Operating systems

Script to the lecture

Study: Efcia

INTRODUCTION

This script was created in order to organize the operating system messages. I hope that you will find it helpful in passing the course and that it will also broaden your knowledge.

The script was created on the basis of:

Dr. Juszczyszyn's materials (Chapters 1-4 and the beginning of Chapter 5) Notes prepared on the basis of Dr. Juszczyszyn's lecture
A. Silbershatz, J.L. Peterson, P.B. Galvin, Operating System Basics
A.S. Tannenbaum, distributed operating systems.

I hope you will appreciate this script, because developing this material (which is quite extensive) cost me a lot of time and effort.

Enjoy your reading.

Efcia

CONTENTS

CHAPTER 1 - OPERATIONAL SYSTEM

Operating system (in the simplest way) - a program that mediates between a computer user and hardware.

THE TASKS OF THE OPERATING SYSTEM

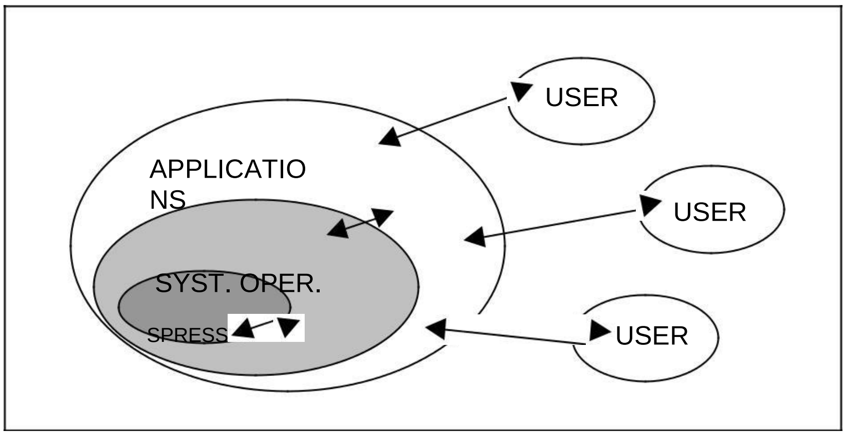
- Hides hardware details of a computer system by creating abstracts (virtual machines). examples:
    - uniform access to external facilities
    - sets of disk blocks seen as files with symbolic names
    - large, fast, dedicated memory
    - simultaneous execution of programs (as a parallel abstraction)

- Resource management:
    - resources are "objects" necessary to execute the program, e.g. memory, CPU time, I/O, communication ports (everything that represents the functional value for the system)
    - resource allocation and recovery strategies (memory management, processor management, file management, device management)
    - the effectiveness of resource management determines the efficient use of computer hardware

- Provides a "friendly" interface
    - convenience of use (setting of switches, perforated cards, perforated tapes, graphic terminals with mouse and keyboard)

THE COMPONENTS OF THE COMPUTERISED SYSTEM

- Hardware - basic computing resources (CPU, memory, I/O devices)



- Operating system - supervises and coordinates the use of equipment
- Application programs - define the way of using system resources to solve tasks set by users
- Users (people, machines, other computers, external programs)

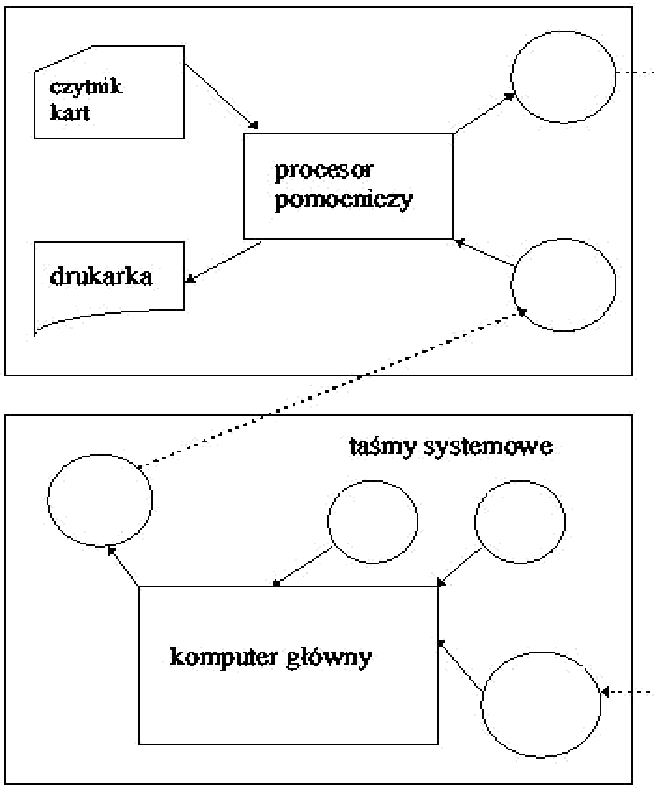
HISTORY OF OPERATING SYSTEMS

EARLY OPERATING SYSTEMS - "NAKED EQUIPMENT"

- Structure:
  - Large, console-operated machines
  - for one user (need for work schedules, etc.) o the programmer/user acted as the operator
  - inefficient use of expensive resources o low CPU utilization o low CPU utilization
  - full sequencing of device operation
  - equipment downtime related to the performance of operator activities
- Early software
  - assemblies, loading programs, connecting programs, libraries of typical functions, compilers, device control programs

BATCH SYSTEMS

- Employment of the operator (user and operator)
- Reduce the time to install a task by preparing a batch of tasks with similar requirements
- Automatic job scheduling - automatic transfer of control from one job to another
- Residential monitor:
  - control initially belongs to the monitor o control is transferred to the task o control is transferred to the task o control is transferred to the task o control is transferred to the task o control is transferred to the task o control is transferred to the task o control is transferred to the task o control is transferred to the task o control is transferred to the monitor o control is transferred to the task o control is transferred to the task o control is transferred to the task o control is transferred to the task o control is transferred to the task o control is transferred to the task o control is transferred is transferred to the task o control is transferred to the task o control is transferred to the task to the monitor.
  - When the task is completed, the control system returns to the monitor.
- Entering the Job Control Language
  - Significant change in operating mode from the user's point of view
  - Increased system throughput at the expense of average job rotation time
  - Problem: low performance (CPU and I/O devices cannot work simultaneously, card reader very slow) Solution: indirect (off-line) operation
- The use of magnetic tape readers:
  - Accelerate calculations by loading jobs from tapes to memory and reading cards and printing results off-line
  - The main computer is not limited by the speed of card readers and printers, but only by the speed of faster tape stations.
  - No changes are required to the application programs when switching to indirect mode
  - Multiple reader-to-belt and ribbon-to-printer systems can be used for a single CPU

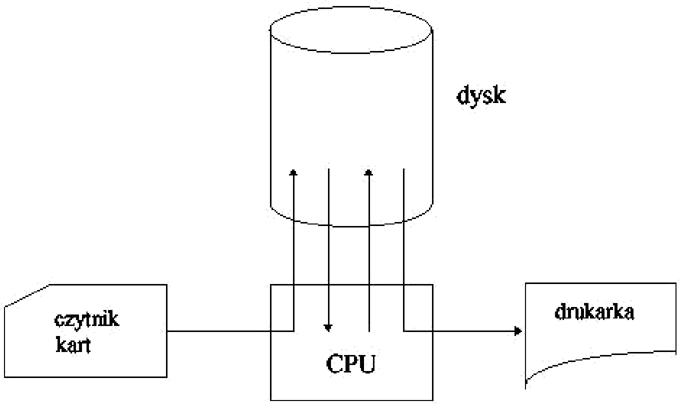


BUFORMANCE

- Simultaneous calculation and I/O method for a single task: o Does not completely eliminate CPU or I/O downtime
  o Requires memory to be allocated to system buffers o Eliminates data processing time fluctuations o Eliminates data processing time fluctuations

SPOOLING <u>(simultaneous peripheral operation on-line)</u>

- Method of simultaneous execution of one task and calculations for other tasks
  - During one task, the operating system reads another task from the card reader to the disk (task queue) and e.g. prints the results of the previous task placed on the disk.
  - Task pool - possibility to select the next task to be performed
  - Further development of the operating system (load module, control module, discharge module)



TIME-SHARE SYSTEMS

- The processor performs many different tasks alternately, with switching so frequent that users can interact with the program while it is running.
  - User interaction with a computer system (an interactive task consists of many short actions)
  - File system directly accessible (access to programs and data)
  - Job exchange between memory and disk (swapping)

EQUILIBRIUM SYSTEMS

- Multi-processor systems with more than one CPU.
  - Systems - processors divide memory and clock; communication usually through shared memory
  - Benefits: increased throughput, cost effectiveness, increased reliability
- Symmetric multiprocessivity (each processor makes an identical copy of the operating system, many processes can be performed simultaneously, without any decrease in performance).
- Asymmetric multiprocessivity (each processor performs its assigned task; the main processor ranks and assigns work to subprocessors - a solution used in large systems).
- Possible dispersion of multiprocessor systems.

REAL-TIME SYSTEMS

- Strict time constraints
- Feedback to user/equipment actions.
- Examples: industrial process control, patient health monitoring....

<u>Equipment development</u>: (four generations): lamps, transistors, small scale integration integrated circuits, large scale integration integrated circuits

HARDWARE REQUIREMENTS FOR MODERN SYSTEMS

- interrupt system (interrupt tables, during interrupt execution other tables are disabled, or there is an interrupt priority).
- Direct memory access (DMA) for high-speed I/O devices
- memory protection (I/O operations only via s.r.o., interrupt board and system protection, mutual task protection, e.g. base and boundary registers)
- multi-state processor
- privileged orders
- clock

GENERATIONS OF OPERATING SYSTEMS

- 1940s Lack of operating systems in the present sense of the word
- The 50s The 50s The possibility of batch processing
- Early 1960s Multi-programming, Multi-processing, Device Independence, Time Division, Real-time Processing.

- Mid-1960s, mid-1970s. General-purpose systems with different operating modes at the same time
- Currently: Personal computers, workstations, database systems, computer networks, distributed systems

THE OVERALL STRUCTURE OF THE OPERATIONAL SYSTEM

FUNCTIONAL MODULES OF THE OPERATING SYSTEM

- memory management
- processor management (at the level of tasks and processes - available operations on processes: create, terminate, abort, signal, allocate/free memory)
- device management (attach/detach operations, request/release, read/write)
- information management (file system - operations: create, delete, open, close, change attributes)
- nuclear (inter alia, ensuring communication between the components of the system and security)

THE SIMPLEST STRUCTURE (MONOLITHICNT S. O.)

- a program composed of many procedures, each can call any limited structurealization (e.g. OS/360, MS-DOS, some Unix versions)

macro nucleus - all SO functions placed in the kernel and executed in the privileged mode (e.g. OS/360)

kernel - part of the function transferred from kernel to user level (e.g. in Unix: kernel and system programs)

LAYERED STRUCTURE

Each layer performs functions depending only on the layer below (T.H.E., RC-4000, VMS); in Multics a variety of layer structure - a set of concentric rings. The macro-nucleus used - all layers are privileged. Layer structure (T.H.E. system):

- level 5: user programmes
- level 4: buffering of I/O devices
- level 3: operator console support program
- level 2: memory management
- Level 1: CPU allocation planning
- level 0: equipment

THE STRUCTURE OF THE MICRONUCLEI

Micro-nucleus includes only process creation, inter-process communication, mechanisms (but not strategies!), memory, processor and device management (at the lowest level), e.g. Mach, QNX (industrial systems in general), etc.

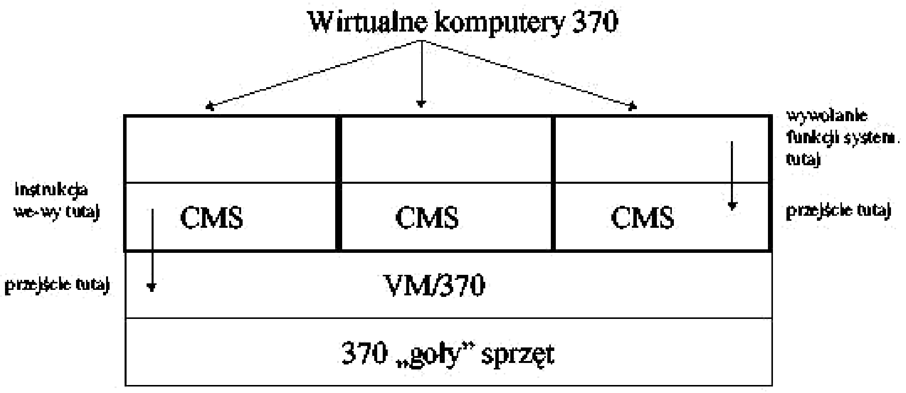
CLIENT-SERVER STRUCTURE

- Modules perform separate tasks, they communicate by sending messages. 8

- Server: provides a service, client: requests a service. Simple adaptation to distributed systems.
- Necessary connectionless Q&A protocol required
- the kernel can be reduced to 2 references to the system (transmit/receive message)
- easy implementation of dispersion

THE CONCEPT OF A VIRTUAL MACHINE

- each process receives a (virtual) copy of the computer being the basis of the system; each virtual machine is identical to the real hardware, so each can run any operating system; e.g. VM for IBM's large computers
- Benefits: every part of the system is much smaller, more flexible and easier to maintain.



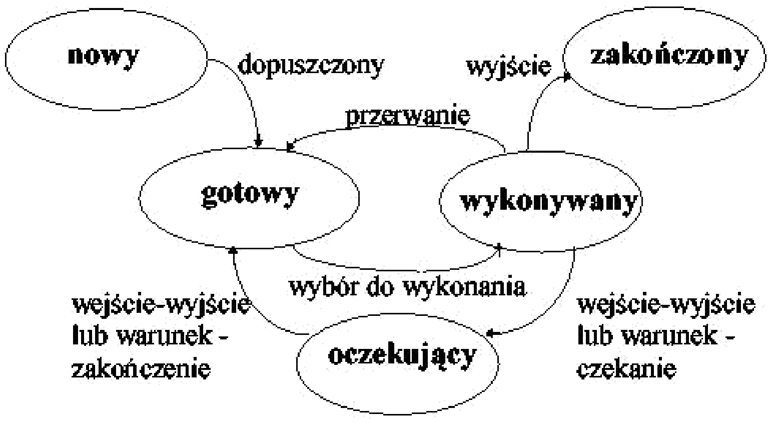
CMS - operating system for individual user

CHAPTER 2 - PROCESSES AND THREADS

PROCESS

Process - program (binary code) during execution; execution is sequential (exception: multithreaded processes (see below).

- Processes are performed concurrently (but not necessarily in parallel - time division)
- Process states:
  - new: the process has been created
  - Implemented: process instructions are followed
  - Pending: the process waits for the event to occur
  - Ready: the process is waiting for the processor to be assigned
  - completed: the process has ended



REPRESENTATION OF THE PROCESS IN THE SYSTEM (CONTENT OF THE PROCESS CONTROL BLOCK)

- state of the process
- identifier (unique number - in UNIX - PID)
- Command counter
- processor registers
- indicator to the process queue
- information about the memory occupied by the process
- a list of open (used) files

PROCESS CREATION

- using the system function (e.g. fork() in UNIX)
- the process created by another process (parent, parent, etc.) is called a child.
- the offspring obtains a new pool of resources or uses the parent's resources (less system overload).

CLOSURE OF THE PROCESS

- after the last instruction.
- a special function (e.g. exit() in UNIX)
- at the request of the parent (abort())
- sometimes: cascade ending: when the parent is finished, the posterity processes are completed.

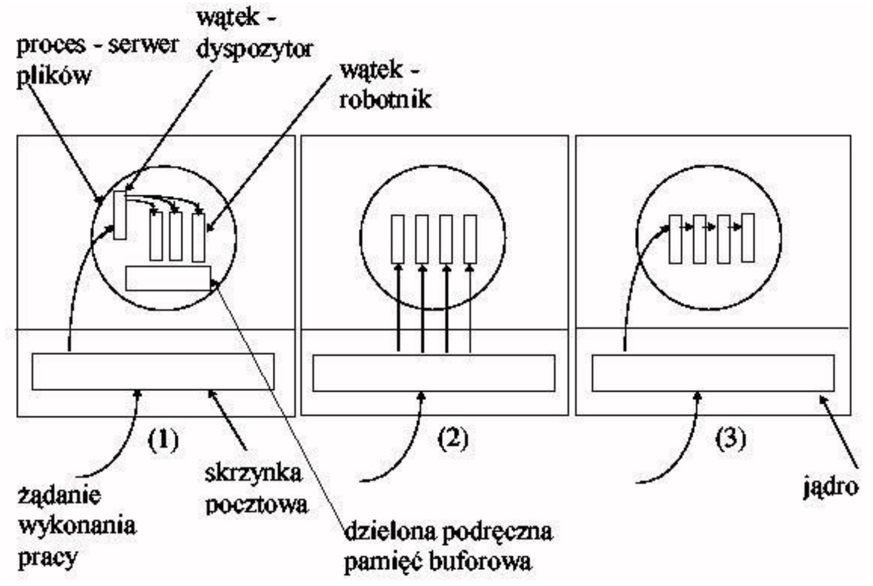
MOTHER

<u>Control thread (so-called light process):</u> a sequence of instructions; traditionally process = one control thread, but sometimes (in some systems) in the process address space allows more control threads to be executed concurrently.

- the thread has the same basic features as the process (states, ability to create child threads, use external devices, etc...)
- there is no memory protection between the threads of the same process.
- analogy: machine processes == process threads
- elements of the thread and process:

Weft	The process
Command counter	address space
pyre	open files, semaphores.
processor registers	global variables
a list of descendants' threads	list of percentages of posterity

- Traditional (heavy) process is equivalent to a task with one thread
- If a task has multiple threads, another thread in the task can be executed while one thread is blocked; if multiple threads work together in one task, you can increase bandwidth and improve performance
- Threads allow parallelism to be combined with sequential execution:
  (1) dispatcher-employee model
  (2) Team model
  (3) stream model



STATIC/DYNAMIC THREADS

The structure of the process threads is fixed (in the process source code) or the process can freely create/delete its threads

IMPLEMENTATION OF THREADS IN THE OPERATING SYSTEM

- A package of threads : a set of elementary actions on threads available in the system (e.g. library procedures).
- Implementation of threads in the user space:
   about the testicle doesn't know about the threads, sees only one threaded processes.
   - o Advantage1: you can use threads on a system that does not implement them (e.g. originally UNIX)
   - o possible fast switching of threads - only reloading of the stack, instructions and registers (the fastest actions in the computer system).
   - o Advantage2:Each process can use its own algae scheduling for its own threads.
   - o Problem1: If you block thread references to the system - the process cannot give up system control, it has to wait for its threads. Use a check code (and. jacket) to see if thread cancellations will block. and threads are used mainly in tasks with cancelling cancellations, where they are supposed to improve performance...
   - o problem2: there is no expropriation, the threads must control the process execution procedure themselves.
- Implementation of threads in the nucleus:
   - o the executive system is part of the kernel
   - o the kernel assumes an array of threads for each process
   - o all functions that can be blocked are in the form of references to the system o when the thread of a czka, the kernel selects the next one - performance!
   - o Defect: cost of appeals to the system (time!)
- General problems (most difficult to implement) with threads: o handling interrupts (signals)
   - o non-renewable system procedures

CHAPTER 3 - PROCESSES - PLANNING TASK

PLANNING

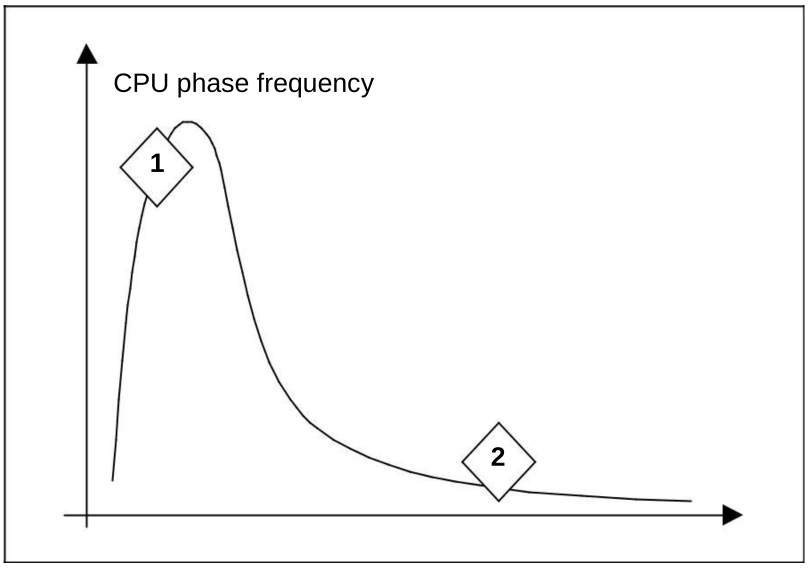
Planning - allocation of a processor to ensure optimal use of the processor (e.g. when the process waits for an I/O device; the control goes to the next one).

- processes are waiting for processor allocation in queues (queue: list of process control blocks)
  - o Queue of tasks: newly created and memorized processes
  - o queue of finished processes: waiting for processor allocation
  - o Equipment queues: processes waiting for equipment allocation
- the scheduler (scheduler - system process) is responsible for scheduling (selection from queues) of processes o the long-term planner: he selects processes from the task queue (reduced in some processes) o the long-term planner: he selects processes from the task queue (reduced in some processes) o the processes from the task queue (reduced in others) o the processes from the task queue.
  systems); it operates every second of every second.
  - o Short-term planner: selects from the queue the pr. ready; works every ms.
  - o time the scheduler is responsible for swapping, i.e. temporarily deleting a task completely from the main memory to the auxiliary memory
- The decision is made when the trial:
  1. changes from a executing state to a waiting state
  2. moves from a state of execution to a state of readiness
  3. moves from a waiting state to a ready state
  4. ends
- The control to the process selected by the planner is given by the dispatcher:
  - o stores the state (context) of the current process.
  - o Switches the context (registers, etc...)
  - o Switches the system into user mode
  - o Jump to the address from the control block

  the delay requested by the dispatcher: 1-100ms (dependent on hardware support)

Process life: two phases (cyclic): 1. processor, 2. we-wy (device waiting)

- I/O-oriented processes (1): spend more time performing I/O operations than calculations; many short CPU demand periods
- processor-oriented (2): spend more time doing calculations; several very long CPU demand periods



Processor phase length

PLANNING CRITERIA (DIFFERENT OPTIONS - MAXIMISATION AND MINIMISATION)

- max processor utilization (preferably 40-90%)
- Max bandwidth - number of processes completed in one time)
- minutes of the processing time of the process (from creation to completion)
- minutes of waiting time in queues (this criterion will be used)
- minutes of the process response time (in interactive systems)

we will minimize the average waiting time in queues

PLANNING ALGORITHMS

FCFS (First Come First Served)

- time allocation in order of submission of processes
- simplest implementation (FIFO queue)
- no expropriation
- troublesome in time-share systems
- an example:
  - Processes and their CPU needs: P1 (24), P2 (3), P3 (3)
  - If processes have arrived in the order P1, P2, P3: Waiting time: P1 = 0, P2 = 24, P3 = 27 W Average waiting time: (0 + 24 + 27)/3 = 17.
  - If processes have arrived in sequence P2, P3, P1: Waiting time: P1 = 6, P2 = 0, P3 = 3 W average waiting time: (6 + 0 + 3)/3 = 3
- effect: short processes are held back by long processes
- advantages: fair, low system surcharge
- disadvantages: long average waiting time and variance of waiting time, unacceptable in time-share systems

SJF (Shortest Job First)

- An algorithm binds to each process the length of its nearest CPU phase. The processor is assigned to the shortest processor.
- With equal CPU phases, we have a FCFS
- SJF is optimal (proof: placing a short before a long one reduces the short waiting time more than increasing the long one)
- SJF's fine:
  - non-expropriatory
  - expropriating (when the length of the new phase is shorter than what is left of the currently executed process)
- Problem: How to estimate the length of future CPU phase?
  - the long-term planner may require his declared processes (tasks are to predefined phase time); short-term cannot (long delay)
  - frequent solutions: length of the next phase (tn+1) = exponential mean of the length n of the previous phases (assumption: length of the phase depends on the length of the previous phases):

Oh, yeah.$_{n+1}$ $= \sum_{i=0}^{n} a(1-a)^{i} t_{n-i}$ where $a<1$.

- the above solutions are adaptive (adapts when the demand of the process changes)

PRIORITY PLANNING

- a special case is SJF (where priority = 1/(processor phase length)).
- usually the priority is determined by an integer, e.g. between [0.7] or [0.4096] - lower value = higher priority.
- problem: a low-priority process may never happen (in one of the industrial sectors) IBM systems, the process was waiting for execution from 1967 to 1973...); solution: process aging (reducing the priority by 1 e.g. every 10 min.); solution: process aging (reducing the priority by 1 e.g. every 10 min.)
- may be expropriatory (but not necessarily)
- to define a priority:
  - externally (via system function)
  - internally by a declaration of the process itself (based on e.g. requirements: memory, processor, etc...)

ROTATIONAL PLANNING (Round Robin - RR)

- time quantum is determined (~10-100 ms)
- the process queue is treated as a cyclical FIFO
- algorithm browses the queue and allocates each time quantum in turn (if the process does not finish in it - returns to the queue, and the length of its processor phase is reduced by the time quantum).
- the algorithm is expropriating
- when there is N processes and the time quantum is Q, the maximum waiting time can be (N-1)Q
- the effects:
  - when Q grows infinitely; FCFS Rotational Planning
  - when Q is going to 0 there is a division of the processor - each process has a processor with a speed of 1/N real time.

- by the way: a similar effect is given by so-called peripheral processors - with e.g. 10 sets of registers; for each of them a task. The peripheral processor executes 1 command per task. Profit: the processor is faster than the memory - the whole thing is not 10x slower...)
- Performance aspects:
  - long time quantum is indicated in comparison with context switching (wpp poor performance)
  - experimental rule: the time quantum should be longer than 80% of the CPU phases for optimal performance.

THE STRUCTURE OF QUEUES IN THE OPERATING SYSTEM (MULTI-LEVEL PLANNING)

- The queue of finished processes is divided into separate queues; for example (in the simplest way):
  - first plan processes (system, interactive)
  - secondary processes (batch processes)

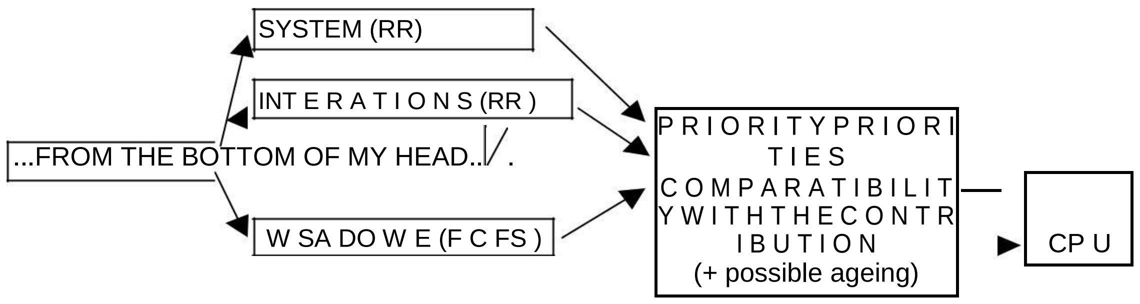
- Each queue has its own scheduling algorithm
- Example:
  - First Planning Processes - Carousel Strategy (RR)
  - secondary processes - FIFO

- Ranking between the queues:
  - Established, e.g. handle all foreground processes first, then background processes; possibility of starvation
  - Time quantum: each queue receives a fixed amount of CPU time to be shared between its processes, e.g. 80% for RR foreground, 20% for FCFS foreground, 20% for FCFS foreground.
  - Processes can move between queues

- Parameters of the system planner:
  - number of queues
  - the scheduling algorithm for each queue
  - process promotion/degradation method (to a queue with a lower/higher priority respectively)
  - queue selection method for a new process

- Example: a process that consumes a lot of CPU time (e.g. all the quantum in RR) goes to the queue about

  a lower priority, but a longer quantum. At the lowest level: FCFS. Interaction and I/O oriented processes will remain in higher priority queues and computationally oriented processes will remain in lower priority queues.


SYSTEM (RR)
INT E R A T I O N S (RR)
...FROM THE BOTTOM OF MY HEAD..
W SA DO W E (F C FS)
P R I O R I T Y P R I O R I T I E S C O M P A R A T I B I L I T Y W I T H T H E C O N T R I B U T I O N
(+ possible ageing)
CP U

- Conclusions from simulation, theory, practice:
  - the phase distribution is generally exponential in real systems
  - if average queue length = n, average handling time = T, new process arrival rate = λ, then n = λT (tw. Little; applies generally to stable queue systems)

- Multi-processor systems:
  - splitting (a separate queue and algorithm for each processor) or...
  - ...a round of drinks together:
    - each processor selects the process itself
    - one processor assigns processes to processors
    - (so-called asymmetrical multi-processing)

CHAPTER 4 - COORDINATION OF PROCESSES

CRITICAL SECTIONS

- concurrent processing expansion for a single CPU: the process starts before the previous one is finished
- possible conflict in operations on shared data (e.g. even the instruction x:=x=1 is 3 instructions (mov AX,x; inc AX; mov x,AX) of the processor...
- Conclusion: each process has (can have) a code segment called the critical section (SK). SK processes are subject to mutual exclusion over time.
- Features of SK:
  - reciprocal exemption SK
  - limited (finished) waiting for entry to the SK
  - progress: only processes willing to join SK are candidates
- e.g. (general process structure and example solution):

```
repeat
     while x<>0 to nothing (input section)
     <<SKY.>>  (SK)
     x:=1       (output section)
<rest of the process code>
until false
```

(imposed alternate (i.e. not fulfilling the progress condition) SK execution for 2(there may be more) processes)

- Correct solution for 2 processes:

```
var  Flag: array[o..1] of Boolean
     number: 0...1.
repeat
     flag[i]:=TRUE          (I want to come in)
     number:=j              (assumption that the other one wants to come in)
     while (flag[j] AND number:=j)to nothing
          <<SKY.>>
     flag[i]:=FALSE
<rest of the process code>
until false
```

(without the variable number it would be possible to set both flags by processes in two consecutive processor instructions (after unlucky context switching) and their permanent looping...)

SEMAFORY

- General semaphore popular solution for SK and synchronization problems
- Semaphore s is a total variable available after the initialization with only 2 operations:
    - o wait(s):`while s <= 0`$_{s\ :=\ s+1;}$ `to nothing; s := s-1;`
    - o Signal(s):

- Interpretation s: amount of free resources; in this case the resource is SK
- Operations on the semaphores are NOT SUBJECT! Implementation:
    - o on a single CPU - blocking interrupts for the duration of the operation
    - o in a multiprocessor system TEST&SET CPU manual

- Usage:

```
s:=1

repeat

      Wait a minute.

      <<SKY.>>

      Signal(s)

<rest of the process code>

until false
```

- Synchronization application (e.g. when service1 must be executed after service2): s:=0

<u>1 proce</u>ss:	Op1;	<u>Proces</u>s 2:	Wait a minute;
	Signal(s);		Op2;

- Disadvantage: s requires active waiting (CPU time!!!!)
    - o solution: after waiting(s) the process is blocked and placed in the queue related to s
    - o downloading the process from the queue is done after signalling(s)
    - o then the semaphore is the record (integer variable + list of processes)
    - o the queue algorithm is not relevant to the processor
- Realisation of indivisibility:
    - o 1 processor: blockade of interrupts for the duration of the operation
    - o multiple processors: e.g. a single "test&set" processor manual

LOCKS AND BOLTS

- Lock state: each process in a set of processes waits for an event that can be caused only by another process from the same set (an event can be e.g. allocation or release of a resource).

e.g:

Semaphores A and B have a value of 1:

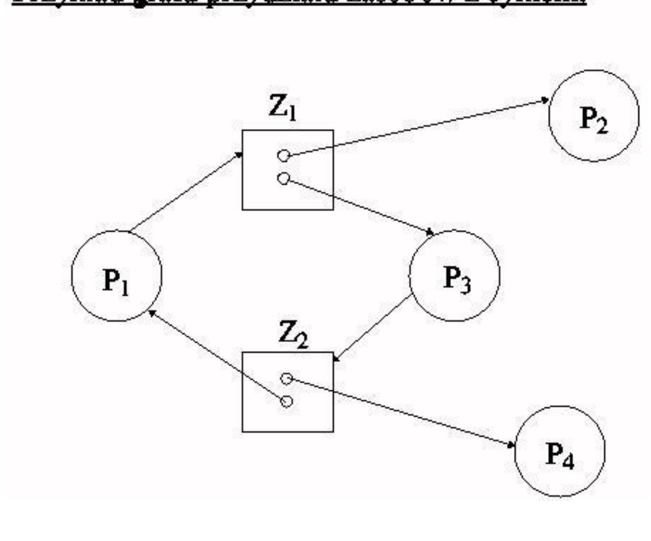
P0: wait(A); wait(B)

P1: wait(B); wait(A)

- A special case: so called starvation - waiting infinitely under the semaphore - e.g. when we use the LIFO queue of the semaphore.
- Conditions for a blockade:
   1. Mutual disconnection (at least 1 resource must be indivisible)
   2. Holding and waiting (a process that holds at least one resource waits for the allocation of additional resources held by other processes).
   3. There is no expropriation of resources.
   4. Cyclic waiting (there is a set of waiting processes {P0, P1, Pn }, such that P 0 waits for a resource held by P1 , P1 waits for a resource held by P2 , ..., P n waits for a resource held by P0 ..., P n waits for a resource held by P0 .
      (4 implies 2, so the conditions given are not independent)

- Description of the blockade:
   - set of P processes, Z resources, "resource allocation graph" - bipartite (edges connect vertices belonging to 2 separate sets - in this case resources and processes) directed graph.
   - edges Pi Zj : edge of the order
   - edges of Zj Pi : edge of allocation
- Drawing:
   - the trial's going on over and over again,
   - resource from a rectangle
   - 1 copy of the resource with a dot in a rectangle.
   - the edge of the ration starts with a dot
- Events in the system:
   - order with p : add edge with p with
   - Order processing: add edge with p, remove edge with p with
   - releasing the resource: we remove the edge

Przykład grafu przydziału zasobów z cyklem:



- Lockdown:
  - if there's no cycle in the graph: there's no lock.
  - if there is a cycle and the resources are single, it is a blockade
  - if there is a cycle and the resources are multiple, it could be a blockade
- Handling blockades:
  - ensure that they never happen (copy of the resource allocation protocol)
  - allow access to the blockade and then remove it
  - ignore and assume that they will not occur (e.g. UNIX and most popular s.o.s.).

PREVENTION AND AVOIDANCE OF BLOCKADES

- Prevent one of the four conditions for blockage from being met:
  - mutual only: generally impossible; some resources are by definition indivisible (printer, writing file, etc.)
  - Storage - it must be ensured that when a process demands a resource, it does not have other resources: allow for ordering only when it releases all of its resources (problem: starvation and inefficient use of resources).
  - You can require a process to order and receive all its resources before it starts operations or request resources when it has no other resources - low resource utilization, starvation possibility, etc.
  - No expropriation: if a process that owns resources requires a resource that cannot be allocated immediately, it must release all resources - Expropriated resources are added to the list of resources that the process waits for
  - Cyclic waiting: a complete ordering of all types of resources is imposed and requires the process to order resources in an increasing order of their numbers (the method excludes the formation of cycles).

    the above methods always limit the bandwidth of the system...

- Avoiding blockades:
  - Requires the system to have some information about future resource needs
  - The simplest model requires the process to declare the maximum number of resources of each type it will need
  - Lock avoidance algorithm dynamically examines the status of resource allocation to ensure that there is never a cyclical expectation
  - The state of the system is determined by the number of available and allocated resources and the maximum demand for processes.
- Safe status - when the process requests an available resource, the system must determine whether the immediate resource allocation will maintain the safe status of the system
- The system is in a safe state when there is a safe sequence <P1 , P2 , ..., Pn > : safe, if for each P and its potential demand for resources can be satisfied by the currently available resources and the resources held by the Pk processes, where k < i.
  - If the system is in a safe state, there is no lock.
  - If the system is not in a safe state, there is a possibility of locking. Locks can be avoided by ensuring that the system never enters a hazardous state.
- Avoidance algorithm using the resource allocation graph: (for individual resources):
  - enter the edges of the declaration (demand) - drawn with a dashed line)
  - we are looking for loops in the graph (complexity O(n2))
  - if there is a loop, we do not allocate the resource.

 Avoidance algorithm (for multiple resources - the so-called algae banker. Lt. A Silbershatz et al.
Basics of système. operational chapter 6.4.1 p. 208.):
- Each process must a priori submit a maximum demand for resources
- A process claiming a resource may have to wait even though the resource is available... o When a process gets all the resources it needs, it will return them in a finite time
- After ordering through the resource process, the system determines whether the allocation of resources leads to a safe state.

Assumptions:

n - number of processes

m - number of resource types

int Available[m] - number of resources of a specific type (e.g. Available[3]=5 is 5 resources of type 3) int Max[n][m] - maximum process requests int Allocated[n][m] - allocated resources

int Potrzebne[n][m] - the resources needed (except that Potrzebne[i][j]=Max[i][j][i][j]) int Zamówienia[n][m] - current process orders.

Algorithm:

1. If Orders[i] <= Needed[i] then perform step 2. Wpp error - the process exceeded the declared request.
2. If Orders[i] <= Available follow step 3. Wpp process and must wait.
3. The system tries to allocate the required resources to the process and changes the system state as follows:

   Available = Available - Orders[i] Allocated[i] =
   Allocated[i] + Orders[i] + Orders[i]

   Needed = Needed - Orders

If the state after the change is safe, the transaction is carried out. If not, the trial, and he has to wait...

Security algorithm:

1. int Praca[m]; int Finish[n]
2. Work = Available; End[i] = FALSE for all and
3. We find both End[i]=FALSE and Need[i] <= Work; If there is no such and to step 5.
4. Work = Work + Allocated; End = TEU; go to step 2.
5. If End[i] = TRUE for all i, then the system is in a safe state...

Cost of determining whether the scheme is free = m x n2

DETECTION AND EXIT FROM THE BLOCKADE

- In systems without prevention and avoidance must be:
  - algae block detection (most often: searching for cycles in the graph)
  - algae exit from the blockade
- When triggering a blockade detection algorithm:
  - when the resource cannot be allocated immediately
  - once for a set period of time (e.g. 10 min)
  - when CPU load drops below ~40% (blockage throttles system bandwidth)
- Exiting the lock (ways):
  - Operator notification (solves the problem manually)
  - Removal of process(es) involved in the blockade

    the entire loop (significant cost, loss of partial results) in

    turn (calling the cycle search after each deletion)
  - different criteria for victim selection (priority, type, resources available, etc.)
  - Expropriation from resources (guarantee that there will be no starvation is needed - e.g. expropriation possible max. k times)
- Generally (lock service):
  - prevention, avoidance, detection, detection, disposal
- In modern systems: - the division of resources into classes:
  - we apply cycle prevention to classes
  - within the classroom:

    - internal resources (counterblocks, etc...) ordering of resources

    - lead memory

    - device and files to avoid blocking

CHAPTER 5 - MEMORY MANAGEMENT

PREMISE

- the memory contains the entire program (with exceptions below)
- Memory is an array of addressed words
- cooperation with memory involves writing to/reading from/to the following address
- processes are downloaded to memory from the task queue.

VARIOUS MECHANISMS

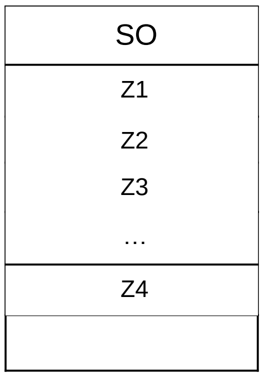
- Binding:
  - the process can reside in <u>any</u> part of the memory; there must be a mechanism to bind commands and data to physical addresses.
  - bonding can occur in time:
    - <u>compilation</u> (assuming that the process starts with the X address - e.g. *.com files in dos)
    - <u>loading</u> (after moving the code it is necessary to reload it)
    - <u>execution</u> (possible displacements)
- Dynamic loading: subprograms are on disk in a removable form and are loaded after
  Advantage: we do not load unused code.
- Dynamic linking (without it, all programs must have their own copies of system libraries (e.g. DLL). There are tabs in the references places (small fr. code, after the call giving the address of the library procedure)
- Overlays (e.g. *.ovl) - only the necessary code is stored in the memory, overlays turn off (usually), are stored on the disk in the form of an absolute memory image.

because the processes are performed only in memory, not always all of them fit in. Well, then:

CHANGE

- The process can be temporarily sent from the main memory to the external memory, and after some time it can be brought back to the main memory.
- The program returns to the same place, unless we are dealing with a binding at the time of execution.
- As an external memory for exchange, a large high-speed drive with direct access is used.
- Main mark-up: transmission time; for good performance, the execution time must be long compared to the switching time.
- The system needs to know about the memory requirements in order to work effectively, the allocation is made by system functions.
- I/O operations (operating on memory buffers) cannot occur during the exchange, no I/O processes are exchanged and the system takes care of the buffers.

MEMORY MANAGEMENT



The operating system stores an array of information about which parts of the memory are available and which are occupied. At the beginning, all memory is available for use by the user processes and is treated as one big block of memory - a hole. When a process arrives with a need for memory, a sufficiently large hole is searched for for it. If found, it allocates memory only in the necessary amount, leaving the rest for future needs.

Incoming processes are placed in the input queue. When allocating memory to processes, the operating system takes into account the memory requirements of each process and the amount of free memory. The process, which has been assigned space, is stored in memory and begins to compete for the allocation of the processor. When a process finishes its work, it releases its memory and the operating system can then place the image of another process in the memory from the input queue.

At any time we have a list of sizes of available blocks and the entrance railway. The operating system can organize the input queue according to the planning algorithm. Processes receive memory allocations until it is clear that the memory requirements of the subsequent process cannot be met. No available memory block (hole) is large enough to accommodate this process. The operating system can then wait for a large enough block to appear, or it can jump over to the output queue to see if it can satisfy a lower memory requirement of any of the other processes.

In general, a collection of holes of different sizes is scattered throughout the entire memory at any time. When a process comes in and orders memory, the system reviews the collection for a hole large enough for the process. If the hole is too big, it is divided into two: one part is assigned to the incoming process, the other part goes back to the set of holes. When the process is finished, the process releases its memory block, which is again placed in a set of holes. If a new hole is adjacent to other holes, the adjacent holes are connected to form a single, larger hole. It is then necessary to check whether any processes are waiting for memory and whether the newly recovered and reorganized memory can meet the requirements of any of these waiting processes.

This is a particular case of the general problem of dynamic memory allocation, which is how to solve the problem of how to fulfill an order for an area of n size on the basis of a list of free holes. There are many solutions to this problem. A set of holes is viewed to determine which of the following they're the best qualified to be assigned. The best-known strategies for selecting a free area from a set of available holes are

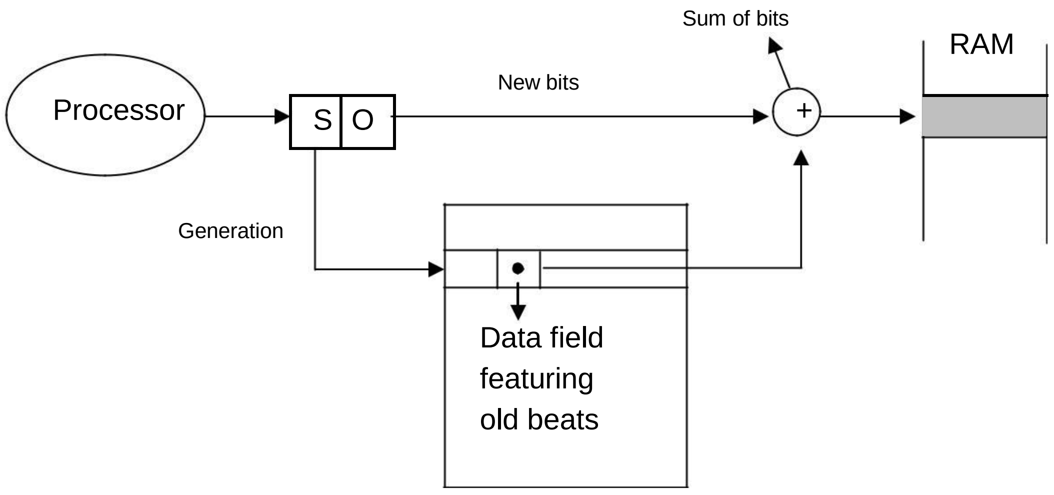
1) First fit - the first hole of sufficient size is allocated.
2) Best fit - the smallest hole is allocated from sufficiently large holes; this strategy ensures the smallest allocation residues.
3) Worst fit - the largest hole is allocated; this strategy leaves the largest hole after allocation, which may be more useful than the remainder of the best fit approach.

Strategies for selecting the first or most suitable hole are better than selecting the largest hole in terms of both time and memory usage. Neither the allocation of first holes nor the best matching holes is clearly better in terms of memory usage, but first fit allocations are generally faster.

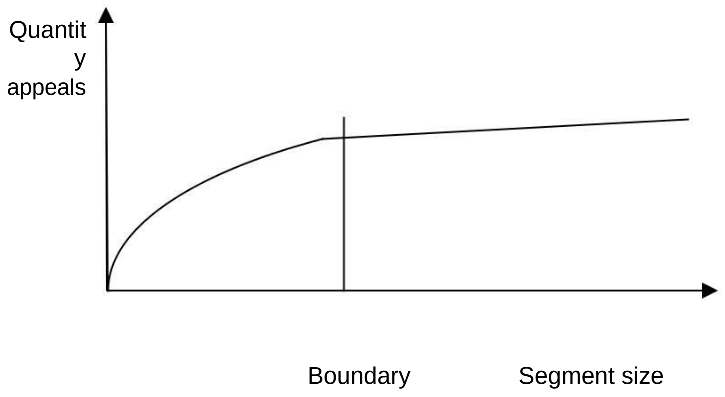
MEMORY SEGMENTATION

Segmentation is a memory management scheme that helps to make the described way of seeing memory by the user a reality. The logical address space is a set of segments. Each segment has a name and a length. The addresses specify both the name of the segmentation and the distance inside the segment. The user defines each address by two sizes: segment name and distance.

To facilitate implementation, segments are numbered and segment numbers are used in references to segments instead of segment names. The logical address creates a pair of <number-segment, distance>.



What should the segments be?



The code and data are worth dividing into
small segments.

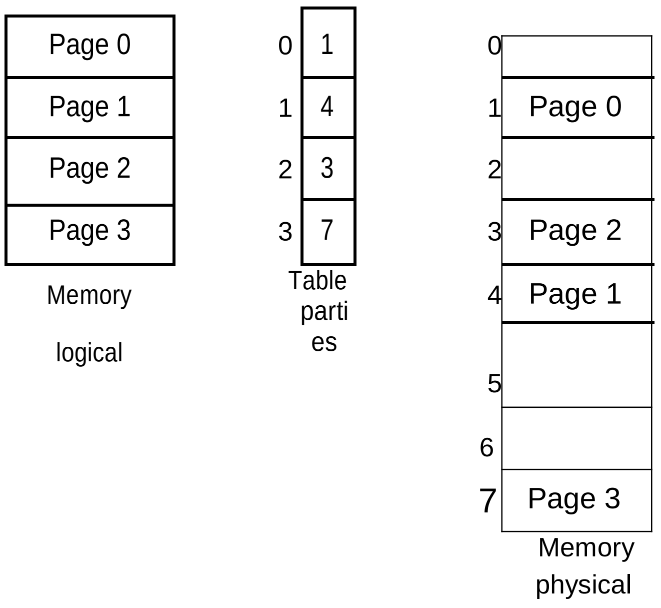
PARTIES

Paging is a memory management scheme that allows for discontinuity of the physical address space of the process. When using paging, you avoid the problem of matching variable-size pieces of memory to the space in the auxiliary memory. Various forms of paging are commonly used in most systems.

Physical memory is divided into fixed-length blocks called frames. The logical memory is also divided into blocks of the same size called pages. When a process is to be executed, the process is to be placed in auxiliary memory on the side of the process, in any RAM frame. Auxiliary memory is divided into fixed length blocks of the same size as frames in RAM.

Each address generated by the processor is divided into two parts: the page number and the distance on the page. The page number is used as an index in the page array. The table of pages contains the base addresses of all the pages in RAM.

Paging model of logical and physical memory:

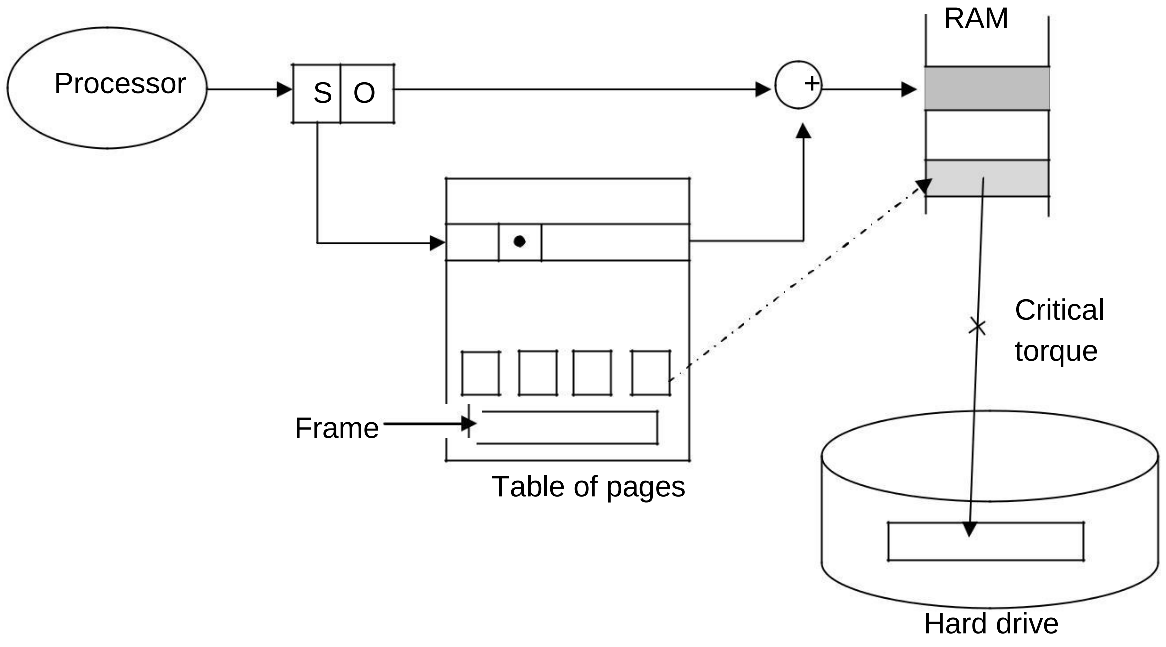


The size of the page (and also the frame) is determined by the hardware. This size is usually the power of 2 between 512 B and 16 MB per page, depending on the architecture of the computer.

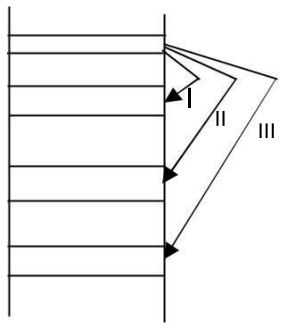
An important aspect of paging is the clear separation of memory viewed by the user from physical memory. A user program is created on the assumption that the memory area is continuous and contains only one program. In fact, the user program is scattered in fragments over the physical memory in which other programs are stored. The discrepancy between the memory seen by the user and the physical memory removes the equipment translating the addresses. Logical addresses are translated into physical addresses. This representation is hidden from the user and supervised by the operating system. The user process is not able to access the memory that it does not own. It has no possibility to address the memory lying behind the table of pages, in turn the table of pages contains only these pages, which belong to the process.

Since the operating system manages the memory, it needs to know all the details of its allocation: which frames are allocated, which frames are free, what is the total number of frames, and so on. This information is stored in an array of frames. It has one position for each physical frame of the page. In each position of the array there is information about whether the frame is free or busy, and if it is busy, to which side of the process or processes it is assigned.

PAGE FAULT - referring to the memory stored on the disk.



The probabilities of shares I-III differ significantly from each other:
I - 90%
II - 9%
III - 1%



PAGE REPLACEMENT ALGORITHMS

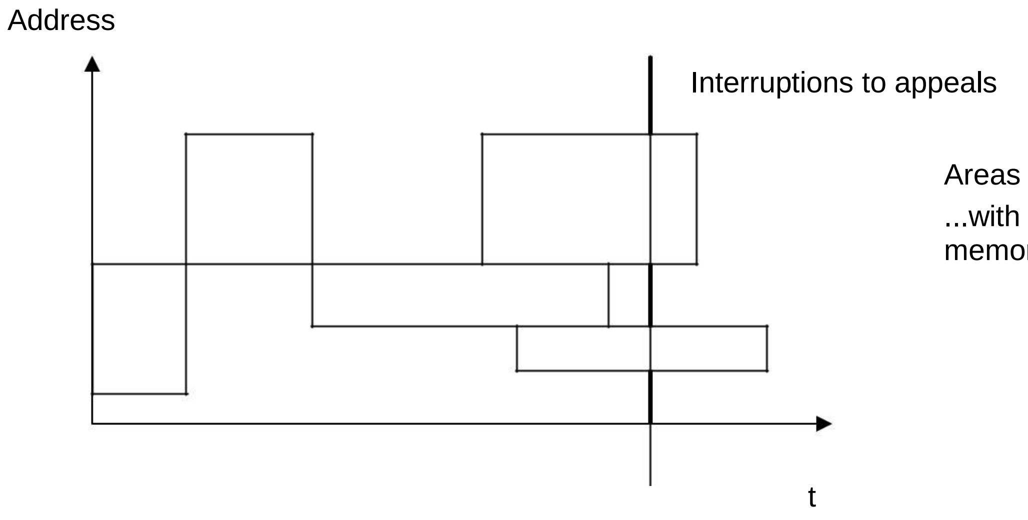
When a page error occurs and there is no free space in the physical memory (frames) then algorithms are used:

1) FIFO - Deletes the longest-serving page in the memory.
2) OPT - optimal algorithm - remove the page that will not be used for the longest time; the page error will occur as rarely as possible; disadvantage - the algorithm requires knowledge about the future reference string
3) LRU - last recently used - delete the page that has not been used the longest, you need to know the past - data about the past must be saved
4) ALRU - approximated LRU (second chance algorithm) - minimized amount of data, 1 bit

   - reference bit; when the page code is executed, the bit is set to 1 when a page error occurs, we look for a page to remove - when we hit a page with bit 1, we change it to 0, and when we hit a page with bit 0 - we remove it.
5) LFU - delete the least frequently used page
6) MFU - delete the most frequently used page

THE PRINCIPLE OF LOCALITY OF APPEALS

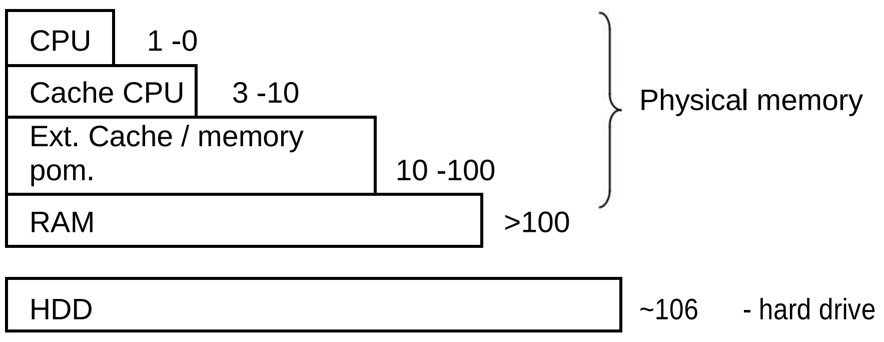
Application performance in SO



Areas are filled
...with references to memory.

The structure of the memory is multi-level.

In case of processor registers it is a matter of 1 or 0 access cycles.

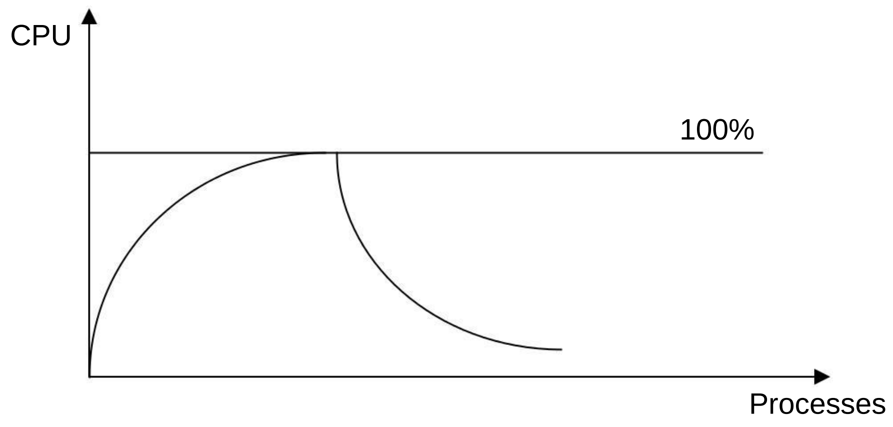


Effectiveness results from telecommunication constraints and economic conditions.

FRAME ALLOCATION ALGORITHMS

Algorithms answer the question of how much space it should be given the application is loaded into memory.
However, this question cannot be answered. There are 2 solutions:

- Allocation equal - each process gets the same number of frames
- Proportional allocation - each process is allocated an available memory according to its size.



There is a limit to the division. As the number of appeals increases, SO has to wait for HDDs. There are more and more applications, but the processor does nothing, we call it struggle.

WAYS OF ELIMINATING STRUGGLE

1. Zone model - is used to provide as many frames as needed for the application; it minimizes the chances of page error.

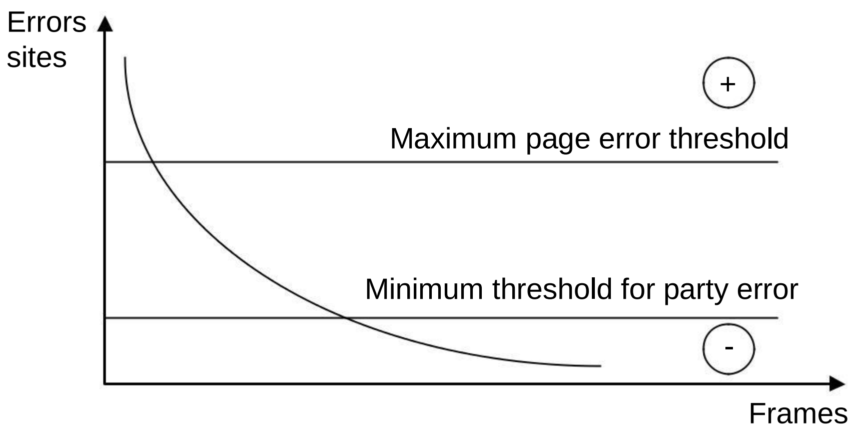
Strony 1, 2, 7, 5, 5, 8, 3, 9, 10, …

- working set window, the number of last references to the state that we are going to consider ZR - the working set of an application or a process

= 6
ZR = {1, 2, 5, 7} -> " ZR = 4

The size of the HRD is almost half as large as the size of the HRD. -> On this basis, the SO allocates 4 frames.

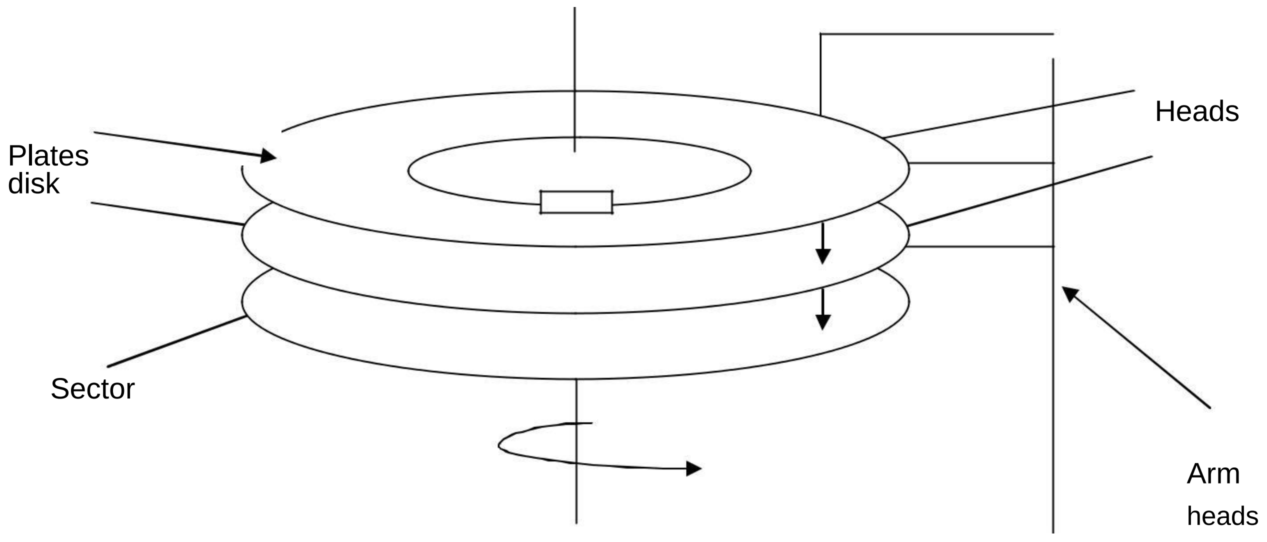
2. Control strategy for page error frequency.



SO returns frames to the process when it crosses the maximum threshold. However, the process that crosses the minimum threshold takes away the frames.

CHAPTER 6 - STORAGE STRUCTURE

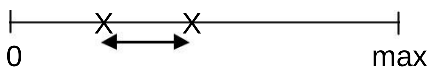
HARD DRIVE



Not every part of the HDD runs at the same speed. The head is much slower than the disk space.

Organisation of the transcript:

- The magnetic medium has magnetic paths that are divided into sectors.
- The information on the disk is stored in blocks.
- Blocks are numbered within one path.
- Then go to the next cylinder and do the same with the next path.

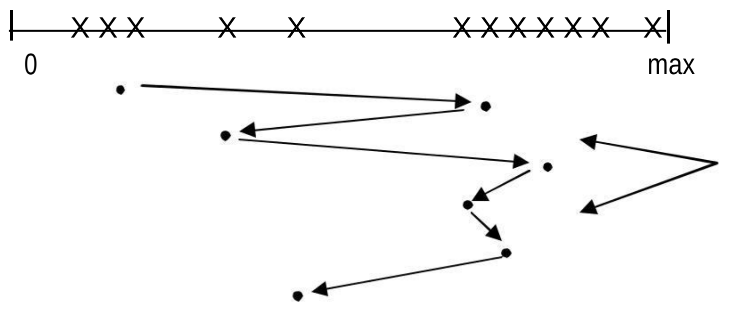


If the addresses are close together, there is a good chance that they are on the same cylinder. To access them, all you need to do is turn the cylinder. If the addresses are located on different cylinders, the head must also be moved - this takes more time.

HANDLING OF A SERIES OF NOTIFICATIONS

One of the tasks of the operating system is to provide quick access to the disk and fast transfer of disk data. Search time is the time taken to move the drive arm to the position where the heads are positioned in the cylinder containing the required sector. Rotational Delay means extra time spent rotating the drive to the position where the required sector goes under the drive head. The bandwidth of a disk is called the total number of bytes sent, divided by the total time that elapses from the first disk service contract until the end of the last transfer. By planning to perform input-output disc operations in the right order, we can improve both access time and bandwidth.

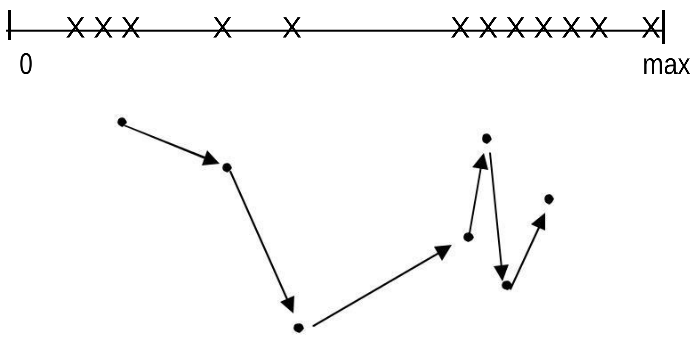
1. FCFS - as in the FIFO queue, the first request is supported. This algorithm is inherently fair, but generally does not provide the fastest service.



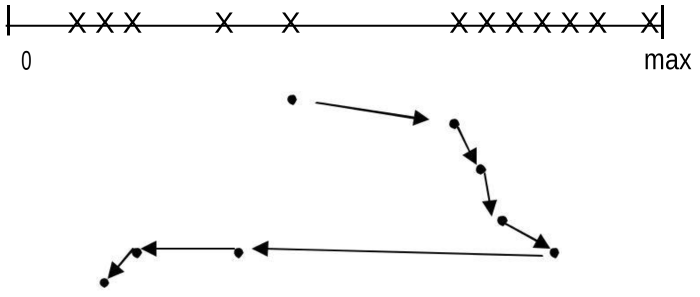
The head moves randomly.

The FCFS principle is ineffective - the average waiting time for processes can be very long.

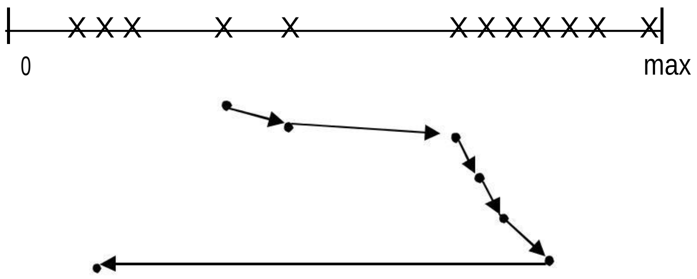
2. SSTF - first the shortest waiting time, the process is selected with the minimum search time against the current position of the head; the process located on one side of the disk can be "starved" if the head "gets stuck" on the other side and subsequent processes will appear there.



3. C-SCAN - The HDD head scans the entire surface for processes and executes them one by one, moving in only one direction.



4. SCAN - the head returns on the last active report. In the event of a heavy load on the drive, the time required to scan the drive from start to finish will be very long.



PLANNING ALGORITHMS

1. EDF - first the first deadline; it reminds the FCFS - it is not known whether the processes with the largest deadlins are in the same cylinder.
2. FD-SCAN - read the nearest deadline on the way up or down (similarly to SCAN).

MANAGING A FREE DISK SPACE

1. Bit vector.

   Each block is represented by 1 bit. If a block is free, the bit is 1, but if it is assigned, the bit is 0. The main advantage of this solution is that it enables relatively easy and efficient finding of the first free block or n subsequent free blocks.

   Unfortunately, the use of bit vectors is inefficient if you cannot store the entire vector in RAM. Bitstream vector storage in RAM is possible for smaller discs (used e.g. in microcomputers), but not for larger ones.

2. Create lists of free areas.

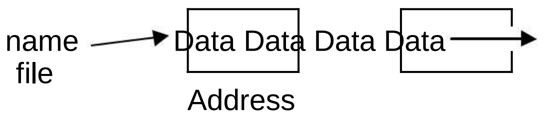
   It consists in binding together all free disk blocks and storing the pointer to the first free block in a special place on the disk and handy - in memory. The first block contains the pointer to the next free block etc.

   However, this method is not efficient - to go through the list, you need to read each block, which requires a significant amount of time spent on input-output operations. Fortunately, passing through the list of free blocks is not a common activity. Usually the operating system uses the first block from the list of free blocks.

SAVING FILES TO DISK

1. CONTINUAL CHAPTER - each file must occupy a set of blocks on the disk. Transitions between blocks do not require head movement. When head movement is necessary (between the last sector on one cylinder and the first sector on the next cylinder), it will move only one path. This means that the number of disk searches is minimal.
   However, this allocation has its drawbacks:
   - If the disk starts to overflow, it will be possible to move file areas.
   - This is a special case of a general dynamic memory allocation problem (chapter 9.3).
   - External Fragmentation - As a result of creating and deleting files, the free disk space is divided into small pieces. This becomes troublesome when the largest continuous piece is not enough to meet the order.
   - It is difficult to determine the size of the area needed for a file. If you don't allocate enough space to a file, it may not be possible to extend it.

2. LIST CHAPTER - removes all problems associated with continuous allocation. Each file is a list of related disk blocks. These blocks can be located anywhere on the disk. The directory contains the pointer to the first and last block of the file, and each block contains the pointer to the next block.
   In order to create a file, we create a new entry in the directory. Each entry in the directory has a pointer to the first block of the file. The initial value of this pointer is nil (this means the file is empty), and the size field is 0. A writing operation causes the system to find free areas of an unoccupied block and save it and link it to the end of the file. Reading the file is done according to the indicators stored in the following blocks.



   Advantages over continuous allocation:
   - There is no need for continuity for a single file, so there is no need to move anything.
   - There is no external fragmentation.
   - Each free block from the list of free areas can be used to fulfill the order.
   - You do not need to declare the file size at the time of creation - the file can grow as long as there are free blocks.

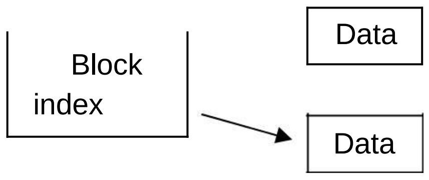
   Disadvantages of the letter assignment:
   - Low efficiency - you have to "jump" on the disk. Each access to the pointer requires reading the disk.
   - If we want to read the last block of the file we have to go through the whole sequence of blocks.
   - Indicators take up some space - each file needs a little more space than in any other case. You can solve this problem by grouping the blocks into several blocks in a so called ""block of blocks".

Clusters and assigning clusters instead of blocks. The cost of this solution is an increase in internal fragmentation, because if the group is partially filled, more space is wasted than in a partially filled unit. The cluster is used in most operating systems.

- Unreliability - files are tied to indicators scattered all over the disk, and if lost or damaged, the indicator may cause the wrong indicator to download. Such an error could cause the file to be bound to a list of free spaces or to another file. A partial solution is to use double-linked lists or memorize the file name and relative block number in each block. However, such solutions increase the cost of each file even more.

3. INDEX CHAPTER - concentrates indicators in one place - in the index block. It solves the problem of letter assignment, where indicators to blocks are scattered all over the disk and need to be recovered in turn. An index allowance combines the advantages of two previous allowances.

   We divide blocks into: index blocks and data blocks. Each file has its own index block, which is an array of disk block addresses.



When creating a file, all the indicators in the index block are given a nil value. When block i is first written, it is downloaded from the Free Zone Manager and its address is placed in the i-th position of the index block.

Advantages:

- An index allowance combines the advantages of two previous allowances.
- Enables direct access without causing external fragmentation, because an order for additional space can meet a free block located anywhere on the disk.
- Sequential access is facilitated.

Defects:

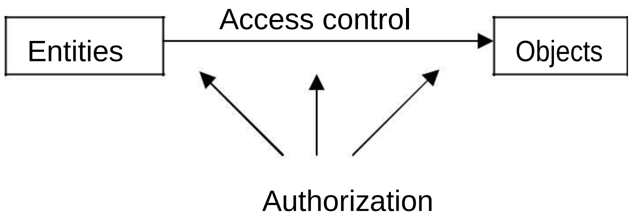
- Waste of space - index block indicators generally take up more space than letter assignment indicators.
- The index assignment schema has similar performance problems as the letter assignment. In particular, index blocks can be cached, but data blocks can be scattered across the entire disc partition.

Waste of space raises the question of what should be the size of an index block. Since every file must have an index block, it is important to keep it as small as possible. However, if it is too small, it will not contain enough indicators for a large file. A mechanism should therefore be developed to deal with this situation. There are 3 solutions:

- List circuit diagram: An index block is usually contained within a single disk block, so that it can be read or written directly. To allow you to organize large files, you can combine several index blocks.
- Multi-level index: A list representation variant is to use a first-level index block to indicate a set of second-level index blocks whose indicators already point to file blocks.
- Combination diagram: Another approach (used in the UNIX system) is to store e.g. the first 15 indicators of an index block in an i-frame file. The first 12 of these indicators point to direct blocks, i.e. it contains the addresses of the blocks with the file data. Thanks to this, data in small files do not need to have a separate index block. The following 3 indicators indicate intermediate blocks. The first display of the intermediate block is the address of the single block. One-direct block is an index block that contains the addresses of data blocks instead of the target data. The two-medium block pointer then follows, which contains the block address with block addresses containing the pointer to the actual data blocks. The last indicator would contain the address of the three-way block.

CHAPTER 7 - SECURITY IN OPERATING SYSTEMS

We want to protect information stored in a computer system against physical damage (reliability) and improper access (protection).



Entities - an active element, its task is to access the Facility
Objects - the container transmitted (file, data structure, external device) stores data

ACCESS CONTROL

- authorisation - is based on the knowledge of the entity or on authorisation based on tokens, volleyball analysis, etc.
- audit - observation and processing of data on the user in the system

Access Control Lists (ACLs) - each entity in the system is assigned an object list. This is an identity-dependent access and consists in associating with each file and directory of the aforementioned fox. It contains the user names and the types of access allowed for each user. Example:

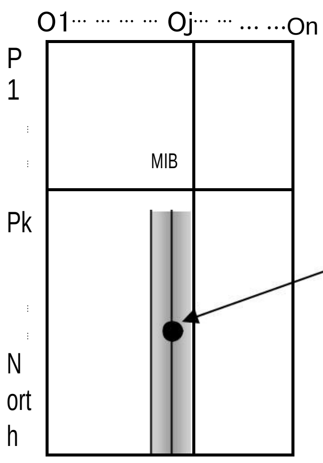
When a user requests access to a file, the operating system will check the list of accesses assigned to that file. If a user appears next to a particular type of access, he or she will get it. Otherwise, the protection rules are violated and the user task is denied access to the file.

The advantages of the list:
- enabling complex access methods

Defects:
- Length of lists.
- If we want to allow everyone to read a file, we need to list all the users who are allowed to read it. This has two negative consequences:
  - Making such a list can be a tedious and unprofitable activity, especially if you don't know the list of users in advance.
  - A directory entry that was previously fixed in size must now be of variable length, which complicates disk space management.

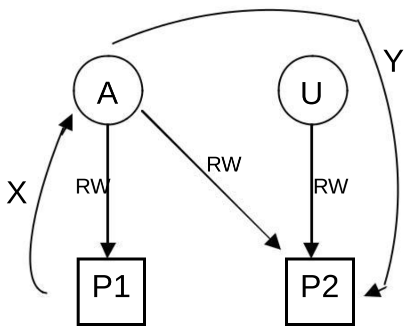
You can avoid these problems by using a dense version of the access list. In order to shorten it, in many systems, 3 user classes refer to each file:
- Owner - the owner is the user who created the file.

- Group or work team - a set of users who share a file and need similar access.
- The universe - all other users.



Pk subject's right to object Oj
R - Reading rights
W - the right to write
C - possibility to change access rights
You can manipulate a whole column of Oj

Basic access right - the right to read and write a file. Some systems have additional rights, e.g. the possibility to change access rights.



A - Administrator
U - user
P1 - top secret file
P2 - Open file
X - administrator reads the contents of P1 file and saves it to a P2 (Y) file. That way... the user has access to the data in the a secret P1 file

THE BREAKDOWN OF OPERATIONAL SYSTEMS FOR REASONS OF SECURITY POLICY

TYPES OF ACCESS CONTROL

- DAC - (Discretionary Access Control) discreet access control - allows the user to fully establish access rights to their resources. Control of access to data objects is based on verification of the identity of the owners/groups to which it belongs.
- MAC - Mandatory Access Control - Mandatory Access Control - is based on trust levels, the entity requesting access to the facility must belong to the appropriate category/group. It's found in high confidentiality systems, military systems. We are also introducing additional assumptions:
  - Existence of safety classes in operating systems - the set of these classes should be partially ordered
    ST > T > P > J

    ST - top secret
    T - secret
    P - confidential

- All data and active entities in the operating systems are labelled - there can be no object in the operating system without security classes.
- The entity's safety class must be greater than the facility's safety class for reading. In the case of recording, the entity's security class must be lower than the facility's security class.
  REGISTER: KB of entity ≥ KB of object
  READING: KB of the entity ≤ KB of the facility

  Example: The P class can be assigned to ST, T and P for writing, and the P class can be assigned to P and J for reading.

  The current security class of the entity can be changed. An operating system may downgrade the safety class to a level that does not exceed the safety class assigned to it.

THE SECURITY CLASSIFICATION OF THE OPERATING SYSTEMS

TCB - computer security class. It is a set of all protective systems in a computer system (hardware, software, firmware) that correctly enforce safety rules.

- D - minimum protection - operating systems that do not meet the requirements of any other class, e.g. DOS, Windows 95
- C - are based on discreet access control (DAC); sets out any rules for the protection and responsibility of users and their activities by allowing them to see what they are doing (audit).
  - C1 - basic access control for multiple users:
    - It contains control measures that enable users to protect information and prevent other users from accidentally reading or destroying data.
    - In this class of environment, cooperating users have access to data at the same sensitization levels.
    - TCB supervises access between users and files by enabling users to define and control rules for sharing objects with named persons or defined groups.
    - The TCB Database requires the user to introduce himself/herself before starting any activities in which the TCB Database acts as an intermediary.
    - Authentication data is protected in the TCB database to make it inaccessible to unauthorized users.
  - C2 - contains the same requirements for class C1 system and additionally:
    - An individual level of access control is added, e.g. file access rights per individual person.
    - The system administrator may selectively track the activities of any user or group of users on the basis of their individual identity.
    - There is an audit - collecting information about users, e.g. number of logins, number of attempts to access secret files, etc.
    - The TCB database also protects itself from changing its code or data structures.

In addition, any information created by a previous user will not be shared with another user who returns to a memory object transferred back to the system.

Some special, protected versions of the UNIX system have C2 level certificates

- B - is based on Mandatory Access Control (MAC); Class B systems have all the characteristics of Class C2 systems and are connected to each facility in the system sensitization label:
  - B1 - meets the basic assumptions (labelling, the principle of class imbalance):

    A security label shall be maintained for each facility in the system used to decide on the mandatory access control, e.g. the user shall not access the file at an even more security-sensitive level (T, ST).

    The TCB database also defines a framework of sensitisation levels for each side of a human readable result.

    The TCB database not only controls the usual authentication information (username and password), but also checks (background checks) and authorizes individual users and provides at least two levels of security. These are hierarchical levels, so the user has the right to access objects whose sensitization labels are equal to or smaller than his or her security certificate, no. A user at a secret level could have access to a file at a confidential level, despite the lack of other access regulations.

    Processes are also insulated by the use of separate address spaces.
  - B2 - additional authorization; formal verification of configuration (SO)

    Sensitization labels are assigned to all system resources, such as memory objects.

    Physical equipment shall be assigned minimum and maximum safety levels, which shall be used by the system to enforce restrictions.

    emissions from the physical environments in which the equipment is located. The B2 system makes secret channels available and enables the tracking of events that can lead to the exploitation of a secret channel.
  - B3 - TCB - Secure database - everything that happens in the operating system is stored, which has a big impact on the reduction of efficiency:

    You can create access control lists that define users and groups who are denied access to the object with a given name.

    There is also a mechanism in the TCB database to monitor events that may indicate a breach of security. This mechanism notifies the administrator in charge of security and (if necessary) closes the event chain in the least loss-making way.

    Obligation to verify each operation.
- A - abstract class - formal verification of the operating system based on the system code is required; mathematical verification of the code is very rarely needed.

The use of the TCB database guarantees only the ability of the system to enforce its security policy. The database does not specify which policy should be adopted. In general, in a given computer environment, security rules for obtaining certificates are developed and a plan approved by a given security agency, e.g. NCSC or TEMPEST, is disposed of.

CHAPTER 8 - DISTRIBUTED OPERATING SYSTEMS

Distributed system (distributed system) is a set of loosely connected processors connected by a transport network. For a specific distributed system processor, the remaining processors and their resources are remote, while their own resources are local.

Processors in a distributed system may differ in computing power and features. They could be among them:

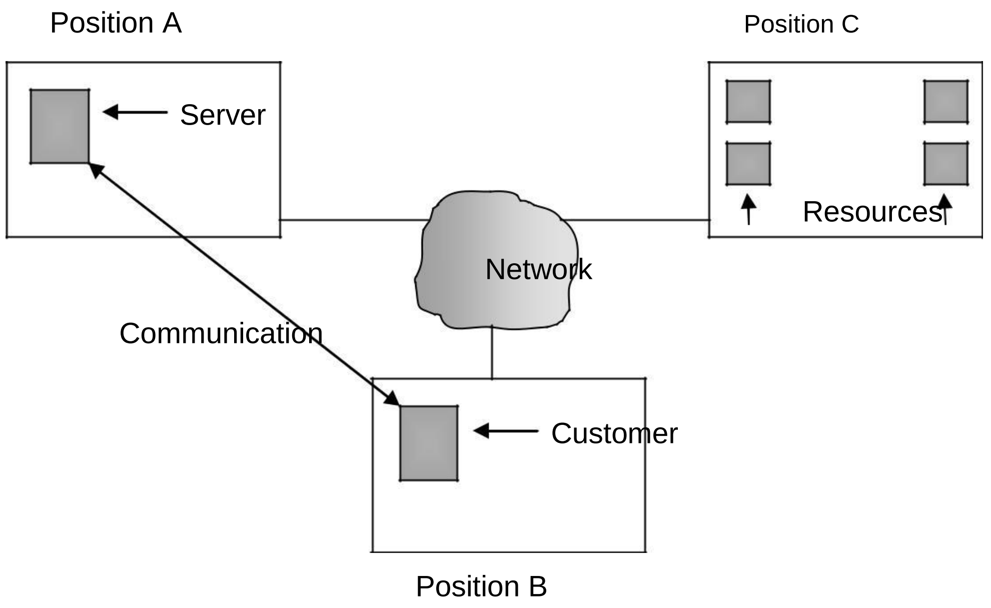
- microprocessors,
- workstations,
- minicomputers,
- large, general-purpose computer systems.

Processors are defined by several names depending on the context in which they are spoken about:

- position - when we want to pay attention to the position of the machines,
- knots,
- network computers - when we will refer to a specific system in a given place.

A certain process on some position - called a server - has a resource that is needed by another process on another position - a client, i.e. a user. The task of a distributed system is to create an efficient and comfortable environment enabling such a way of resource allocation.

Schematic diagram of the distributed system:



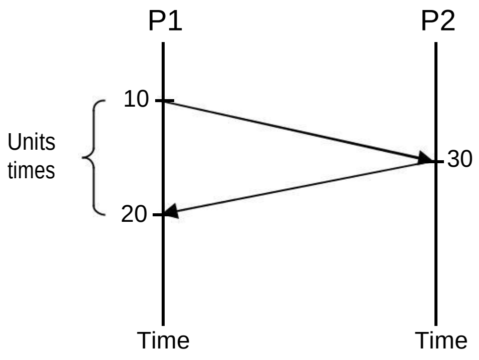
Characteristics of distributed systems:

- There is a sharing of resources - devices give each other access to their resources.
- Flynn Taxonomy - A classification of computer architectures based on the number of data streams and command streams processed:
  - SISD (Single Instruction, Single Data) - one data stream is processed by one executed program (traditional single-processor computers).

- SIMD (Single Instruction, Multiple Data) - multiple data streams are processed by a single executed program (matrix processors with one instruction execution unit, which takes the instruction and then manages multiple units in parallel - each of them operates on its own data).
- MISD (Multiple Instruction, Single Data) - many parallel programs process one common data stream simultaneously. This solution is a bit
  is pointless and has few applications (none of the known computers responds to this model).
- MIMD (Multiple Instruction, Multiple Data) - parallel execution of many programs, each of which processes its own data streams (multiprocess and muli computers).

SYNCHRONIZATION

Distributed architectures do not directly translate parameters into functionality. Other issues such as time and synchronicity are important. In the case of complex architectures, the indications of the clock may vary. This causes the problem of establishing simultaneity of events and determining the order of events.



Techniques used in distributed systems to address this problem:

- Lamport Algorithm - setting the sequence of events in the system. The algorithm detects time anomalies and corrects them in real time (e.g. detecting an event before it is broadcasted, as shown in the above drawing). Verifies clock readings - in the example above, he would add up to 10 +20 so that the event is received at the right time rather than before it is broadcast. WADA: With such a strategy, all clocks align their time with the fastest moving timer in the system.
- System C clock synchronization (coordination) -

  Absolute clock indications

  $\neq 1$

  Because ~~it~~'s heading for 1, so:

  $1 - \beta \leq — \leq 1 + \beta \Rightarrow dt - \beta dt \leq dC \leq dt + \beta dt$

  $\Downarrow$ If we assume there are two identical clocks.

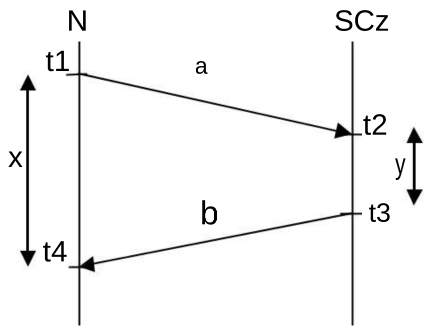
  $2\ \beta dt \leq$

dt ≤ — - the maximum time after which we synchronize

β - low value, coefficient of deviation of the clock in the system

CLOCK SYNCHRONISATION STRATEGIES

1. Cristiana Algorithm - in dispersed systems there is one machine that we treat as a basic clock and call a time server.



N - sender
SCz - time server

a - the sender's machine asks for the current clock reading
y - time to send a reply
t4 - the sender's machine knows what time it was on t3

The time of sending the message is negligently small in relation to the time synchronization, so the sender has a precisely measured time x. We assume that times a and b are symmetrical and small.

* T3 - t1 ——— - an estimate of the difference
* between t3 and t1

Unfortunately, this is not entirely the case as set out above. There are systems closer and farther away from the time server, so in a larger distributed operating system this procedure gives different time for different elements of the system.

2. Distributed (decentralized) algorithm of clock synchronization - all elements are treated fairly, no matter the distance from the server.

   Each clock sets the time taking into account the average time of all the clocks, including the travel time of the messages.
   This procedure is rarely used.

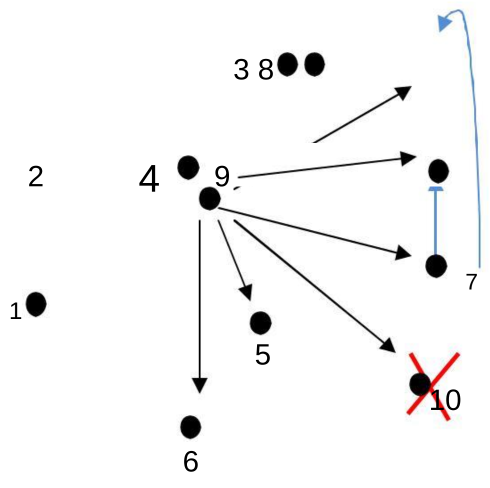
3. Election algorithms - selection of a single machine from the available pool.

   Many distributed algorithms require one of the processes to act as coordinator, initiator, arranger or other special function. In general, it does not matter which process takes over these special duties, but only one process should carry them out.

If all processes are exactly the same, without any characteristic features, there is no possibility to choose a special one from among them. Therefore, we can assume that each process has a unique number, such as its own network address. Election algorithms usually try to locate the process with the largest number and appoint it as the coordinator.

In addition, it is assumed that each process knows the numbers of all other processes. Processes do not know, however, which of them are currently working and which are immobilized.

1) The tyrant's algorithm



A process that notices that the coordinator has stopped responding to orders starts the selection process. The P process carries out the selection as follows:

1. P sends an election message to all processes with larger numbers.
2. If no one answers, P wins the election and becomes the new coordinator.
3. If any of the processes with a larger number sends a response, it takes control and the role of the P process ends.

At any time, the process can receive an election message from a smaller colleague. When such a message is received, the recipient sends an OK message to the sender to inform him that he is active and takes over control. Then the customer will continue with the election, if he has not already done so. In the end, all but one of the processes have failed to elect. This one remaining process becomes the coordinator. Announces its victory by sending a message to all processes.

If a process, which was previously switched off, returns to the action, it starts to make choices. If it happens that it has the largest number among the currently executed processes, it wins the election and takes over the functions of the coordinator. It is therefore a situation that the process with the biggest number always wins the election.

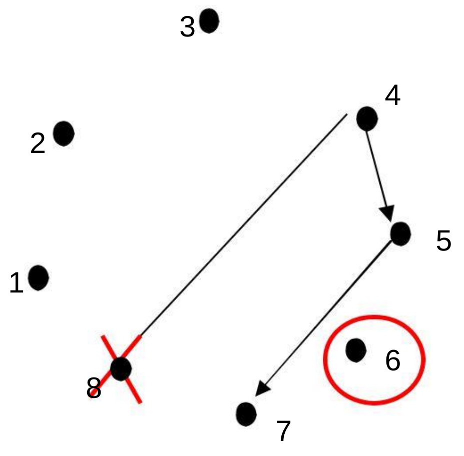
On the basis of the above drawing it looks like this:

Process number 10 is destroyed, which is noticed by Process number 4. Everyone with a higher number than 4 gets a message. Each of them does the same thing - sends a message to higher numbers (as an example, the sending of a message from 7 to 8 and to 10 is given). 9 will not receive a single confirmation because 10 are "dead", so 9 wins and sends a message to the others about the end of the election. At this point, the 9th becomes the most important.

The tyrant's algorithm leads to the emergence of a new central unit. This is a fast-track strategy

- after 2 passages, a new governing unit is elected. However, there is a problem when the process number 1 starts sending messages to all (not number 4 as in the example). Then 1 alone will send n-1 messages.

2) Ring algorithm



In this algorithm it is assumed that the processes are physically and logically ordered, so that everyone knows their successor. When any process detects a coordinator's inaction, it builds an ELECTION message that contains its own number and assigns it to its successor. If the successor is turned off, the sender omits him and contacts the next member of the ring or the next one behind him - until a working process is found. In each step the sender adds his process number to the list in the message.

Ultimately, the communication comes to the process that initiated its circulation. The process recognizes this event when it receives a message with its own number. Here the message type is changed to KOORDYNATOR and the circuit is repeated, this time to inform all other processes about the new coordinator (the process which has the biggest number in the created list) and about the members of the new ring.

On the basis of the above drawing it looks like this:

Process number 4 notes that there is no coordinator (Process number 8 has died) and sends an acknowledgement message to Process number 5, which passes it on to attach its number. Six is broken, so he's sending seven and seven to be the new coordinator.

The number of messages will be smaller than the tyrant algorithm, but the time will be worse because the messages have to promise the entire ring. This algorithm is used in computers even though the tyrant algorithm is more efficient and faster.

SYNCHRONIZING THE OPERATION OF PROCESSORS

Access to critical sections will take place through TRANSACTION. (page 178)

Transaction - has its beginning and end, consists of structured sections of activities. Features:

- Atomicity - a sequence of actions can be performed in full or not at all, cannot be performed in part
- consistence - the transaction must not affect the system's stability; the operating system does not have much to say here
- Isolation - if there is more than one transaction in the system, they overlap in time
- durability - once a transaction is stopped, it cannot be revoked; the effects of the transaction cannot be reversed

The above properties are often referred to as ACID from the first letters of the English names.

Granulation of resource handling - if a file is opened by a large number of transactions it can be troublesome.

TRANSACTION IMPLEMENTATION METHODS

1) Private workspace

   A process starting a transaction is assigned a private workspace containing all the files (and other objects) to which it has access. The transaction will not be approved or abandoned until all reading and writing operations relate to a private workspace and not to a "real" filesystem.

   There are two optimizations for this implementation:

   1. There is no need to make a private copy of a file that is read by the process but not changed. In this case, the real copy can be used (unless it has been changed after the start of the transaction). As a result, it is enough to create a private workspace for a process that starts a transaction and has nothing but a backward pointer to the workspace of the parent process. For transactions at the highest level, the working space of the parent process is the "real" file system. If the process opens a reading file, the indicators apply backwards until the file is located in the workspace of the parent process (or further ancestors).
   2. If a file opens up for saving, it can be found just as it was when you read it, but now you have to copy it to your private workspace first. The second optimization removes most of the copying operations here. Instead of copying the entire file to the workspace, only the index of the file is copied. An index is a data block associated with each file that contains information about the location of the file's disk blocks. Using a private index, you can read the file in the usual way, because its disk addresses refer to the original blocks of the file on the disk. The first modification of a file block causes a copy to be made, the address of which is inserted into the index. You can then update the block without affecting it.

the original. Added blocks are treated similarly. New blocks are sometimes called the shadows of blocks.

The process executing the transaction looks at the changed file, but all other processes still look at the original file. For a more complex transaction, a private workspace can contain multiple files instead of one. If you stop transactions, you simply remove the private workspace and all private blocks return to the list of free blocks. If a transaction is approved, private indices are moved indivisibly into the workspace of the parent process. Blocks that have become inaccessible after this procedure are placed on the list of free blocks.

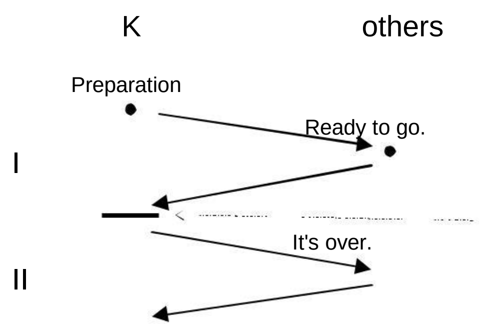
2) The register of pre-registrations is sometimes referred to as a list of intentions.
This method allows you to modify files in places where they occur, but before changing any block, a record is saved to the register of pre-recorded records stored in the persistent memory. Records are recorded there:
- Which transaction makes the change
- What files, sectors are subject to transactions?
- When a transaction takes place
- Status and new provisions

A change in the file shall be made only after the data have been successfully recorded in the register.
If the transaction is successful and approved, the validation record is recorded in the registry, but this does not entail changes to the data structures as they have already been updated.

Transaction Approval - Once a transaction has been completed, everyone involved in the transaction must know that the transaction has been completed and with what result.

The transaction approval protocol is used - it assumes that there are 2 process groups - a transaction coordinator and others.

2-Phase Approval:
1. Preparation for the approval of the transaction - when the coordinator sees that all the preparations have been completed.

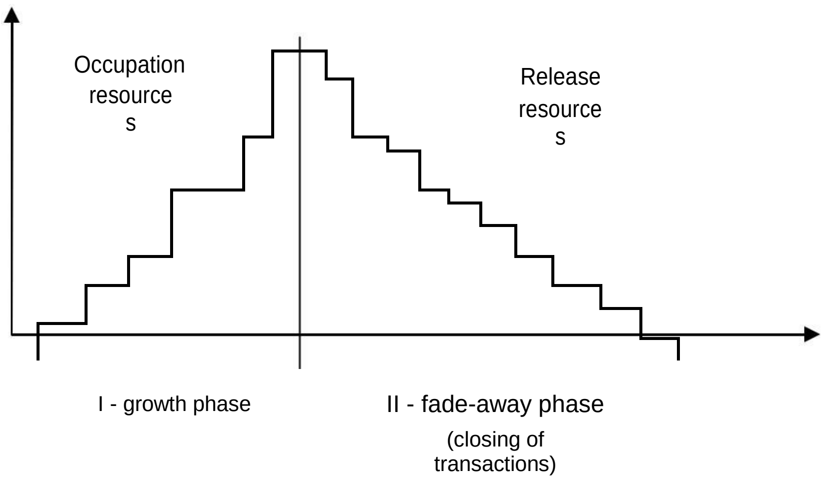


The coordinator knows that the transaction has been completed, yet others do not know about it

It is used to implement trading systems:

1. Occupation

- If the system reads something in a transaction, the data is closed and no one has access to it during the transaction.
- Occupations can be performed with one centralised class manager or with local managers on individual machines that manage local files.
- Performing classes and layoffs exactly when needed and when no longer necessary can lead to inconsistencies and blockages.
- In order to avoid blockages, the technique of TWO-FORE CONTRIBUTION OF RESOURCES is used.



The process in the growth phase first performs all the activities and then slows them down in the disappearance phase. If the process refrains from updating any file until it reaches the malaise phase, then the failure to do any activity can be dealt with simply by slowing down all the activities, waiting for a while and starting all over again.

In many systems, the fade phase does not occur until the transaction is completed by approval or omission. This is called a strict two-phase occupation.

Occupation, even in a two-phase organization, can lead to blockades. If two processes try to do two of the same activities in reverse order, a blockade may occur. In such cases, typical techniques are used, such as performing all classes according to a certain canonical order, which prevent holding and waiting loops. You can also use blockade detection by maintaining a graph that clearly shows which processes have done what, and which processes are applying for it, and analyzing it for loops.

2. Time stamps

Each transaction shall be assigned a time stamp at the time of its reporting. Each file in the system shall be associated with a reading time stamp and a writing time stamp informing of the time at which the approved transaction was last read or written. If transactions are short and rarely spaced out over time, then when a process tries to access a file, the reading and writing time file tags will generally be smaller (older) than the current transaction time tag. Such...

orderly means that transactions are processed in the right order and everything is in order.
It's okay. It's okay.

3. Optimistic supervision of transactions:
   - We don't control anything.
   - We assume that there will not be a blockade, everything happens in due course.
   - This approach can be used when the number of transactions is small.
   - The information about which files have been read and stored shall be maintained; at the time the transaction is confirmed, it shall be checked whether any other transaction has changed the supervised transaction files after the start of the transaction; if so, the transaction shall be abandoned, otherwise the transaction shall be confirmed.
   - The advantage of this solution is its resistance to blockages and enabling maximum parallelism of actions, because no process ever has to wait to be occupied.
   - The disadvantage is that a possible failure leads to a repeat of the entire transaction.

LOCKS AND BOLTS

The interlocks in distributed systems are similar to those in single-processor systems, but they are even worse. It is more difficult to avoid, prevent or even detect them, and it is more difficult to apply corrective measures when they are detected, because all the necessary information is scattered across many machines.

DETECTION OF DISPERSED INTERLOCKS

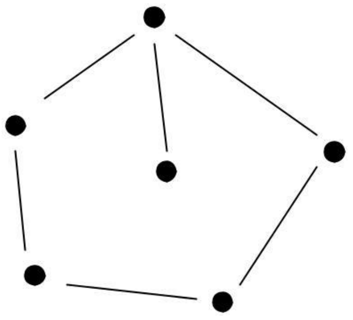
1. Centralised lock detection

   Although each machine maintains a graph of its own processes and resources, the central coordinator works on a graph of the entire system's resources (created by combining all the individual graphs). If the coordinator detects a loop, he ends one of the processes in order to break the blockade.

   Unlike in the centralised case, where all information is automatically available in the right place, in a distributed system it should be provided openly. Each machine supports a graph of its own processes and resources. There are several ways to obtain this information. First of all, every time you add or remove an edge in the resource graph, you can send a message to the coordinator with a corresponding update. Second, periodically, each process can send a list of added or deleted edges since the previous update. The second method requires less messages than the first. Thirdly, the coordinator can ask for the necessary information when the time comes. Unfortunately, none of these methods work properly.

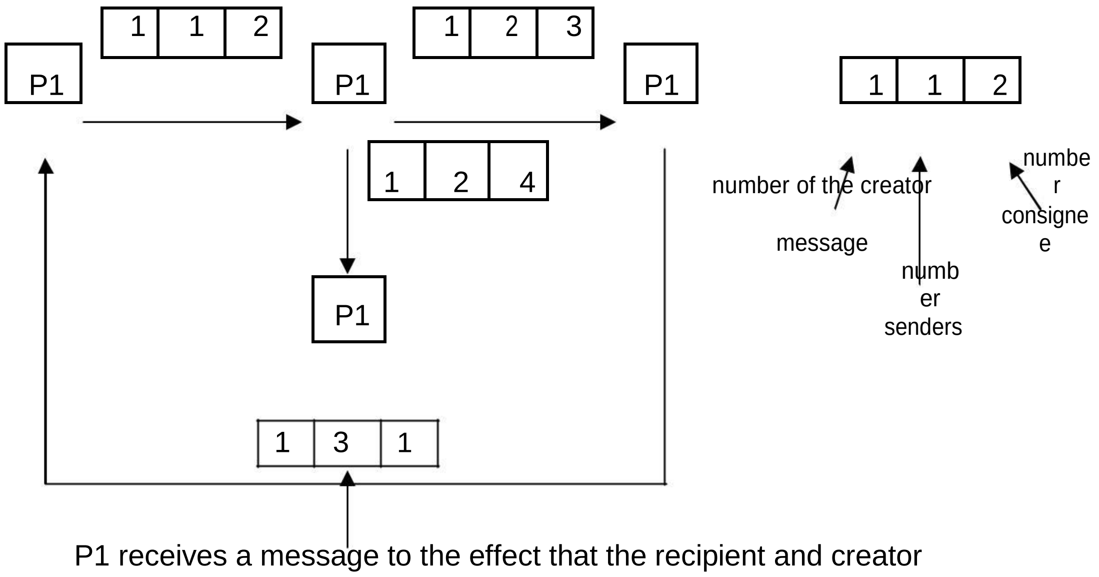
   There could be a situation called a false blockade. Many algorithms for blockages in distributed systems produce false blockages due to incomplete data or delayed information.

No idea for a graph of resource allocation in distributed operating systems.



2. Chandy-Misra_Haasa algorithm - an example of distributed blockade detection

- Processes exchange information about resource needs.
- The process waiting for a resource sends a message to another process (P1 waits for something that has P2).



P1 receives a message to the effect that the recipient and creator of the message is 1, so there's a blockade.

Only potential message creators are involved in the detection of blockages. The solution is the suicide of the author of the communiqué. Other solutions in the case of interlocks - the same as in the case of the operating system.

PROCESS MANAGEMENT IN DISTRIBUTED SYSTEMS

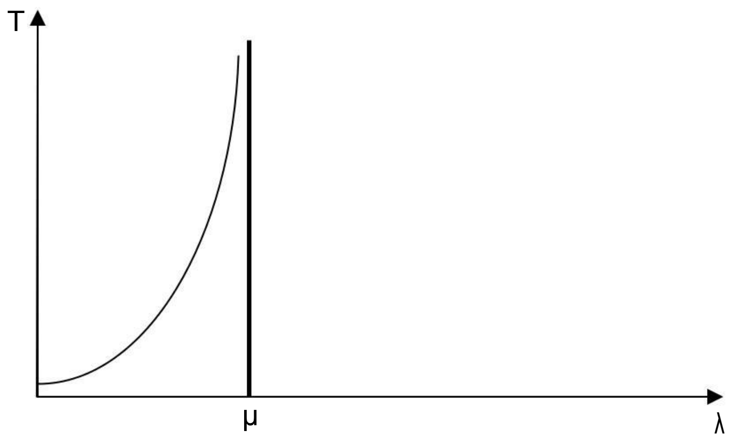
It is assumed that the processors are identical and the processes can be moved.

λ - stream density (stream of submissions) of processes on a given processor (e.g. 10 processes per second)

μ - handling flux - ability of the ALU to execute a number of processes at a given time $\mu > \lambda$

T - waiting time of the process in the queue

$T = \frac{}{\mu}$ - in case of one queue, each process has its own queue



Assuming we make one big round:

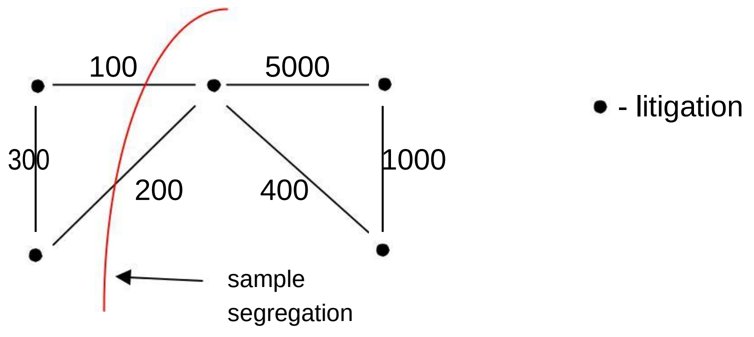
$T = \frac{}{\mu} = - \cdot$

In distributed systems, centralized management leads to a n-fold reduction in time.
BUT the creation of such a queue is cumbersome - there must be good communication between processes:

- In the case of a muli of computers, we have a complicated situation - you have to take into account the time of sending messages.
- It is not always worthwhile to run processes on random computers.

SAMPLE ALGORITHMS FOR CPU ALLOCATION

1. Deterministic algorithm based on theoretical graph - used to minimize network transmissions

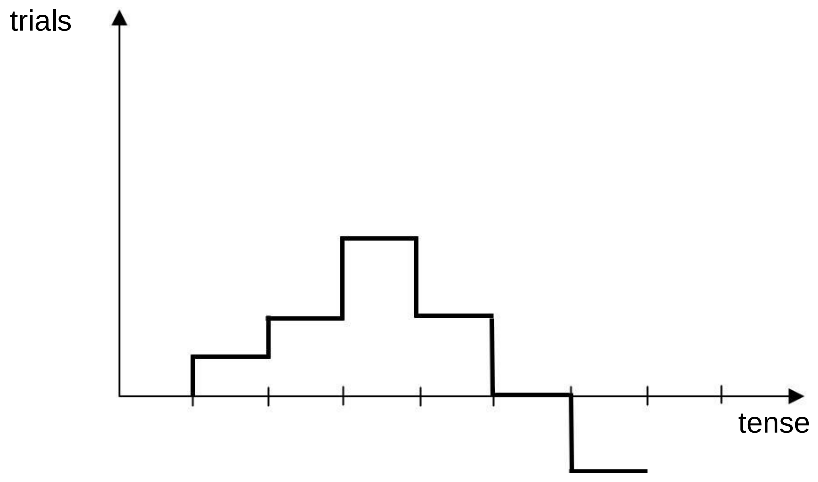


   The graph is divided into partitions so that the sum of the connecting edges is minimal. WADA - you need to know the numbers and the set of vertices. For open architecture, the set of vertices and edge labels are not known.

2. Algorithm UP AND DOWN - process groups are to be handled fairly.

   The coordinator maintains use tables that contain one entry per user, initially zero. When significant events occur, messages are sent to the coordinator to update the table. Allocation decisions shall be made on the basis of secret ballot papers.

the array at the moment when the event triggering the scheduling procedure occurs: CPU demand, CPU release or clock pulse.



The entries in the array can be positive, zero or negative. Positive entries mean that the workstation uses the network resources of the system, while negative entries mean that the workstation needs resources. Zero entries are neutral.

This strategy is adaptive. If a new user appears, who needs a lot of processes - the graph grows very much, but also quickly drops to negative values and will wait.

3. Distributed algorithm for balancing central unit loads
   - We assume that for each calculation unit a threshold value is allowed, which cannot be exceeded.
   - If we detect the threshold being exceeded, the system has to select a new unit and if it has a value lower than the threshold, all tasks are directed to it - after a certain number of random samples, the draw stops and all activities on the overloaded unit are stopped.
   - <u>Algorithm mutation</u> - the processor on which the draw is finished can draw another one and check if it has crossed the threshold; then it takes away tasks.

REACHING AN AGREEMENT IN THE SYSTEM

1. Byzantine algorithm (Byzantine generals)

1
2
3
flawed
4

Processors, each of which generates a message and sends it to the others. Some processors can send out false messages. Everyone sends their message to the neighbors.

1. – messaging

2. – (1, 2, x, 4) (1, 2, y, 4) (?, ?, ?, ?, ?) (1, 2, z, 4) - message vectors

Processor 1 has collected the response vectors and is looking for a traitor.
Processors 2 and 4 get the same resultant vector.

3. - messaging
4. - – (1, 2, y, 4)
   (a, b, c, d)
   (1, 2, z, 4)
   ____
   $\boxed{(1, 2, ?, 4)}$ - resultant vector - we know which processors work well and which do not
   ↑

In the resultant vector, enter the values that occur in the predominance.

For 3m + 1 processor a maximum of 3 traitors can be detected.

DISTRIBUTED FILE SYSTEMS

We divide into two groups:

- File services - operations on files, e.g. reading, writing, changing permissions
- Directory services - file location: control of file copying mechanisms

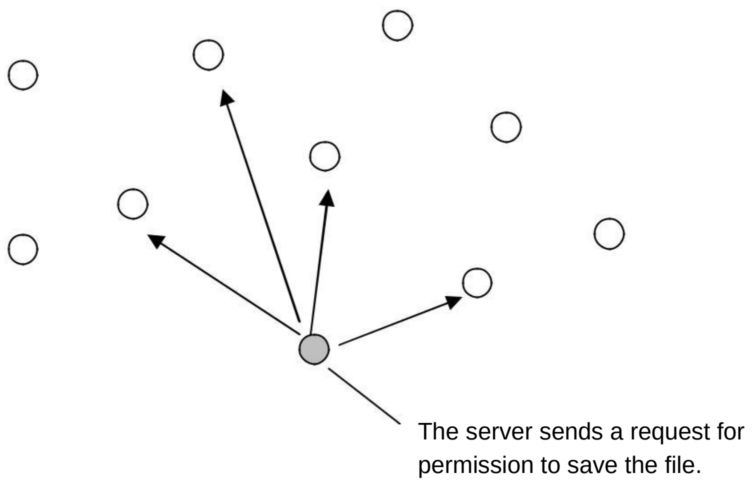
Semantics of file sharing:

1. UNIQUE semantics - reading from a file returns the last written value
2. SESSION Semantics - changes that occurred during the session are visible only after the session is finished
3. FIXED FILE Semantics - there are no write operations to the file, each generation of the file is creating a new file, there is no access to the file that has outdated data; other files must obtain a new file name
4. TRANSACTIONAL semantics - each access to the file must be performed by means of a transaction; file operations must have transaction characteristics

Most of the files on the system are temporary; only a small part of the files are permanent.
Catalogue services must be provided in two ways:

- The file identifier contains the position identifier

It is assumed that for read and write operations it is necessary to obtain permission from N/2 + 1 server.



In the picture above we have 4 outdated versions and 5 current files. Among N/2 + 1 servers there is at least one that has the current version of the file.

Modifications of the method:

- The N-value is divided into the reading and writing quorum and the sum of the N-values must be sharply greater than

  N:

  N >

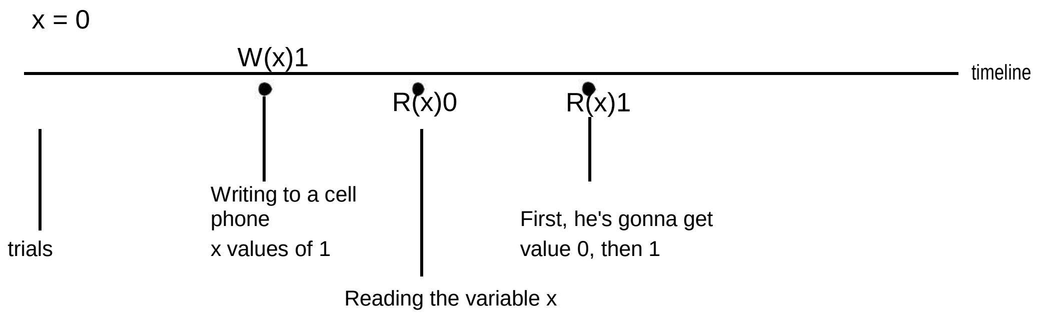
In the case of RAM, the situation is worse than in the case of file operations:

- Memory - the demand for sharing resources is very high

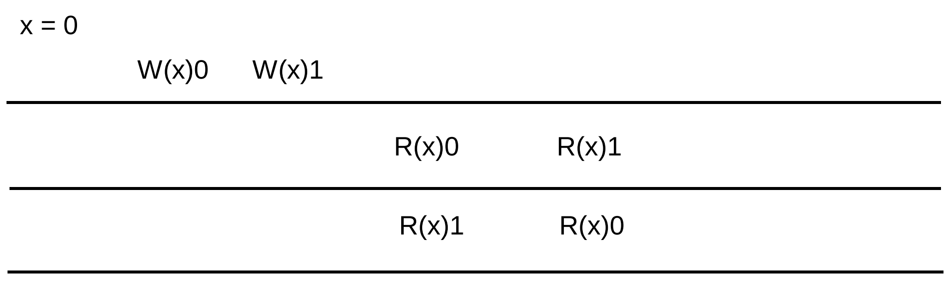
MEMORY INTEGRITY

RAM cohesion models:

1. Consistency CONTENTS - reading a memory cell returns the last written value; in distributed systems this is almost impossible to do; time required for synchronization; it is difficult to achieve consistency

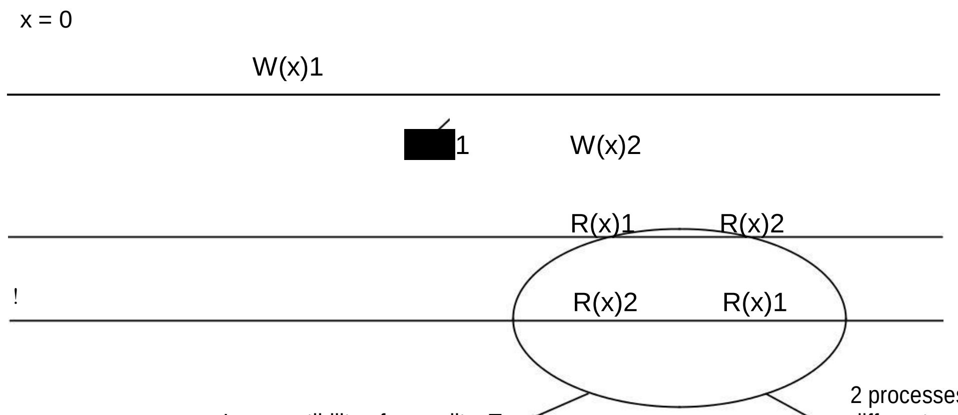


SECOVENTIONAL CONTROL - all processes see the same sequence of references to memory



There is no need to read the changed value immediately.

3. PRINCIPAL CONTROL Coherence



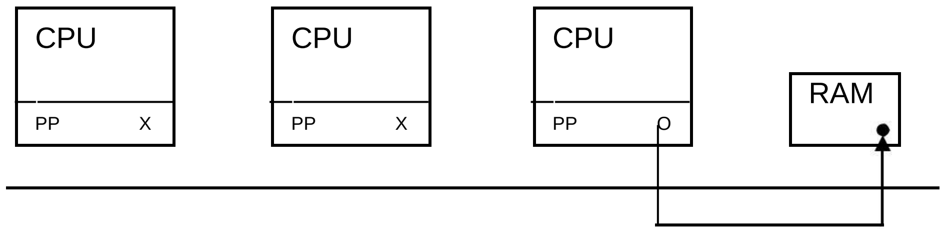
Incompatibility of causality. To there was a compliance need to be eliminated
R(x)1, then have the right.
to see the different sequences,
because I'm gonna
are cause-independent.

2 processes see different

sequence of values

Constant Consistency - There are synchronizing variables in the system that contain segments or pages from memory. No control of the sequence of events.

If synchronizing variables are changed, everyone needs to see it. Access to synchronizing variables is consistent sequentially. Memory operations associated with a synchronizing variable are suspended in the system.



PP - Memory handy, e.g. cache

It may be a system that the same memory cell may be located in multiple locations. Each processor can change something in the cell, then all the other processors become obsolete.

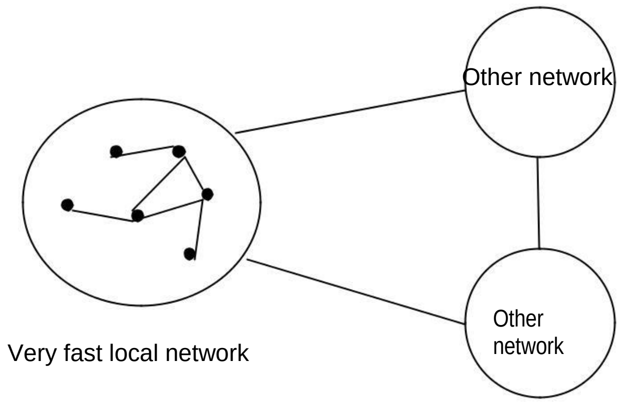
There are 2 update strategies:

1. Transcription protocol - when we are dealing with a recording system; when a change of state is detected, the processor is obliged to send copies of the new page to others.
2. Single transcription protocol - it is to limit bus transfers; each page in the memory or cache can be in 3 different states (clean, dirty, invalid):
   - Clean page - all copies are clean; when a write operation occurs, you need to update and the page changes to dirty - it is important but has no copies and initially exists only in one copy; you should ask the cache to read it
   - After the registration, all pages are dirty or even invalid (only the pages participating in the registration).

CHAPTER 9 - OTHER SYSTEMS

Characteristics of new systems:
- Based on high-speed networks
- They will be dispersed (located in the country or continent) systems
- They will be organised in such a way that a uniform memory, etc., is visible to the user.

Link bandwidth is growing much faster than computational capacity.
Fiber optic - the bandwidth depends on what is at the end of the cable and reaches 600,000 GB/sec. But we're far from that border.



GRID - systems of this class are so called; in such a network we have access to all resources regardless of where we connect to

Mechanisms for operating systems that should be adapted to such systems.

1. Assuming a free area, the routing mechanisms must work differently.
2. If we have a dependence of hardware and computing resources on the current situation in the system - global data processing (the process is performed regardless of the fact that some processor is connected to the system), lossless data storage (turning off one device can not cause data loss).

LOCATION AND NOMENCLATURE OF RESOURCES

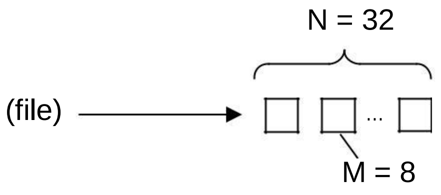
Assumptions in the GRID class system:

1. All system resources (especially files) are seen by means of GVID - global identifier of resource location.
2. It is assumed that we do not use file names and addresses, but we do use such identifiers. A HASHING FUNCTION is used to create this identifier:
   1) It's computationally easy.
   2) The result is small in terms of data volume

3) It is irreversible or difficult to reversible
4) A good hash function should have a pseudo-random value - a small change in the argument, e.g. 1 bit in a file, causes a drastic change - the result can change on all bits

a) HASHING FUNCTION H
   (date, owner, file)
b) CODING M with N - not only ensures the uniqueness of the name, but also lossless data storage

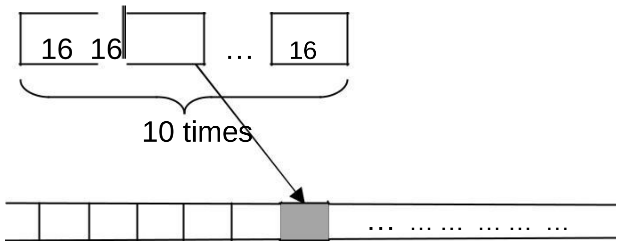
A file encoded in the form of N fragments that can be placed on any machine in the network. M of N fragments allows you to play back the file



Out of 32 machines, 25 must fail for the system to fail. If 8 always works properly, we will always get the file played back.

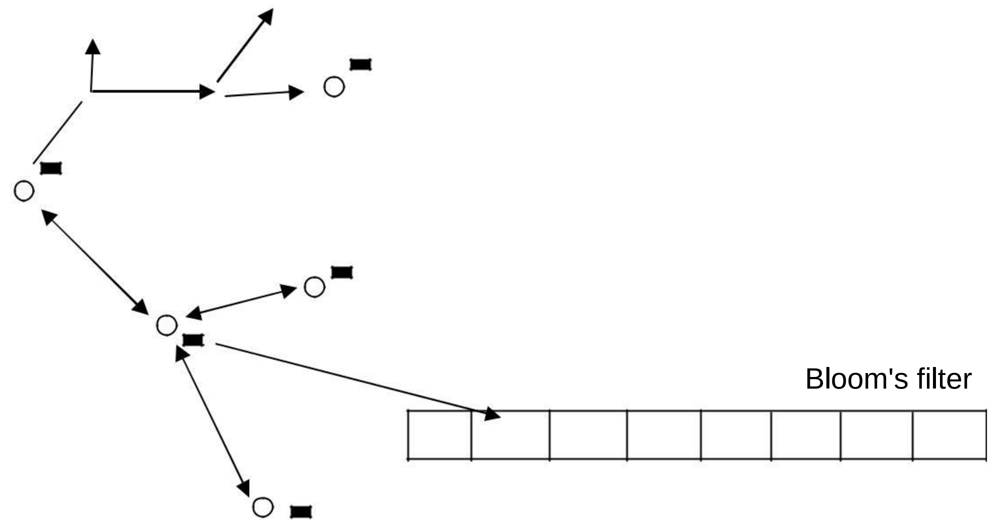
ROUTING THE LOCATION OF RESOURCES

1. Flood strategy (search in flood mode) - the most expensive solution. There is one possibility of 100% localization of resources. The searcher sends a query to all the neighbors whether they have the file, and the neighbors do the same until the file is found. A large number of messages and many machines are involved even though they do not have the required resources.
2. Probabilistic algorithm - lack of 100% effectiveness, based on cryptographic mechanism.
3. SHA-1 - secure hash algorithm - 160 bits, after obtaining global identifier of the resource we have a sequence of 160 bits, is divided into 10 portions of 16 bits each.
4. Bloom filter - 2# bits = 64 Kb - memorize the location of objects in a given network node



If we have a 160-bit resource identifier then it shows us the position in 64Kb of the string.

Most of the bits are reset to zero. The filter is able to remember the location of nearly several hundred or even several thousand objects. It's reliable, but if we write a lot to the filters, it will appear as many ones and the filter will be confused and will tell you that it always has the object.



No center. We want to transfer data about the object's connections somewhere. If a Bloom filter ( )■ is associated with each node, it will be very effective.

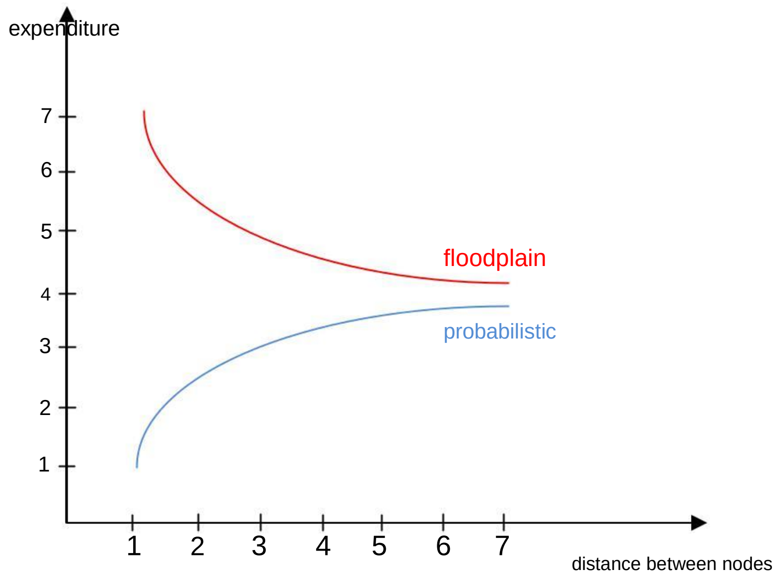
How to search for information on the scale of the whole network?

There are more Bloom filters associated with each node (often 4). They're telling us what he's like.
- probability):

- p = 1 - as in the case of nodes
- p < unity - can we find an object at a distance of 2 nodes
- p = 3 - connection at a distance of 3 knots
- p = 4 - connection at a distance of 4 knots

1. Flood search
2. Probabilistic search - we use the information contained in Bloom filters and if the system contains a lot of objects, the Bloom filter fills up with ones and often responds false - it answers that there is an object in the node, but in fact it is not there.

If the probabilistic strategy fails, we use a flood strategy.



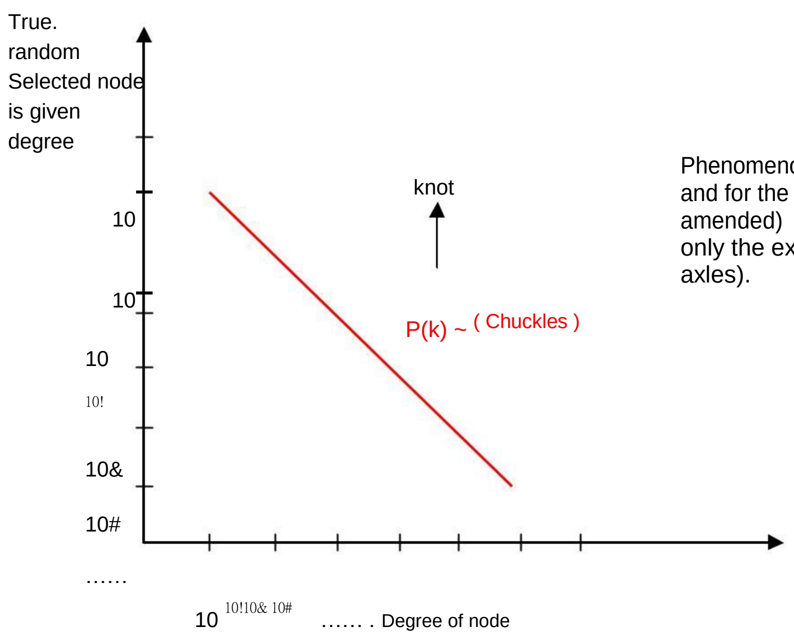
Only about 20% of people need to have a flood strategy. However, if the nodes are close together, the value is close to 0.

Summary:

- This is an architecture in which there are no management structures.
- No machine in such a system has full information about resources.
- Distance affects the efficiency of algorithms.

INTERNET MAPPING PROJECT in the 1990s

An experiment was conducted in the 1950s. Steven Milgen wrote a letter: "If you know Kowalski, send him this letter, if you don't, send it on." Then he sent this letter to several thousand people. Some of them came and some of them went missing. The average length of the list of people is 5-6. A similar length for the whole planet is estimated at about 7-8. In the case of a random graph:



Phenomenon for social networks and for the Internet (as amended) only the exponents on the axles).

In such networks there are so-called "hobbies" - nodes of a huge number of connections (neighbours).

Emergent phenomena (energent phenomenon) - it is not possible to exclude them from the description of system components.