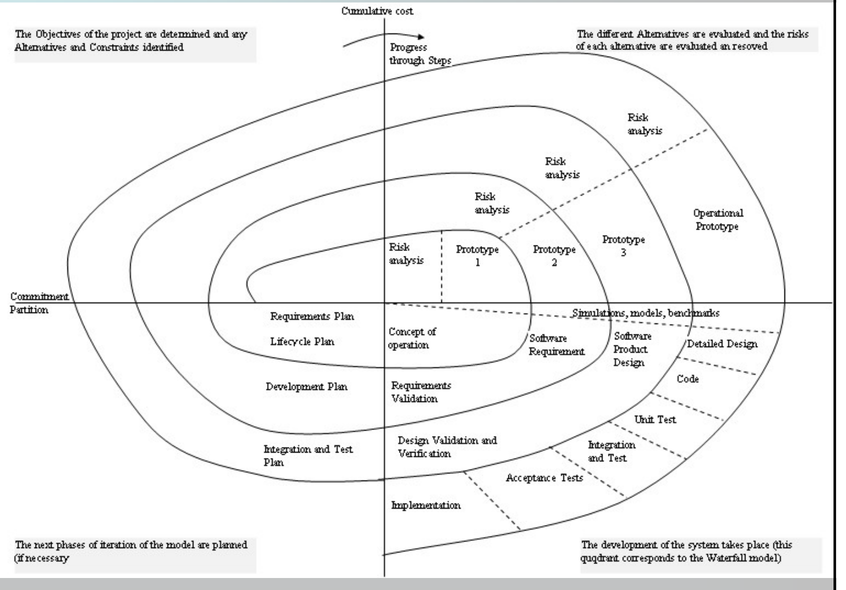

Figure 1.33. The spiral model of systems development

Information System Life Cycle

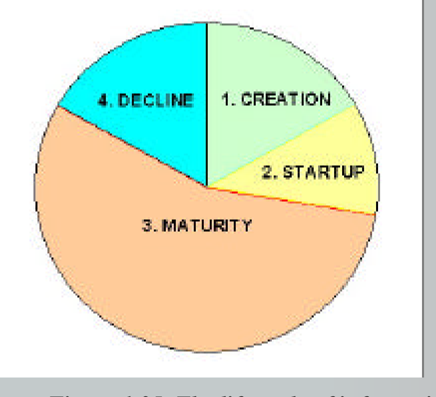

Figure 1.35. The life cycle of information systems

Steps in problem solving process by using a programming environment

- The problem solving process starts with the problem specification and ends with a concrete (and correct) program.
- The steps to do in the problem solving process may be: problem definition, problem analysis, algorithm development, coding, program testing and debugging, and documentation.

- The stages of analysis, design, programming, implementation, and operation of an information system forms the life cycle of the system.
- We briefly describe the steps in problem solving process by using a programming environment (it can allow the “around” application programming by the possibility of generating programs from general templates, for example) and by considering only a specific process from the whole system. In this context the stages can be:

- **1st. Defining/Specifying the problem** [Theme] - by answering to questions as:

  What the computer program do?

  What tasks will it perform?

  What kind of data will it use, and where will get its data from?

  What will be the output of the program?

  How will the program interact with the computer user?

Specifying the problem requirements forces you to state the problem clearly and unambiguously and to gain a clear understanding of what is required for its solution.

Your objective is to eliminate unimportant aspects and to focus on the root problem, and this may not be as easy as it sound.

- 2^{nd}. **Analyzing the problem** [Analysis] involves identifying the problem (a) inputs, that is, the data you have to work with; (b) outputs, the desired results; and (c) any additional requirements or constraints on the solution.

- 3^{rd}. **Algorithm development**: find an algorithm for its solution [Design].

Write step-by-step procedure and then verify that the algorithm solves the problem as intended.

The development can be expressed as:

- **pseudocode** – a narrative description of the flow and logic of the intended program, written in plain language that expresses each step of the algorithm;
- **flowchart** - a graphical representation that uses graphic symbols and arrows to express the algorithms.

♦ After you write the algorithm you must realize step-by-step simulation of the computer execution of the algorithm in a so called desk-check process (verifying the algorithm).

♦ **4th. Coding** (or programming): is the process of translating the algorithm into the syntax of a given programming language [Programming]. You must convert each algorithm step into one or more statements in a programming language.

♦ 5th. **Testing and debugging**:

- **testing** means running the program, executing all its instructions/functions, and testing the logic by entering sample data to check the output;
- **debugging** is the process of finding and correcting program code mistakes:
  - **syntax errors;**
  - **run-time errors;**
  - **logic errors** (or so called **bugs**).
- **field testing** is realized by users that operate the software with the purpose of locating problems.

♦ 6th. **Documenting** the program by:

- **internal documentation**;
- **external documentation**.

♦ 7th. **Integrate the program in the data process flow** (Implementation) and use the program to solve problems [Exploitation].

Steps for preparing a program for execution

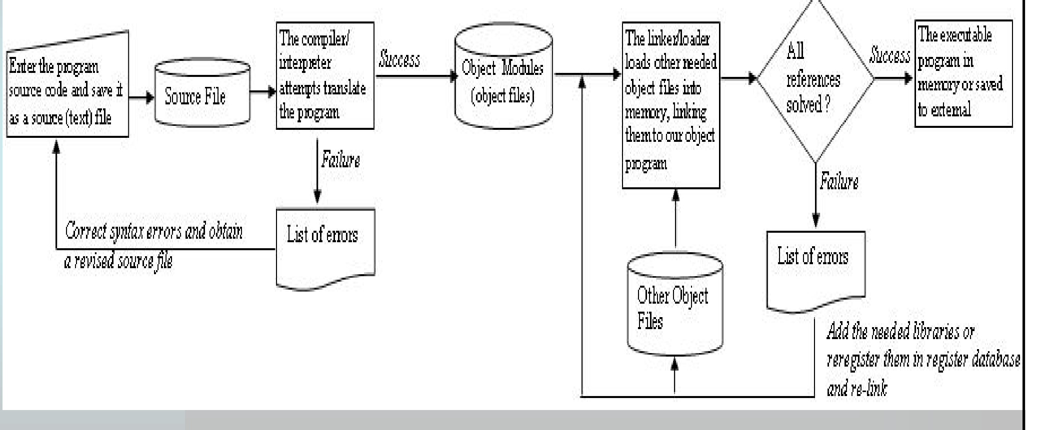

Figure 3. 1 Steps for preparing a program for execution

Executing a Program

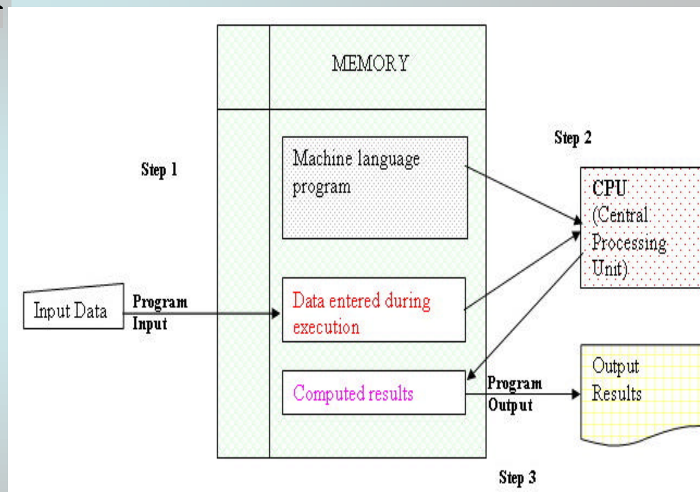

Figure 3. 2 Executing a program

Virtual Memory

Definition.

To use the processor and the I/O facilities efficiently, it is desirable to maintain many processes, as possible, in main memory. In addition, it is desirable to free programmers from size restrictions in program development than to restrict them with small sizes (that happened in the older computers). The restriction to a predefined size redirects the programmers effort from the use of better programming techniques to a continuously effort to make fit in that size a solution, not necessarily the optimal one. The way to address both of these concerns is virtual memory (VM). Virtual memory systems are an abstraction of the primary memory in a von Neumann computer. Even in a time of decreasing physical memory costs, contemporary computers devote considerable resources to supporting virtual address spaces that are much larger than the physical memory allocated to a process. Contemporary software relies heavily on virtual memory to support applications such as image management with huge memory requirements.

Virtual Memory

The virtual memory abstraction is built on the idea of runtime address binding. The compiler and the linkage editor create an absolute module that the loader traditionally binds to physical addresses before the program executes. Hardware facilities enable a memory manager to automatically load portions of a virtual address space is left in secondary memory. With virtual memory, all address references are logical references that are translated at run time to real addresses. This allows a process to be located anywhere in main memory and for that location to change over time. Virtual memory also allows a process to be broken up into pieces. These pieces need not be contiguously located in main memory during execution end, indeed, it is not even necessary to all of the pieces of the process to be in main memory during execution.

Virtual Memory

Address Space Mapping

The components of a source program are represented using: symbolic identifiers, labels and variables these entities are elements of the name space. Each symbolic name in the name space is translated into an absolute image by the compiler and link editor. Each virtual address is converted to a physical address in the primary memory when the absolute image is translated into an executable image is translated into an executable image by loader (see figure 6.1).

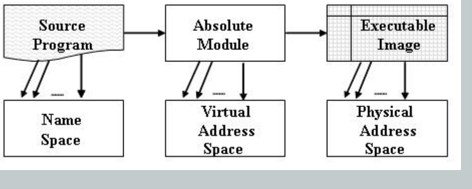

Figure 6.1. Address space mapping

Virtual Memory

Address Space Mapping

Each program can contain data and instructions whose allocation, realized by the compiler starts with 0. When we load many programs into the main memory (for example in Windows you want use simultaneously Word, Paint and Power Point to realize data transfers by using the clipboard between this applications) these programs cannot start from address 0; instead, they starts from an address allowed by the operating system. In this context the hardware must distinguish between the relative address (a_{rel}) and the absolute address (a_{abs}). A relative address is considered relative to the memory address to which the program is loaded (we consider for the shake of simplicity that the program is continuously loaded starting with that address). If we store somewhere this address (base address – a_{base}) allowed to the program then the computation of the absolute (physical) address will be simply done by the formula: $a_{abs}=a_{base}+a_{rel}$.

Virtual Memory

Implementing Virtual Memory

To basic approaches to providing virtual memory are: paging and segmentation.

Paging. With paging, each process is divided into relatively small, fixed-size pages. Paging systems transfer fixed-sized blocks of information between primary and secondary memories. Because of the fixed pages size and page frame size, the translation from a binary virtual address to a corresponding physical address is relatively simple, provided the system has an efficient table lookup mechanism. Paging systems use associative memories to implement page translation tables. Paging uses single-component addresses, like those used to address cell within any particular segment. In paging, the virtual address space is a linear sequence of virtual address (a format that differs from the hierarchical segmentation address space. In a paging system, the programmer has no specific mechanism for informing the virtual memory system about logical units of the virtual address space, as is done in segmentation. Instead, the virtual memory manager is completely responsible for defining the fixed-size unit of transfer – the page – to be moved back and forth between the primary and secondary memories. The programmer need not be aware of the units of virtual address space loaded into or unloaded from the physical memory. In fact, the page size is transparent to the process.

Virtual Memory

Implementing Virtual Memory

Segmentation. Segmentation provides for the use of pieces of varying size. It is also possible combine segmentation and paging in a single memory-management scheme. Segmentation is an alternative to paging. It differs from paging in that the unit transfer between primary and secondary memories varies. The size of the segments, are also explicitly known by the programmer. Translating a segment virtual address to a physical**.**
Segmentation is an extension of the ideas suggested by the use of relocation-limit registers for relocating and bound checking blocks of memory. The program parts to be loaded or unloaded are defined by the programmer as variable-sized segments. Segment may be defined explicitly by language directives it implicit by program semantics as the: text, data and stack segments created by the UNIX C compiler. address is more complex that translating a paging virtual address.

Virtual Memory

Every Win 2k process is given a fixed-size virtual address space – 4 gigabytes (GB), which, of course, is much larger than the amount of primary memory in any contemporary computer. The process does not necessarily use all of the virtual address space – only as much as it need. Ordinary the .EXE for a program is very much smaller than the address space. Part of the address space, usually 2 GB, is used to reference addresses used by the OS (it is the supervisor space). Even though the supervisor space portion of the address space exists in a process's virtual address space, the memory can be referenced only by a thread if the processor is in supervisor mode. The OS needs some means of determining the amount of the address space that the process intends to use. The link editor builds the static execution image in an .EXE file that will generally define the address space. Dynamically linked libraries an anther dynamically allocated portions of the address space can be added to the virtual address space at runtime. There are two phases to dynamically adding addresses to the address space (see figure 6.6)

Virtual Memory

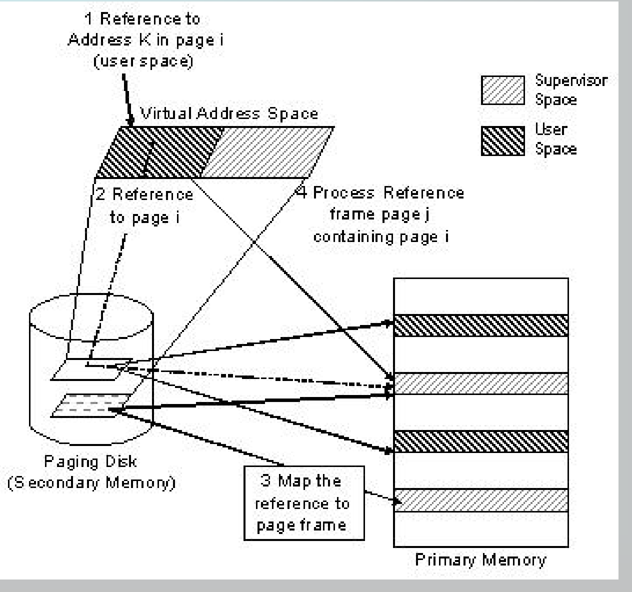

Figure 6.6. Windows NT Paging System

Virtual Memory

1. Reserving a portion of the address space called a region;
2. Committing a block of pages in the region in the address space.

A thread in a process can dynamically reserve a region of virtual addresses without actually causing anything to be written to the secondary storage page file (also sometimes called the paging file). A thread in the process may also subsequently release a region of addresses it previously reserved.

The second phase is to commit addresses that were previously reserved. Once a portion of the address space has been committed, space is allocated in the page file. If a thread in the process that references committed memory, the page containing the referenced address will be loaded from the page file into the primary memory. Each processor support a particular allocation granularity to determine the minimum size of a block of addresses that can be reserved.

A virtual memory-management scheme requires both **hardware** and **software**. The hardware support is provided by the processor. The support includes dynamic translation of virtual address to physical address and the generations are interrupted when a referenced page or segment is not in main memory.

Process Description, Control, and Management

Process. Let P be a program described in a programming language that we designate by the term Language(P). A process, denoted by Process(P), of P is a description of a series of actions or operations, needed to completely solving of P, description realized in such a way as that in which the program executes in a virtual processor that implements the Language(P). A process is nothing else than a description of a program behavior and, consequently, a running program can be considered a process. However, that definition of a running program as a process is not quite exactly.

Examples:

Process(P{p, q}): Process1(p) and Process2(q);

P*{p, r} with p€ P{p, q}. P and P* are different programs.

Process Inference

Process Description, Control, and Management

Process Inference

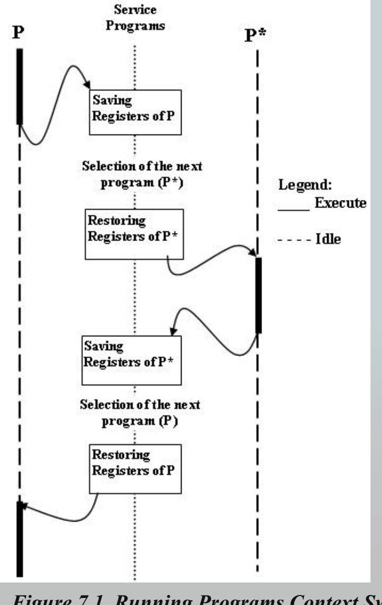

Figure 7.1. Running Programs Context Switching

Process Description, Control, and Management

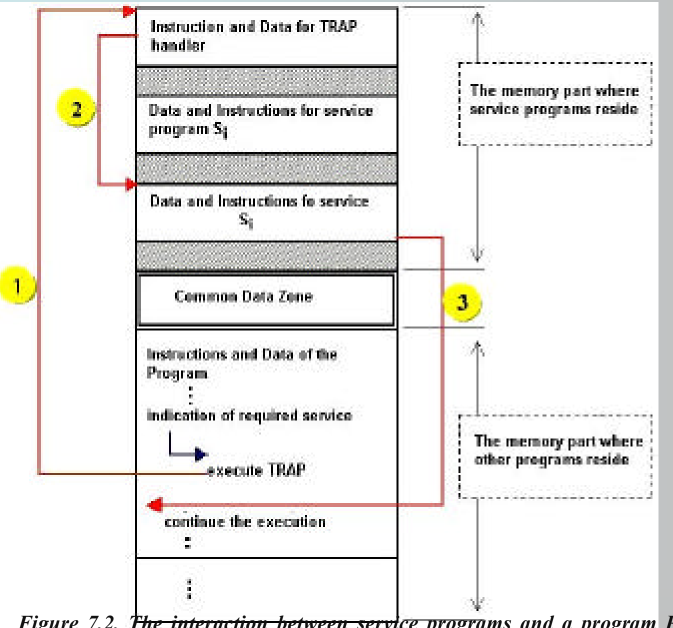

Figure 7.2. The interaction between service programs and a program P that requires a service S_i

Process Description, Control, and Management

In some operating systems there are three levels of concepts used that relate to what is traditionally referred to as a process:

- *Thread*: A dispatch-able unit of work;
- *Process*: A collection of one or more threads and associated system resources (such as memory, opened files, and devices);
- *Session*: A collection of one or more processes associated with a user interface (menus, switchboard, keyboard, display, mouse etc).

Process Management

Process management refers to the full spectrum of OS services to support the orderly administration of a collection of processes .
The processor manager is responsible for creating the environment in which the sequential process executes, including implementing resource management.
The community of processes that exists in the OS at any given time is derived from the initial process that is created when the computer begins operation. The initial process boots up the OS, which, in turn, can create other processes to service interactive users, printers, network connections and so on. A program image is created from a set of source modules and previously compiled library modules in relocate-able form. The *link-editor* combines the various relocate-able object modules to create an absolute program in secondary memory. The loader places the absolute program into the primary memory when a process executes the program. The program image, along with other entities that the process can reference, constitutes *the process address space*. The address space can be stored in different parts of the machine's memory hierarchy during execution.

Process Management

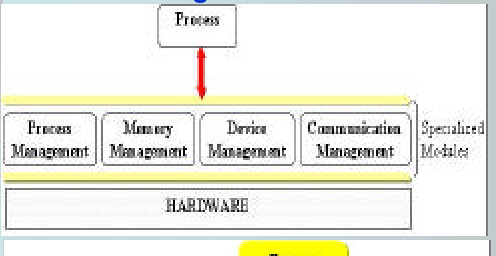

Figure 7.3. Organizing microkernels in groups of modules

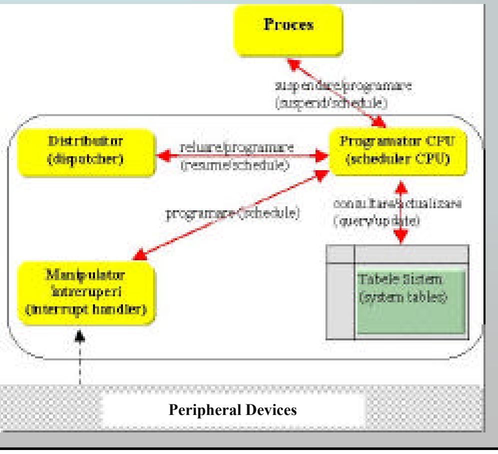

Figure 7.4. Logical Organization of Process Manager

Process Creation and Termination

The life of a process is bounded by its **creation** and **termination**. Between this boundaries and including the boundaries the process can switch into several states (we assume a computer with a single processor), such as:

• **New:** The process that has just been created but has not yet been admitted to the pool of executable processes by the operating system;

• **Ready:** The process is in main memory and available for execution;

• **Ready, suspend:** The process is in secondary memory but is available for execution as soon as it is loaded into main memory;

• **Blocked:** The process is in main memory and awaiting an event;

• **Blocked, suspend:** The process is in secondary memory and awaiting an event;

In anyone of the Blocked states, the process exists, is known to the operating system, and is waiting for an opportunity to execute.

• **Running:** The process that is currently being executed. From time to time, the currently running process will be interrupted and the dispatcher portion of the operating system will select a new process to run.

• **Exit:** A process that has been released from the pool of executable processes by the operating system, either because it halted or because it aborted for some reason.

Process State Transition

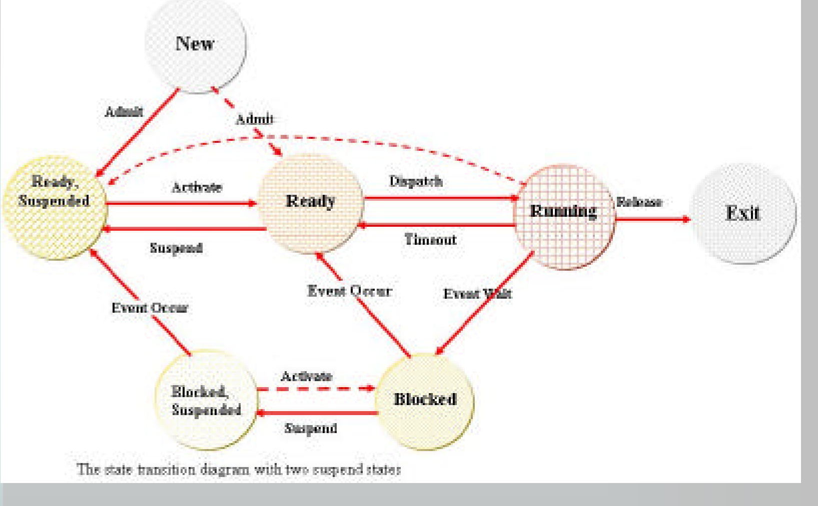

Figure 7.10 The state transition diagram with two suspend states

Process State Transition

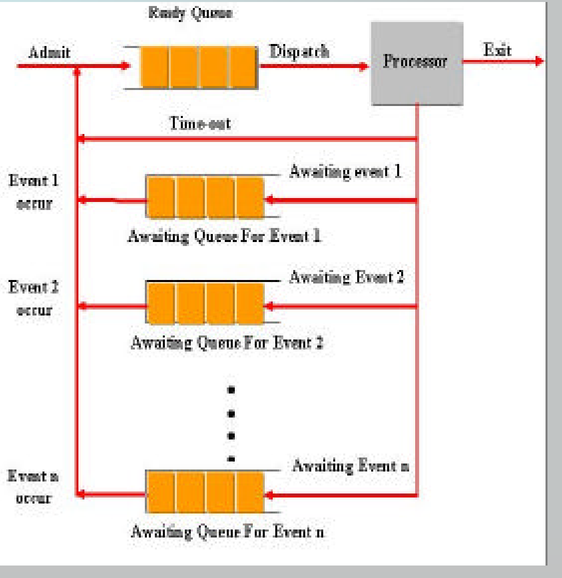

Figure 7.11. The principle of a queuing mechanism with event specialized queue

Contents

Bibliography

1.[Av.00]Avram Vasile - *Sisteme de calcul si operare*, volumul II, Editura Dacia Europa Nova Lugoj, 2003

2. [AvDg.03]Avram Vasile, Dodescu Gheorghe – *Informatics: Computer Hardware and Programming in Visual Basic*, Editura Economica, Bucuresti, 2003, (Chp. 1 pages 46-52; Chp. 3 pages 86-89)

3. Gh. Dodescu, V. Avram: Informatics: Operating Systems and Application Software, Ed. FRQRP LF a, Bucuresti, 2005 (Chp. 6 pages 123-128 and139-142; Chp. 7 pages 147-159)