Real-time systems
“Foundation of operating systems for soft real-time scheduling”

Mathieu Delalandre
University of Tours, Tours city, France
mathieu.delalandre@univ-tours.fr

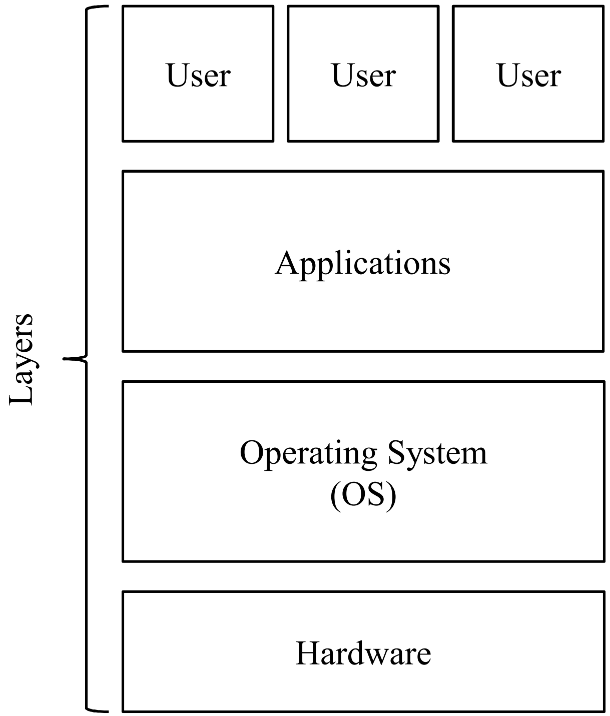
Lecture available at http://mathieu.delalandre.free.fr/teachings/realtime.html

Foundation of operating systems for soft real-time scheduling

1. Introduction
2. Process description and control
3. Short-term scheduling
   3.1. About short-term scheduling
   3.2. Context switch, quantum and ready queue
   3.3. Process and diagram models
   3.4. Scheduling algorithms
   3.5. Modeling multiprogramming
   3.6. Evaluation of algorithms
4. Soft real-time scheduling

Introduction
"Definition of OS"

Middle view: the general definition



Users: could share a same computer (through session / terminal).

Application: designed to help the user to perform a singular or multiple related specific tasks (e.g. office, programming toolkit, web browser, etc.).

Operating system: interface between the hardware and the user, responsible for the management and coordination of activities and the sharing of resources, that acts as a host for running applications on the machine.

Hardware: physical electronic components and mechanical parts that make up a piece of computer equipment (keyboard, disk drive, CPU, motherboard, etc.).

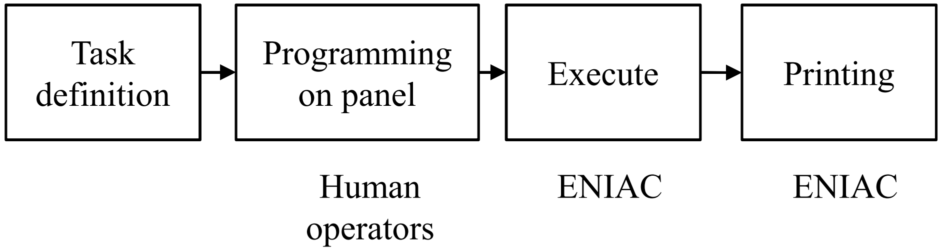
Rq. This layer separation is quite subjective

Application / OS	e.g. firewall, gesture recognition, voice command, fingerprint / eyes recognition, image indexing and retrieval, etc.
Hardware / OS	e.g. virtual memory, synchronization, DMA, SIMD / AVX instructions, GPU, bit counting, Neural Processing Unit (NPU), CPU Deep Learning instructions, etc.

Introduction
"A brief history" (1)

	Generation	Batch	Compatibility	Multi-prog	Parallelism	Microcomputer	IHM	Network	Mobile systems	Multimodality	Visualization	Quantum computer	Ubiquitous computing
1945-1955	1st												
1955-1965	2^{sd}	√											
1965-1980	3rd		√	√	√								
1980-today	4th					√	√	√					
in the queue									√	√	√	√	√

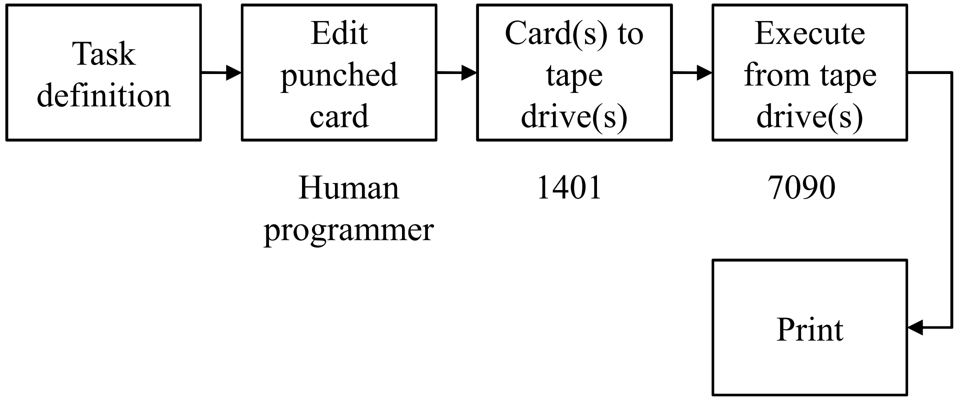
1st generation e.g. ENIAC



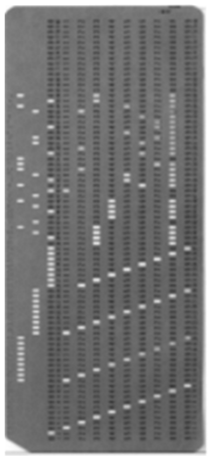
Introduction
“A brief history” (2)

	Generation	Batch	Compatibility	Multi-prog	Parallelism	Microcomputer	IHM	Network	Mobile systems	Multimodality	Visualization	Quantum computer	Ubiquitous computing
1945-1955	1st												
1955-1965	2sd	√											
1965-1980	3rd		√	√	√								
1980-today	4th					√	√	√					
in the queue									√	√	√	√	√

2sd generation e.g. IBM 1401 & 7090

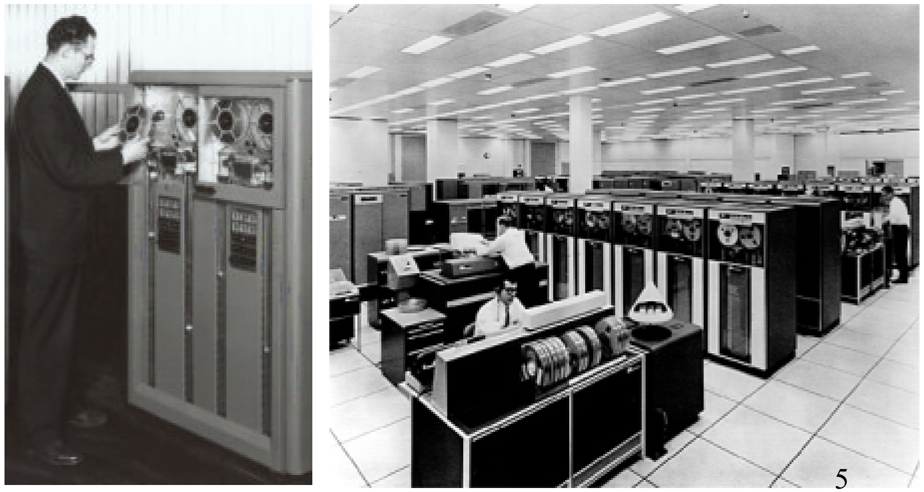


Punched card



1401

7090



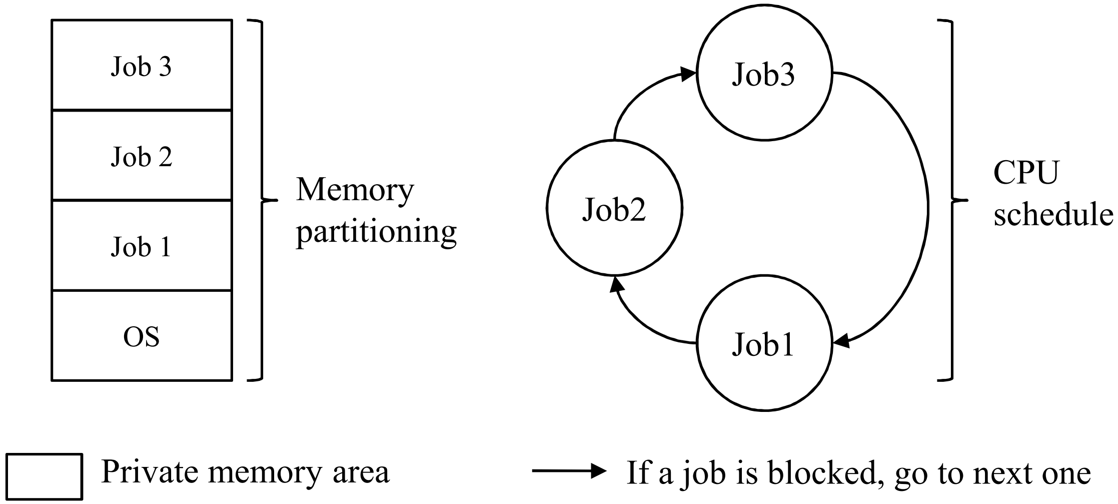
Introduction
“A brief history” (3)

	Generation	Batch	Compatibility	Multi-prog	Parallelism	Microcomputer	IHM	Network	Mobile systems	Multimodality	Visualization	Quantum computer	Ubiquitous computing
1945-1955	1st												
1955-1965	2sd	√											
1965-1980	3rd		√	√	√								
1980-today	4th					√	√	√					
in the queue									√	√	√	√	√

3rd generation e.g. IBM system/360

- ✓ a game of compatible computer 360/(A-L)
- ✓ implement multi-programming and spooling

Multiprogramming is the allocation of a computer system and its resources to more than one concurrent application / job



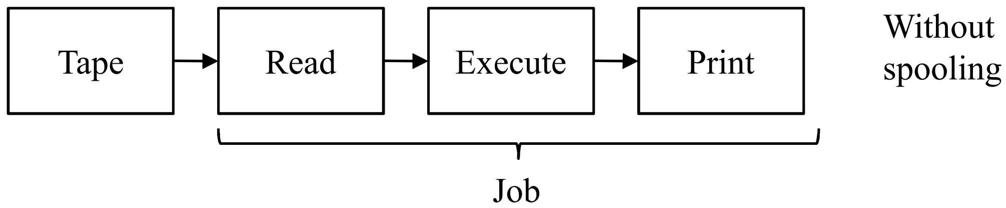
Introduction
"A brief history" (4)

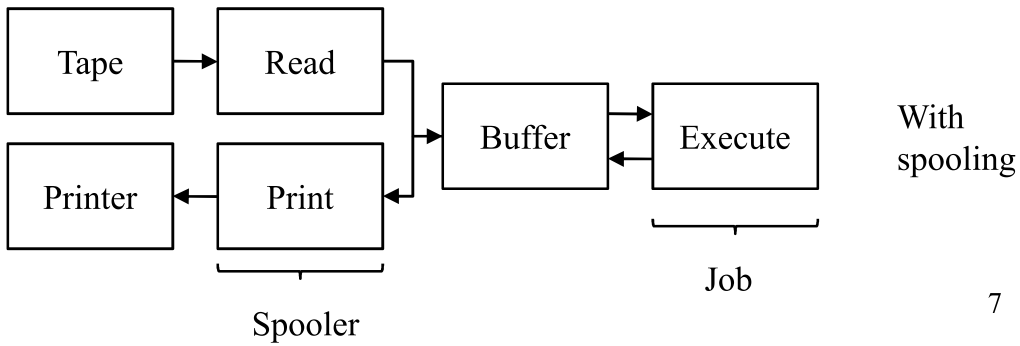
	Generation	Batch	Compatibility	Multi-prog	Parallelism	Microcomputer	IHM	Network	Mobile systems	Multimodality	Visualization	Quantum computer	Ubiquitous computing
1945-1955	1st												
1955-1965	2sd	√											
1965-1980	3rd		√	√	√								
1980-today	4th					√	√	√					
in the queue									√	√	√	√	√

3rd generation e.g. IBM system/360

- ✓ a game of compatible computer 360/(A-L)
- ✓ implement multi-programming and spooling

Spooling (simultaneous peripheral operation on-line) refers to a process of transferring data by placing it in a temporary working area where another program may access it for processing at a later point in time.





Introduction
“A brief history” (5)

	Generation	Batch	Compatibility	Multi-prog	Parallelism	Microcomputer	IHM	Network	Mobile systems	Multimodality	Visualization	Quantum computer	Ubiquitous computing
1945-1955	1st												
1955-1965	2sd	√											
1965-1980	3rd		√	√	√								
1980-today	4th					√	√	√					
in the queue									√	√	√	√	√

4th generation e.g. Personal Computer

✓ LSI (Large Scale Integration) made possible PC

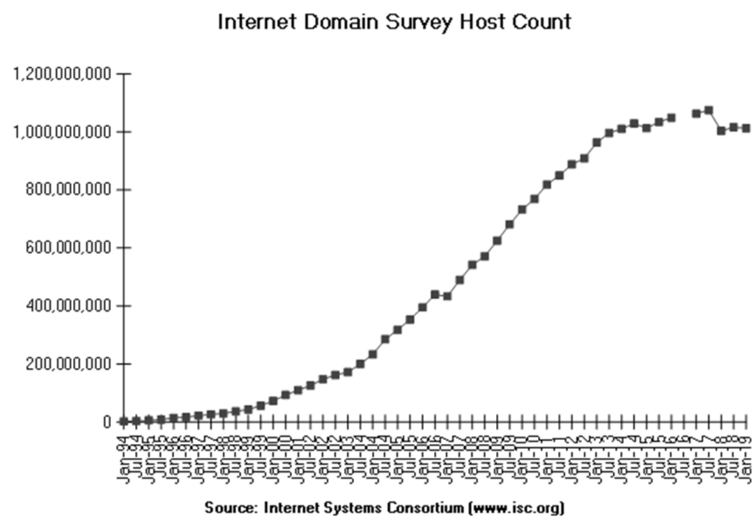
✓ Doug Engelbart proposed IHM in the early 60s, implemented in the first Apple computer in 1984

✓ Since end of 80s, Internet becomes part of the computer world

First IBM PC to laptop

First keyboard and mouse

Internet host computers



Introduction
“A brief history” (6)

	Generation	Batch	Compatibility	Multi-prog	Parallelism	Microcomputer	IHM	Network	Mobile systems	Multimodality	Visualization	Quantum computer	Ubiquitous computing
1945-1955	1st												
1955-1965	2sd	√											
1965-1980	3rd		√	√	√								
1980-today	4th					√	√	√					
in the queue									√	√	√	√	√

In the queue ….

- ✓Mobile systems
- ✓Multimodality
- ✓Visualization
- ✓Quantum computer
- ✓iOS

Mobile systems



Multimodality
“ Voice recognition”

Multimodality
“Gesture based interaction”





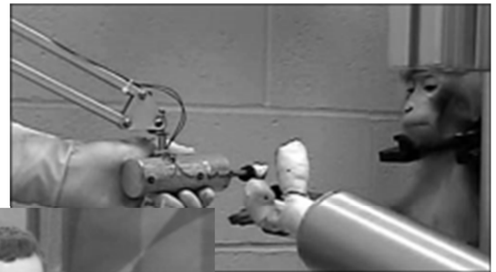
Introduction
"A brief history" (7)

	Generation	Batch	Compatibility	Multi-prog	Parallelism	Microcomputer	IHM	Network	Mobile systems	Multimodality	Visualization	Quantum computer	Ubiquitous computing
1945-1955	1st												
1955-1965	2sd	√											
1965-1980	3rd		√	√	√								
1980-today	4th					√	√	√					
in the queue									√	√	√	√	√

In the queue ….

- ✓Mobile systems
- ✓Multimodality
- ✓Visualization
- ✓Quantum computer
- ✓iOS

Multimodality
"Brain computer interface"



Multimodality
"Eye tracker"

Introduction
“A brief history” (8)

	Generation	Batch	Compatibility	Multi-prog	Parallelism	Microcomputer	IHM	Network	Mobile systems	Multimodality	Visualization	Quantum computer	Ubiquitous computing
1945-1955	1st												
1955-1965	2sd	√											
1965-1980	3rd		√	√	√								
1980-today	4th					√	√	√					
in the queue									√	√	√	√	√

In the queue ….

- ✓Mobile systems
- ✓Multimodality
- ✓Visualization
- ✓Quantum computer
- ✓iOS

Visualization
“Electronic paper”

Visualization
“3D screen”

Visualization
“real virtuality”

Visualization
“Augmented reality and tagging the real world”

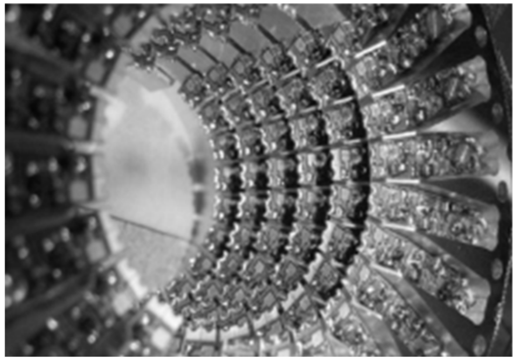
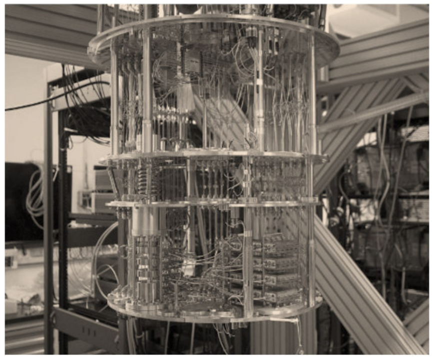
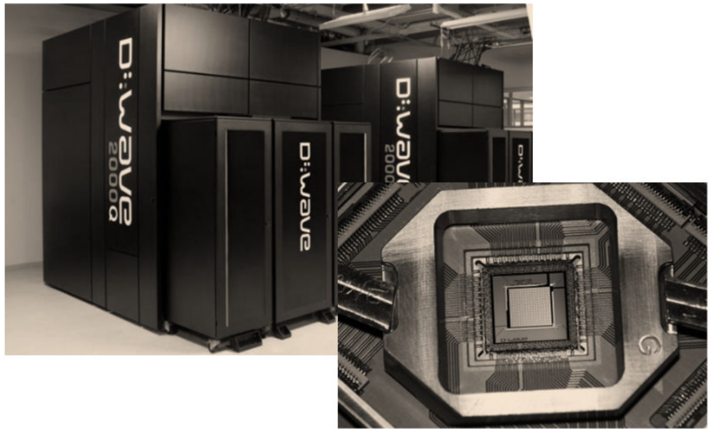
Introduction
"A brief history" (9)

	Generation	Batch	Compatibility	Multi-prog	Parallelism	Microcomputer	IHM	Network	Mobile systems	Multimodality	Visualization	Quantum computer	Ubiquitous computing
1945-1955	1st												
1955-1965	2sd	√											
1965-1980	3rd		√	√	√								
1980-today	4th					√	√	√					
in the queue									√	√	√	√	√

In the queue ….

- ✓Mobile systems
- ✓Multimodality
- ✓Visualization
- ✓Quantum computer
- ✓iOS

Quantum computer

Introduction
“A brief history” (10)

	Generation	Batch	Compatibility	Multi-prog	Parallelism	Microcomputer	IHM	Network	Mobile systems	Multimodality	Visualization	Quantum computer	Ubiquitous computing
1945-1955	1st												
1955-1965	2sd	√											
1965-1980	3rd		√	√	√								
1980-today	4th					√	√	√					
in the queue									√	√	√	√	√

In the queue ….

- ✓Mobile systems
- ✓Multimodality
- ✓Visualization
- ✓Quantum computer
- ✓iOS

iOS (Artificial) intelligence Operating System is a system that manages computer software and hardware and provides common service for the computer using its intelligence by a computer or a machine in order to solve complex problems with ease.

Introduction
“Taxonomy of OS”

OS depends of computer application

- **Mainframes** are powerful computers used mainly by large organizations for critical applications, typically bulk data processing such as census, industry and consumer statistics, etc.
- **Server computers** link other computers or electronic devices together. They often provide essential services across a network, either to private users inside a large organization or to public users via the Internet.
- **Multicomputers** offer a major-league computer power by connecting multiple CPUs/GPUs together (e.g. a workstation). They need a special OS support for communication, connectivity and consistency.
- **Personal Computers (PC)** are any general-purpose computers whose sizes, capabilities, and original sale prices make them useful for individuals, and which is intended to be operated directly by an end user with no intervening computer operator.
- **Real-Time Systems (RTS)** implement hardware and software components that are subject to real-time constraints i.e. operational deadlines from events to system responses.
- **Embedded systems** are designed to perform one or a few dedicated functions often with real-time computing constraints. They are embedded as part of a complete device often including hardware and mechanical parts.
- **Mobile systems** include personal digital assistants (PDA) or cellular telephones, many of which use special purpose embedded systems.

	Ergonomics	Communication	Robustness	Optimization
Mainframes			√	√
Servers		√	√	
Multicomputers			√	√
PC	√			
RTS			√	√
Embedded systems		√	√	√
Mobile systems	√	√		

√ a major feature of concerned OS

Foundation of operating systems for soft real-time scheduling

1. Introduction
2. Process description and control
3. Short-term scheduling
    3.1. About short-term scheduling
    3.2. Context switch, quantum and ready queue
    3.3. Process and diagram models
    3.4. Scheduling algorithms
    3.5. Modeling multiprogramming
    3.6. Evaluation of algorithms
4. Soft real-time scheduling

Process description and control (1)

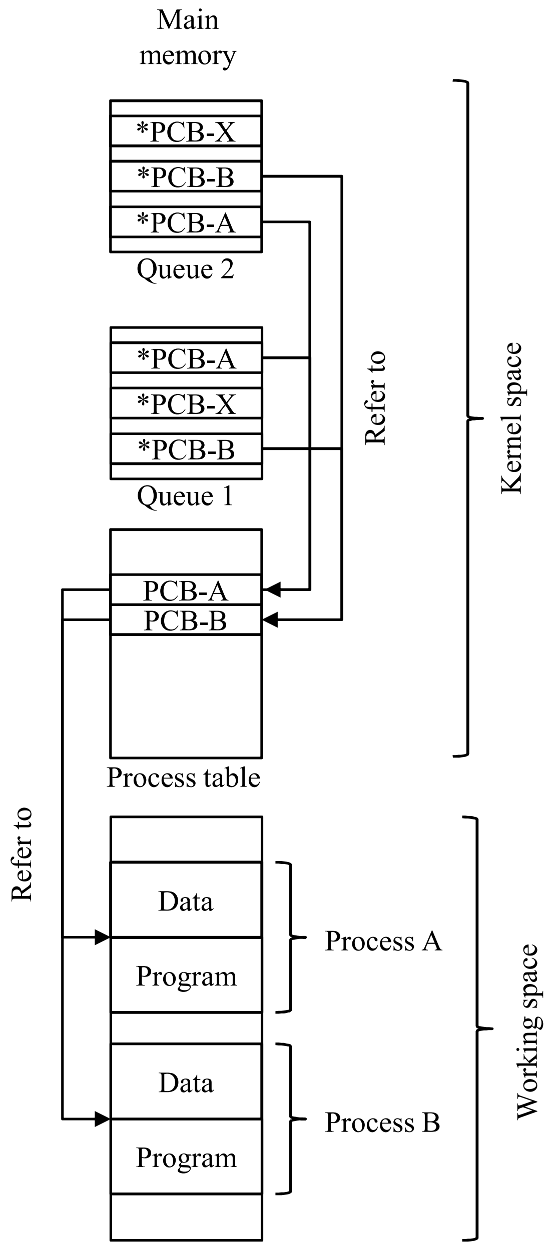
A process(us) is an instance of a computer program that is being executed. It contains the program code and the liked data.

PCB "Process Control Block" (i.e. Task Controlling Block or Task Structure) is a data structure in the operating system kernel containing the information needed to manage a particular process.

Processus Table is an area of memory protected from normal user access, to manage the PCBs, as they contain critical information for processes.

A thread takes part of a process but it has its own program counter, stack and registers. The threads belonging to a process share common code and data.

TCB "Thread Control Block" is a data structure in the operating system kernel containing the information needed to manage a particular thread (PCB look-like).



Process description and control (2)

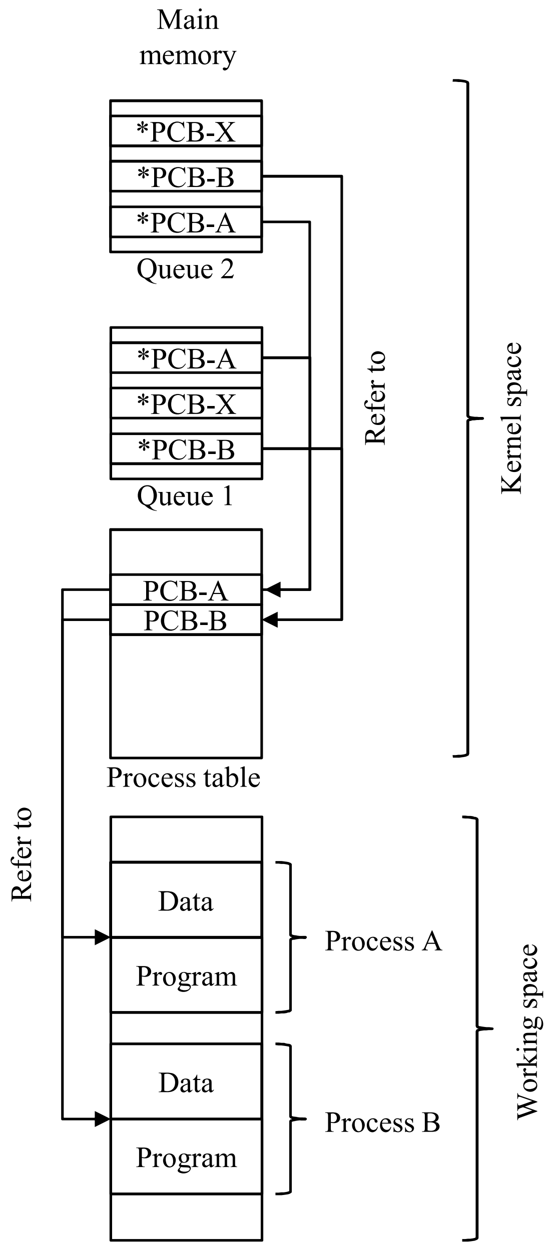
List of frequent data appearing in a PCB

✓**Process identifier (pid)** refers the process in the OS.
✓**Group data**, hierarchy information (e.g. parents and childs), type of process and group memberships.
✓**CPU-scheduling information** e.g. process priority, pointers to scheduling queues, etc.
✓**Process state** e.g. ready, running, waiting, terminated, etc.
✓**Program counter (PC)** refers the current execution state of the process.
✓**CPU registers** correspond to the current state of the CPU.
✓**Security attributes** refer the owner or set of permissions (allowable operations) of the process.
✓**Accounting information** e.g. start time, end time, amount of CPU used, real-time used, etc.
✓**Etc.**

related to process management

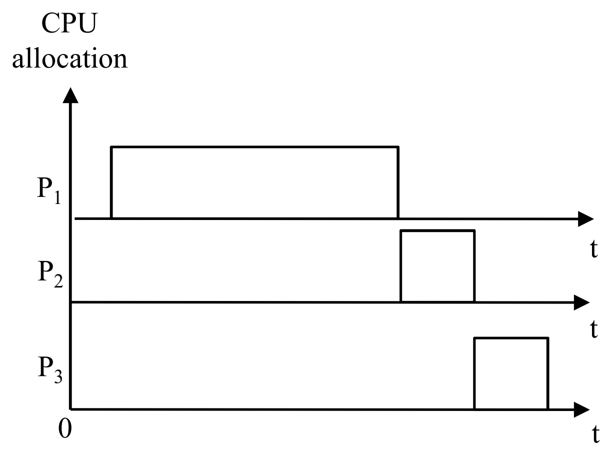
✓**Memory management information** includes page and segment tables on the executable code, call stack (to keep track of active subroutines and/or other events), etc.
✓**Operating system descriptors** refer to the resources that are allocated to the process, such as files, devices, other data sources.
✓**Etc.**

related to data management

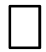


Process description and control (3)

Multitasking (i.e. multiprogramming) is a method by which multiple tasks share common processing resources such as a CPU. With a single CPU, only one task can run at any time. Multitasking solves the problem by scheduling the tasks i.e. which task must run on the CPU, and which task must wait.



scheduling diagram of processes



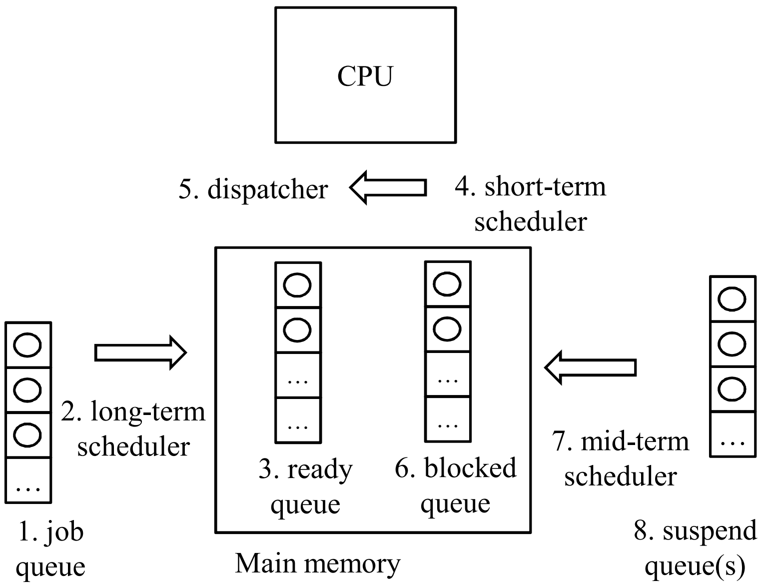
The process P_i is running

Process description and control (4)

Scheduling refers to the way processes are assigned to run on the CPU. The aim of scheduling is to assign processes to be executed by the processor over the time, in a way that meets objectives of the system, such as the response time, throughput and processor efficiency.

In many systems, the scheduling activity is broken into three separate functions: long, medium and short-term scheduling.

Scheduling affects the performance of the system because it determines which processes will wait and which will progress. Scheduling is a matter of managing queues to minimize queuing delay and to optimize performance in a queuing environment.



Process description and control (5)

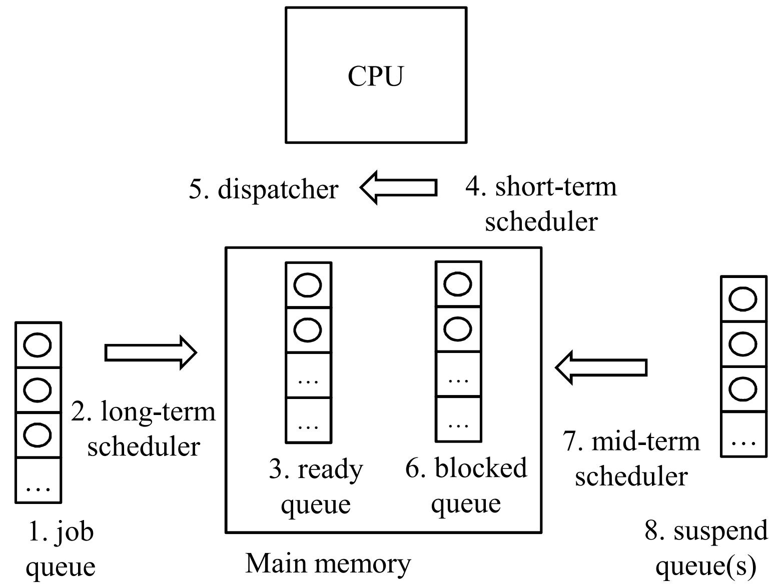
1. Job queue stores processes to enter in the system, they are put into the job queue. The job queue contains the list of processes to create.

2. Long term scheduler (admission scheduler) decides which processes are to be admitted to the ready queue, they are then created and loaded into the main memory.

3. Ready queue is a data structure to keep in the main memory the processes that are in a ready state.

4. Short-term scheduler (i.e. CPU scheduler) decides which of the ready, in-memory processes, are to be executed (allocated to the CPU) following a clock interrupt, an I/O interrupt, an operating system call or ayn other form of signal.

5. Dispatcher gives the control of the CPU to the process selected by the short-term scheduler.

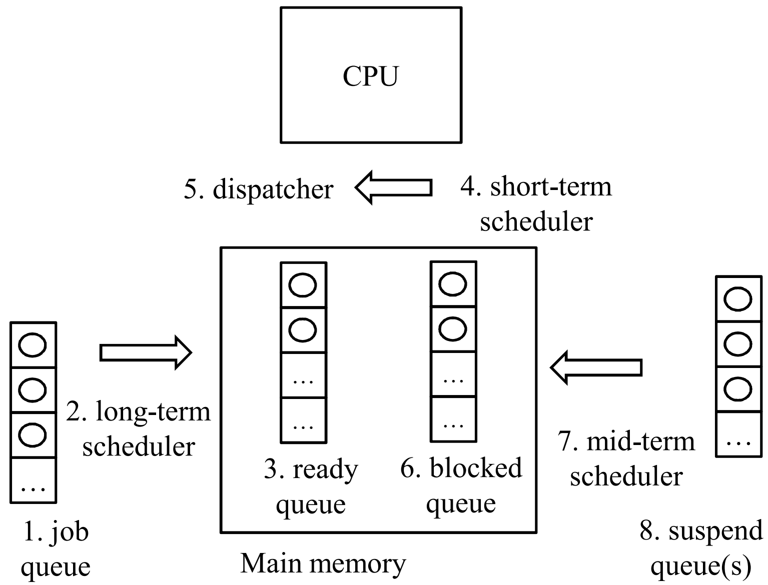


Process description and control (6)

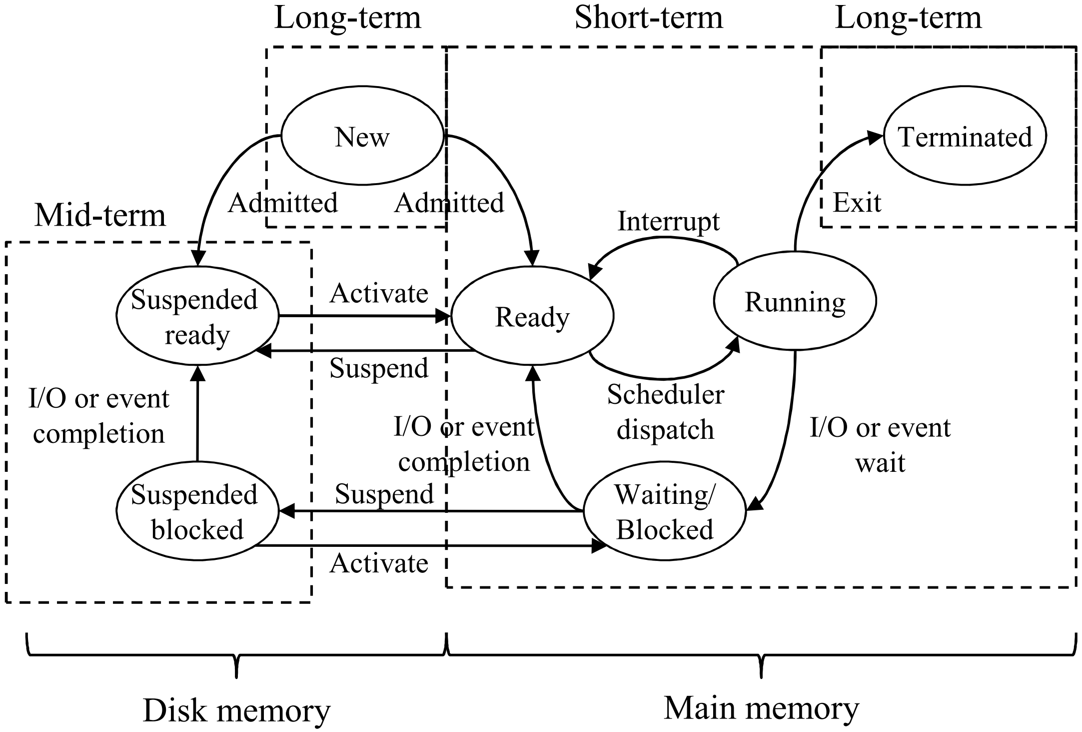
6. Waiting/Blocked queue is a data structure to keep in the main memory the processes in a blocked state.

7. Mid-term scheduler removes processes from the main memory (if full) and places them on a secondary memory (such as a disk drive) and vice-versa.

8. Blocked suspend queue(s) contain lists of processes moved to the disk (i.e. swapping). Two queues are usually managed, related to the processes in a suspended-blocked or a suspended-ready state.



Process description and control (7)



As a process executes, it changes its state. The state of a process is defined in part by the current activity of the process.

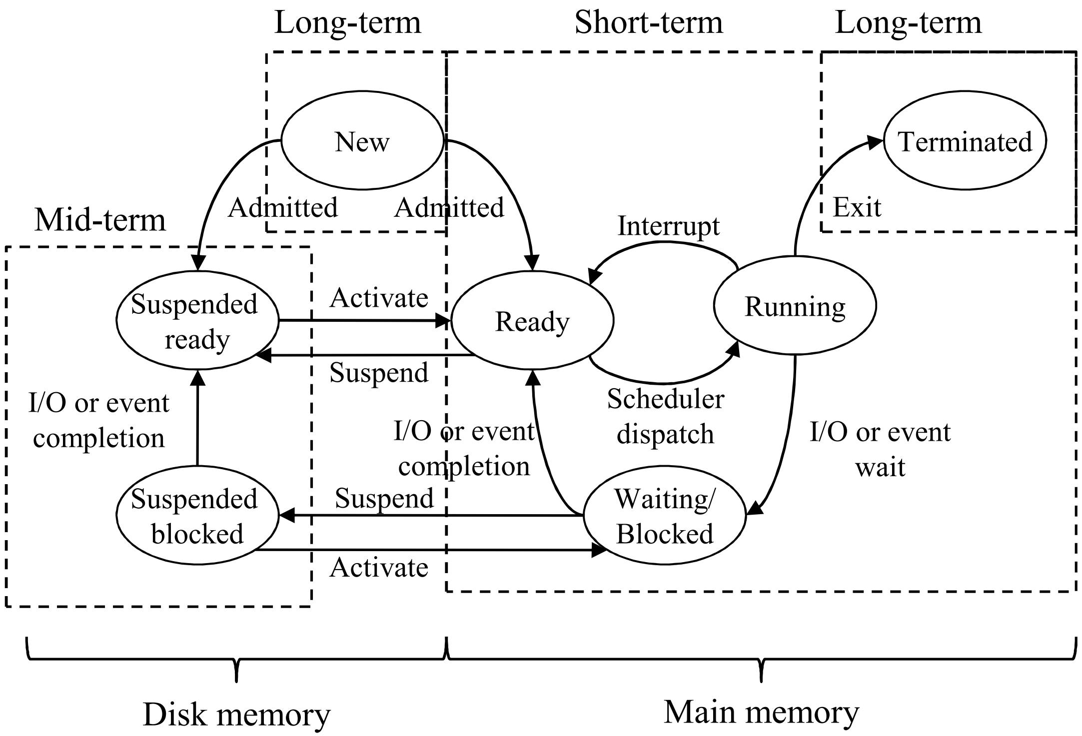
✓**New:** in this state, the process awaits for an admission to the ready state. This admission will be approved or delayed by a long-term, or admission, scheduler.

✓**Ready**: a ready process has been loaded into the main memory and the ready queue and is awaiting for an execution on the CPU (to be loaded into the CPU by the dispatcher following the decision of the short-term scheduler).

✓**Running:** process is being executed by CPU.

✓**Terminated:** a process may be terminated, either from the running state by completing its execution or by explicitly being killed. If a process is not removed from the memory, this state may also be called zombie.

Process description and control (8)



As a process executes, it changes its state. The state of a process is defined in part by the current activity of the process.

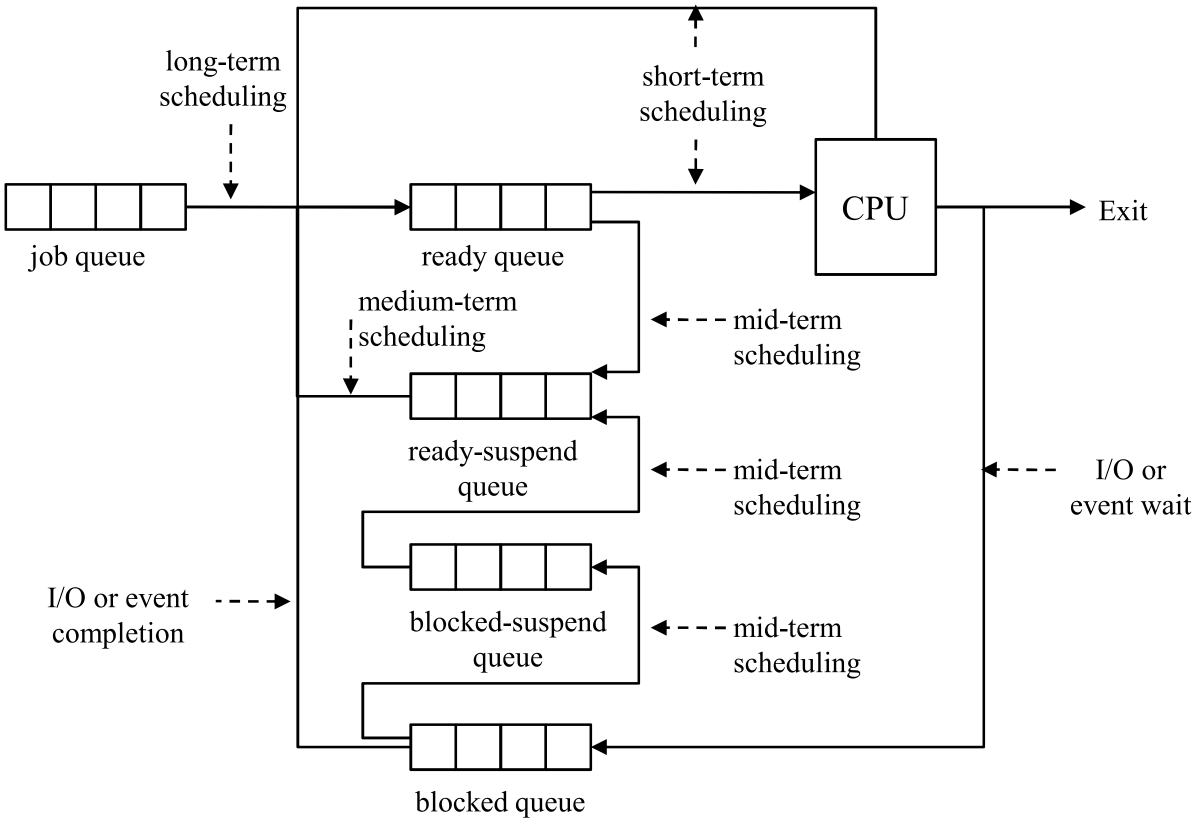
✓**Waiting/Blocked:** a process that cannot execute until some events occurs, such as the completion of an I/O operation or a signal. Every blocked process is moved to the blocked queue.

✓**Suspended blocked:** a process is put in the disk memory by the mid-term scheduler (i.e. swapping out).

✓**Suspended ready:** a process is ready to be loaded from the disk to the main memory (i.e. swapping in).

Process description and control (9)

Queuing diagram for scheduling shows the queues involved in the state transitions of processes.
Rq. For simplicity, this diagram shows new processes going directly to the ready state without the option of either the ready state or either the ready/suspend state.



Process description and control (10)

✓**New → Ready:** the OS will move a process from the new state to the ready state (i.e. from the job queue to the ready queue) when it is prepared to take an additional process. Most of the systems set some limits based on the number of existing processes in memory.

✓**Ready → Running:** when it is time to select a process to run, the OS chooses one of the processes in the ready state. This is the job of the scheduler.

✓**Running → Terminated:** the currently running process is terminated by the OS if the process indicates that it has completed, or if it aborts.

✓**Running → Ready:** the most common reasons for this transition are

(1) in the case of a preemptive scheduling, the OS assigns different levels of priority to different processes; thus a process A can preempt a process B and B will go to the ready state and shift to the ready queue.

(2) the running process has reached the maximum allowable time for an uninterrupted execution (all the multiprogramming OS impose this type of time discipline).

(3) a process may voluntarily release the control of the processor (e.g. a background process that performs some accounting or maintenance functions periodically).

Process description and control (11)

✓**Running → Blocked:** a process is put in the blocked state (and moves to the blocked queue) if

(1) it requests something (i.e. a resource) for which it must wait such as a file, a shared section, etc. that is not immediately available (e.g. a down operation on a Mutex).

(2) it requests a service to the OS that is not prepared to perform immediately. A request to the OS is usually in the form of a system service call; that is; a call from the running program to a procedure that is part of the OS.

Blocked → Running: a process in the blocked state is moved to the ready state (and moved to the ready queue) when the event for which it has been waiting occurs (e.g. up operation on a Mutex, system call return, etc.).

Ready → Terminated: this transition is not shown on the state and queuing diagrams, in some systems, a parent may terminate a child process at any time. Also, if a parent terminates, all child processes associated wit that parent may terminate.

Terminated → Ready: this transition has nosense.

Foundation of operating systems for soft real-time scheduling

1. Introduction
2. Process description and control
3. Short-term scheduling
   3.1. About short-term scheduling
   3.2. Context switch, quantum and ready queue
   3.3. Process and diagram models
   3.4. Scheduling algorithms
   3.5. Modeling multiprogramming
   3.6. Evaluation of algorithms
4. Soft real-time scheduling

About short-term scheduling (1)

(Short-term) scheduler is a system process running an algorithm to decide which of the ready processes are to be executed (i.e. allocated to the CPU). Different performance criteria have to be considered:

Performance criteria related to the user:

- ✓Response time: total time between submission of a request and its completion
- ✓Predictability: to predict execution time of processes and avoid wide variations in response time

Performance criteria related to the system:

- ✓Waiting time: amount of time a process has been waiting in the ready queue
- ✓Throughput: number of processes that complete their execution per time unit
- ✓CPU utilization: to keep the CPU as busy as possible
- ✓Fairness : a process should not suffer of starvation i.e. never loaded to CPU
- ✓Enforcing priorities: when processes are assigned with priorities, the scheduling policy should favor the high priority processes
- ✓Balancing resources: the scheduling policy should keep the resources of the system busy
- ✓Etc.

About short-term scheduling (2)

Depending of the considered systems (mainframes, server computers, personal computers, real-time systems, embedded systems, etc.), different scheduling problems have to be considered:

Scheduling problems	Parameters		Standard parameters in a time-sharing system
algorithm's features	on-line	off-line	on-line
	preemptive	no preemptive	both
	relative deadline	strict deadline	relative deadline
	static priority	dynamic priority	both
	optimal	not optimal	both
Processus model	independants	dependants	both
	without resource	with resource	both
	aperiodic	periodic	aperiodic
Type of system	mono-core	multi-core	mono-core
	centralized	distributed	centralized

About short-term scheduling (3)

Depending of the considered systems (mainframes, server computers, personal computers, real-time systems, embedded systems, etc.), different scheduling problems have to be considered:

- ✓**On-line/off-line:** off-line scheduling builds complete planning sequences with all the parameters of the process. The schedule is known before the process execution and can be implemented efficiently.

- ✓**Preemptive/non-preemptive:** in a preemptive scheduling, an elected process may be preempted and the processor allocated to a more urgent process with a higher priority.

- ✓**Relative/strict deadline:** a process is said with no (or a relative) deadline if its response time doesn't affect the performance of the system and jeopardize the correct behavior.

- ✓**Dynamic/static priority:** static algorithms are those in which the scheduling decisions are based on fixed parameters, assigned to processes before their activation. Dynamic scheduling employs parameters that may change during the system evolution.

- ✓**Optimal:** an algorithm is said optimal if it minimizes a given cost function.

About short-term scheduling (4)

Depending of the considered systems (mainframes, server computers, personal computers, real-time systems, embedded systems, etc.), different scheduling problems have to be considered:

✓**Dependent /independent process:** a process is dependent (or cooperating) if it can affect (or be affected by) the other processes. Clearly, any process than share data and uses IPC is a cooperating process.

✓**Resource sharing:** from a process point of view, a resource is any software structure that can be used by the process to advance its execution.

✓**Periodic/aperiodic process:** a process is said periodic if, each time it is ready, it releases a periodic request.

✓**Mono-core / Multi-core:** when a computer system contains a set of processor that share a common main memory, we're talking about a multiprocessor /multi-core scheduling.

✓**Centralized/distributed:** scheduling is centralized when it is implemented on a standalone architecture. Scheduling is distributed when each site defines a local scheduling, and the cooperation between sites leads to a global scheduling strategy.

About short-term scheduling (5)

The general algorithm of a short-term scheduler is

While

1. A timer interrupt causes the scheduler to run once every time slice
2. Data acquisition (i.e. to list processes in the ready queue and update their parameters)
3. Selection of the process to run based on the scheduling criteria of the algorithm
4. If the process to run is different of the current process, to order to the dispatcher to switch the context
5. System execution will go on …

The real problem with the scheduling is the definition of the scheduling criteria, algorithm is little discussed.

Foundation of operating systems for soft real-time scheduling

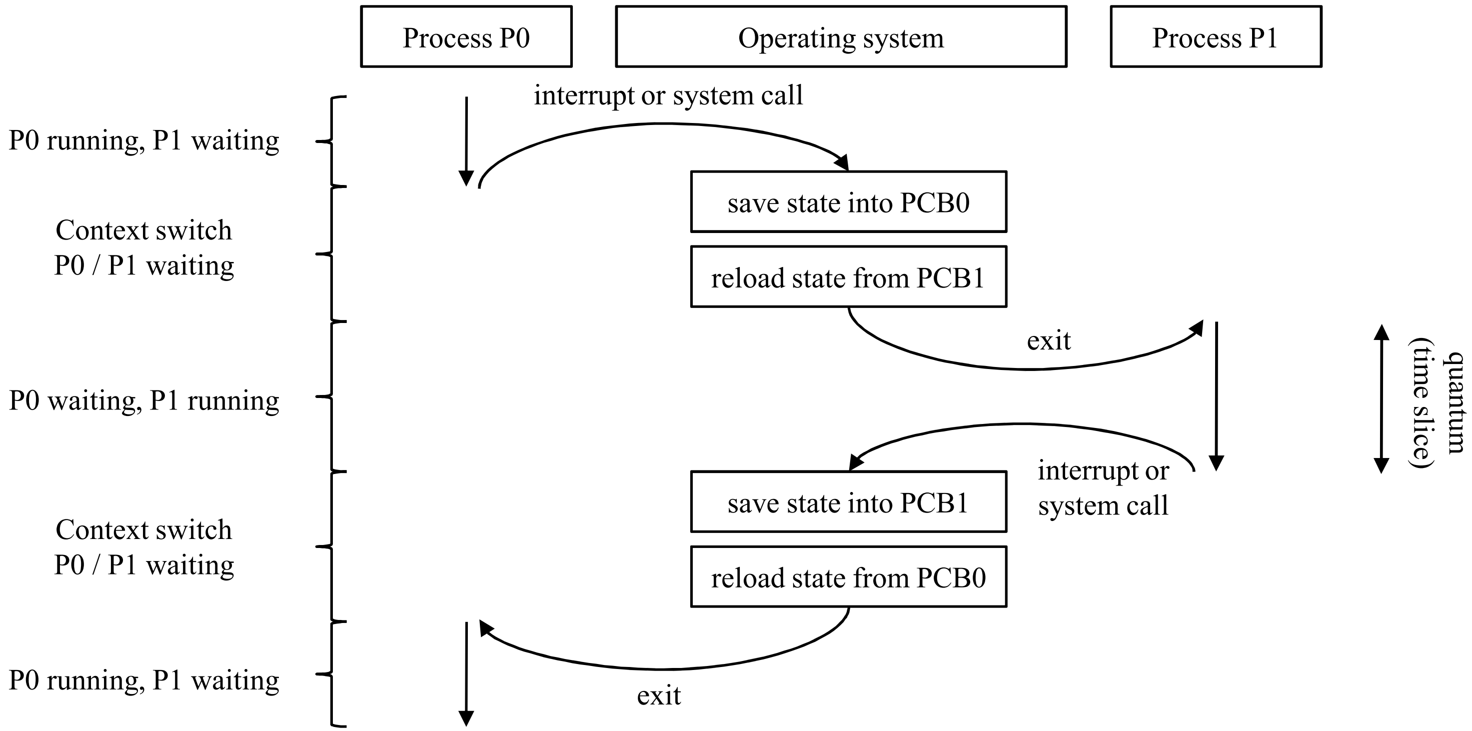
1. Introduction
2. Process description and control
3. Short-term scheduling
   3.1. About short-term scheduling
   3.2. Context switch, quantum and ready queue
   3.3. Process and diagram models
   3.4. Scheduling algorithms
   3.5. Modeling multiprogramming
   3.6. Evaluation of algorithms
4. Soft real-time scheduling

Context switch, quantum and ready queue (1)

Dispatcher is in charge of passing the control of the CPU to the process selected by the short-term scheduler.

C**ontext switch** is the operation of storing and restoring state (context) of a CPU so that the execution can be resumed from the same point at a later time. It is based on two distinct sub-operations, state safe and state restore. Switching from one process to another requires a certain amount of time (saving and loading the registers, the memory maps, etc.).

Quantum (or time slice) is the period of time for which a process is allowed to run in a preemptive multitasking system. The scheduler is run once every time slice to choose the next process to run.



Context switch, quantum and ready queue (2)

e.g. We consider the case of

i. Three processes A, B, C and a dispatcher which traces (i.e. instructions listing), given in the next table.

Process A	Process B	Process C	Dispatcher
5000	8000	12000	100
5001	8001	12001	101
….	8002	…	…
5011	8003	12011	105

ii. Processes are scheduled in a predefined order (A, B then C)

iii. The OS here only allows a process to continue for a maximum of six instruction cycles (the quantum), after which it is interrupted.

Cycle	Instructions	
1	5000	A starts
2	5001	
3	5002	
4	5003	Process A
5	5004	
6	5005	A interrupted
7	100	
8	101	
9	102	Dispatcher
10	103	
11	104	
12	105	
13	8000	B starts
14	8001	Process B
15	8002	
16	8003	B ends
17	100	
18	101	
19	102	Dispatcher
20	103	
21	104	
22	105	
23	12000	C starts
24	12001	
25	12002	Process C
26	12003	
27	12004	
28	12005	C interrupted

Context switch, quantum and ready queue (3)

e.g. We consider the case of

i. Three processes A, B, C and a dispatcher which traces (i.e. instructions listing), given in the next table.

Process A	Process B	Process C	Dispatcher
5000	8000	12000	100
5001	8001	12001	101
….	8002	…	…
5011	8003	12011	105

ii. Processes are scheduled in a predefined order (A, B then C)

iii. The OS here only allows a process to continue for a maximum of six instruction cycles (the quantum), after which it is interrupted.

Cycle	Instructions
29	100
30	101
31	102
32	103
33	104
34	105
35	5006
36	5007
37	5008
38	5009
39	5010
40	5011
41	100
42	101
43	102
44	103
45	104
46	105
47	12006
48	12007
49	12008
50	12009
51	12010
52	1211

Dispatcher (cycles 29–34)

A continues

Process A (cycles 35–40)

A ends

Dispatcher (cycles 41–46)

C continues

Process C (cycles 47–52)

C ends

Context switch, quantum and ready queue (4)

e.g. We consider the case of

i. Three processes A, B, C and a dispatcher which traces (i.e. instructions listing), given in the next table.

Process A	Process B	Process C	Dispatcher
5000 5001 …. 5011	8000 8001 8002 8003	12000 12001 … 12011	100 101 … 105

ii. Processes are scheduled in a predefined order (A, B then C)

iii. The OS here only allows a process to continue for a maximum of six instruction cycles (the quantum), after which it is interrupted.

Quantum	<	**i**	**i+1**	**i+2**	**i+3**	**i+4**
Instruction cycle	Na	6	4	6	6	6
Scheduled process by the CPU	Na	A	B	C	A	C
Ready queue state	A B C	B C	C A	A	C	

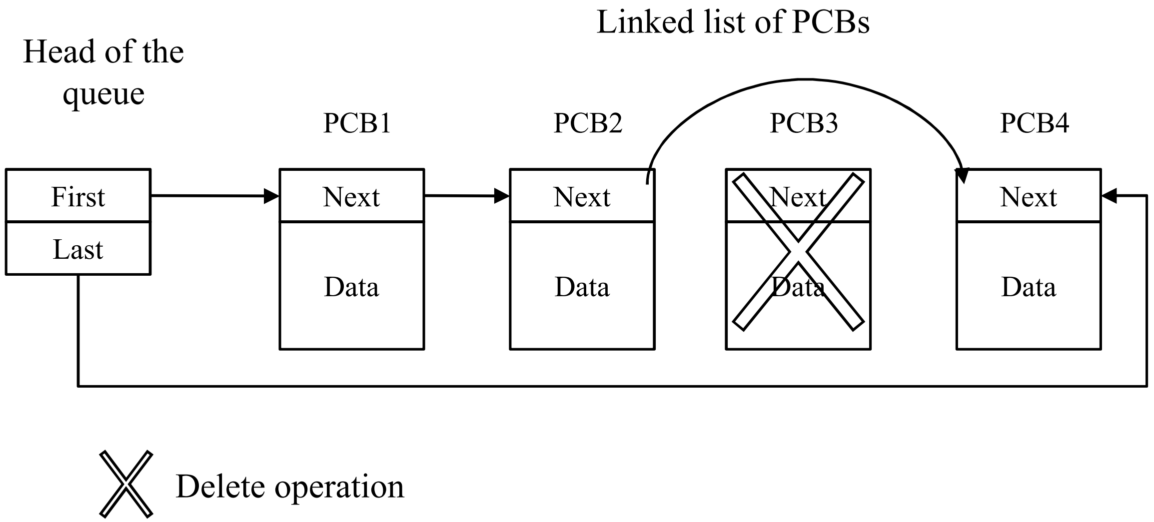
5 quanta / 4 context switches (n-1 quanta)
28 process instruction (6+4+6+6+6)
6×4=24 dispatcher instructions
a maximum of two processes in the ready queue

The length of the quantum can be critical to balance the system performance vs. process responsiveness.

- If the quantum is too short then the scheduler will consume too much processing time.
- If the quantum is too long, processes will take longer to respond to inputs.

Context switch, quantum and ready queue (5)

The **ready queue** is a huge-data list generally composed of PCB pointers, it is stored as a linked list in the main memory, managing pointers from the first to the last PCB.



First, last and next are PCB pointers in the list.

If we delete a PCB (i), pointer of the previous PCB (i-1) jumps to next one (i+1) i.e. it is not necessary to fill the empty space or to move (copy) the PCBs.

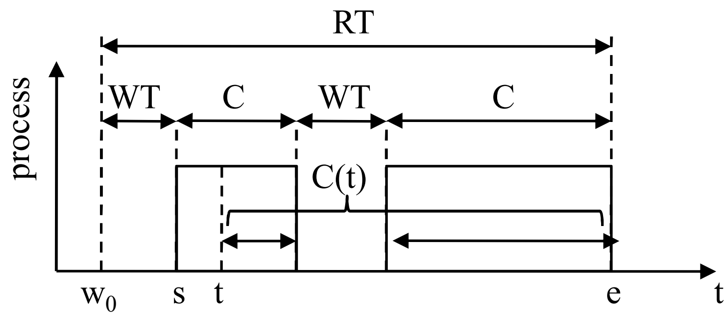
Foundation of operating systems for soft real-time scheduling

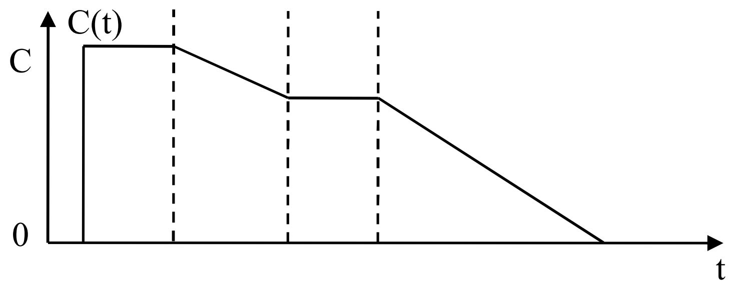
1. Introduction
2. Process description and control
3. Short-term scheduling
   3.1. About short-term scheduling
   3.2. Context switch, quantum and ready queue
   3.3. Process and diagram models
   3.4. Scheduling algorithms
   3.5. Modeling multiprogramming
   3.6. Evaluation of algorithms
4. Soft real-time scheduling

Process and diagram models (1)

Process model and context parameters

PID	process number	Process parameters
rank	rank in the ready queue	
w_0	wakeup time	
C	capacity	
P	priority	
s	start time (run as a first time)	context parameters
e	end time (termination)	
$RT = e - w_0$	response time	
$WT = RT - C$	waiting time	
$C(t)$	residual capacity at t $C(w_0) = C$, $C(e)=0$	
$T(t)=C-C(t)$	CPU time consumed at t $T(w_0)=0$, $T(e) = C$,	
$E(t)=t-w_0$	CPU time entitled $E(w_0)=0$, $E(e)=RT$	
$WT(t)=E(t)-T(t)$	waiting time at t $WT(w_0)=0$, $WT(e)=WT$	





Process and diagram models (2)

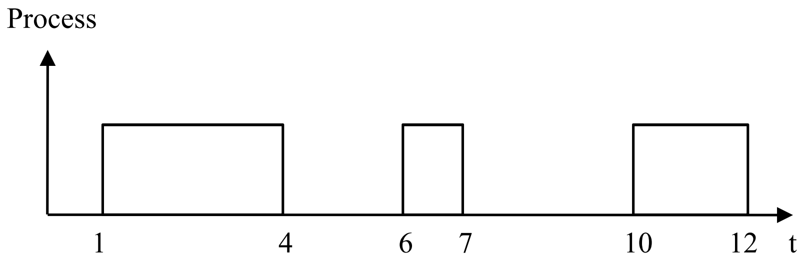
Process model and context parameters

Process parameters:

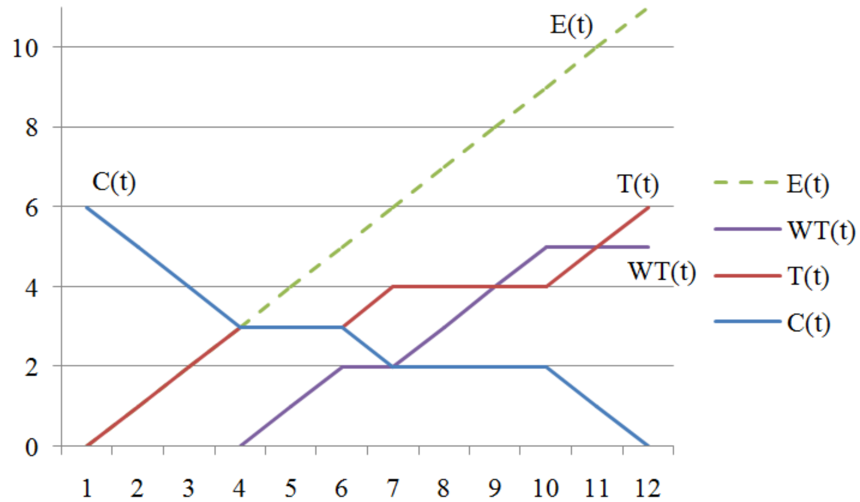
PID	process number
rank	rank in the ready queue
w_0	wakeup time
C	capacity
P	priority

context parameters:

s	start time (run as a first time)
e	end time (termination)
$RT = e - w_0$	response time
$WT = RT - C$	waiting time
C(t)	residual capacity at t $C(w_0) = C$, $C(e)=0$
$T(t)=C-C(t)$	CPU time consumed at t $T(w_0)=0$, $T(e) = C$,
$E(t)=t-w_0$	CPU time entitled $E(w_0)=0$, $E(e)=RT$
$WT(t)=E(t)-T(t)$	waiting time at t $WT(w_0)=0$, $WT(e)=WT$



$\mathbf{w_0}$	1 (if = s)	**s**	1
C	6	**e**	12
P	Na	**RT**	12-1 = 11
		WT	11-6 = 5



Process and diagram models (3)

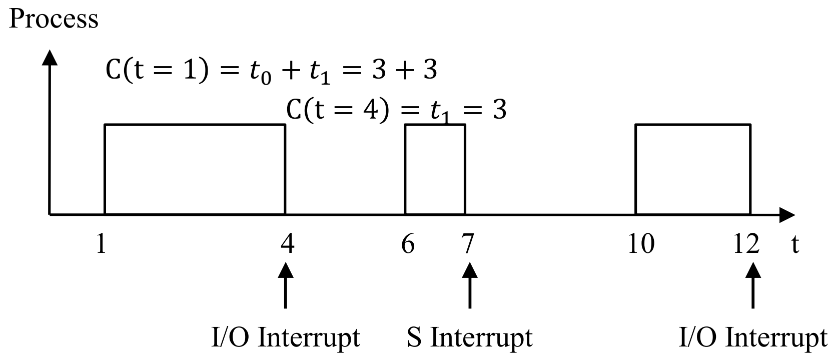
Process model and context parameters

CPU burst time is an assumption of how long a process requires the CPU between I/O waits. It means the amount of time that a process uses the CPU without interruption.

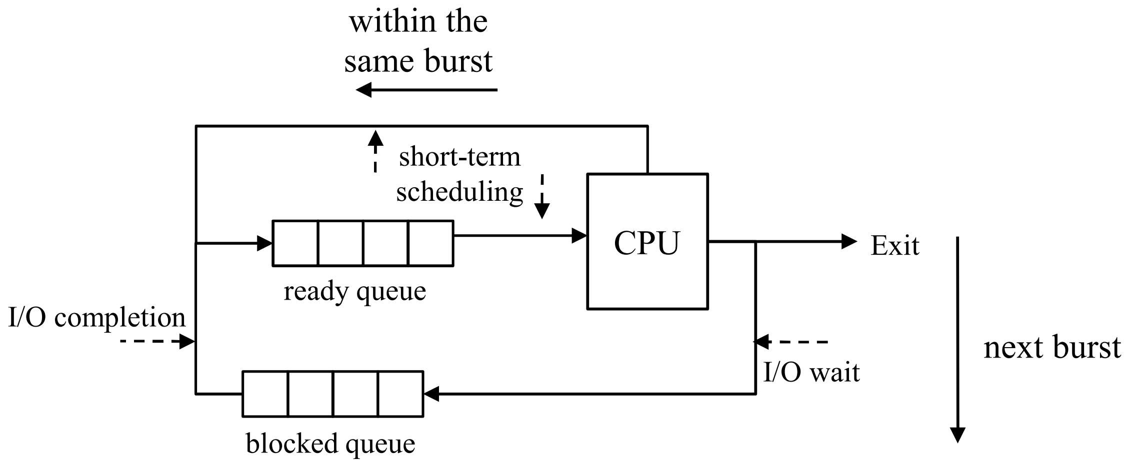
There is a direct and relationship between the durations of the burst t_n to come and the residual capacity C(t) (i.e. any future burst is a fraction of the residual capacity):

$$C(t) = \sum_{\forall n} t_n$$

context parameters

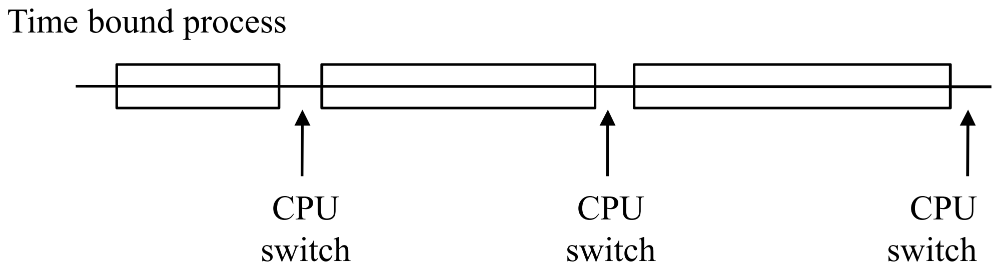


	Burst	
id	**Position**	**Duration t**
t_0	1, 4	4-1=3
t_1	4, 12	(7-6)+(12-10)=3



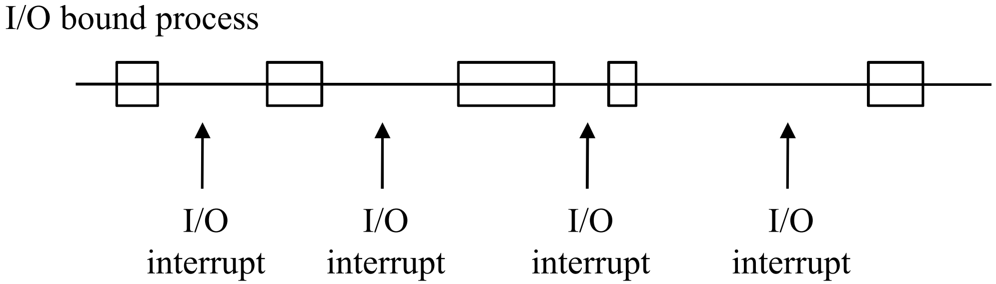
Process and diagram models (4)

Process behavior: some processes spend most of their time computing (time-bound), while others spend most of their time waiting for I/O (I/O bound).

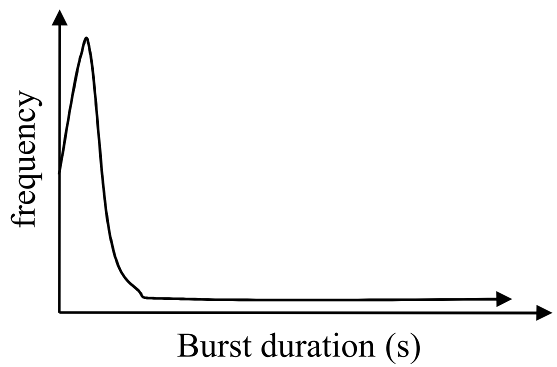


The key factor is the length of the CPU burst, not the length of the I/O burst i.e. The I/O bound processes do not compute much between the I/O requests.

It is worth nothing that as a CPU gets faster, processes tend to be bounded with I/O. As a consequence, resource scheduling become an important issue.



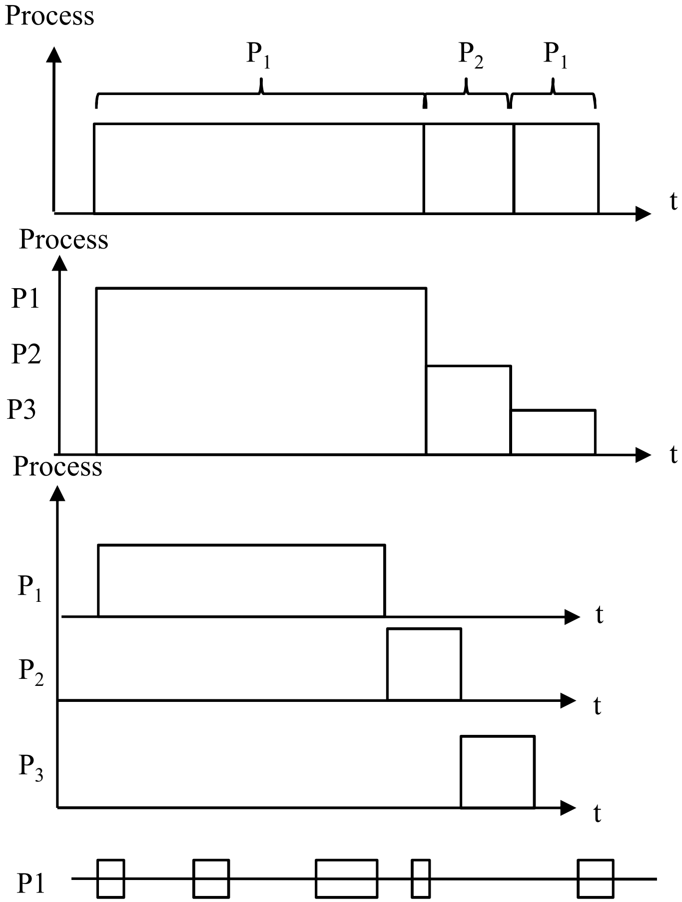
Time measurement is related to the analysis of the duration of CPU bursts. The CPU bursts tend to have a frequency characterized as an exponential. This law varies from process to process and from computer to computer.



Process and diagram models (5)

Scheduling diagrams vary from book to book and from lecture to lecture.

- Process diagram



- Gantt diagram

- Table

id processus

time or quantum		0-1	1-2	2-3	3-4	4-5	5-6	6-7	7-8	8-9
P1	C(t)	7 - 6	6 - 5	5 - 4	4 - 3	3 - 2	2 - 2	2 - 2	2 - 1	1 - 0
P2	C(t)					2 - 2	2 - 1	1 - 0		

variation of C(t) only

value when the quantum starts C(t=2) = 5

value when the quantum stops C(t=3) = 4

x-x waiting process

x-x running process

id processus

time or quantum		0-1	1-2	2-3	3-4	4-5	5-6	6-7	7-8	8-9
P1	C(t)	7-6	6-5	5-4	4-3	3-2	2-2	2-2	2-1	1-0
	α(t)	14-13	13-12	12-11	11-10	10-9	9-9	9-9	9-8	8-7
P2	C(t)					2-2	2-1	1-0		
	α(t)					10-10	10-9	9-8		

variation of C(t) with an other criterion α(t)

x-x waiting process

x-x running process

- The diagram is a text ….

Foundation of operating systems for soft real-time scheduling

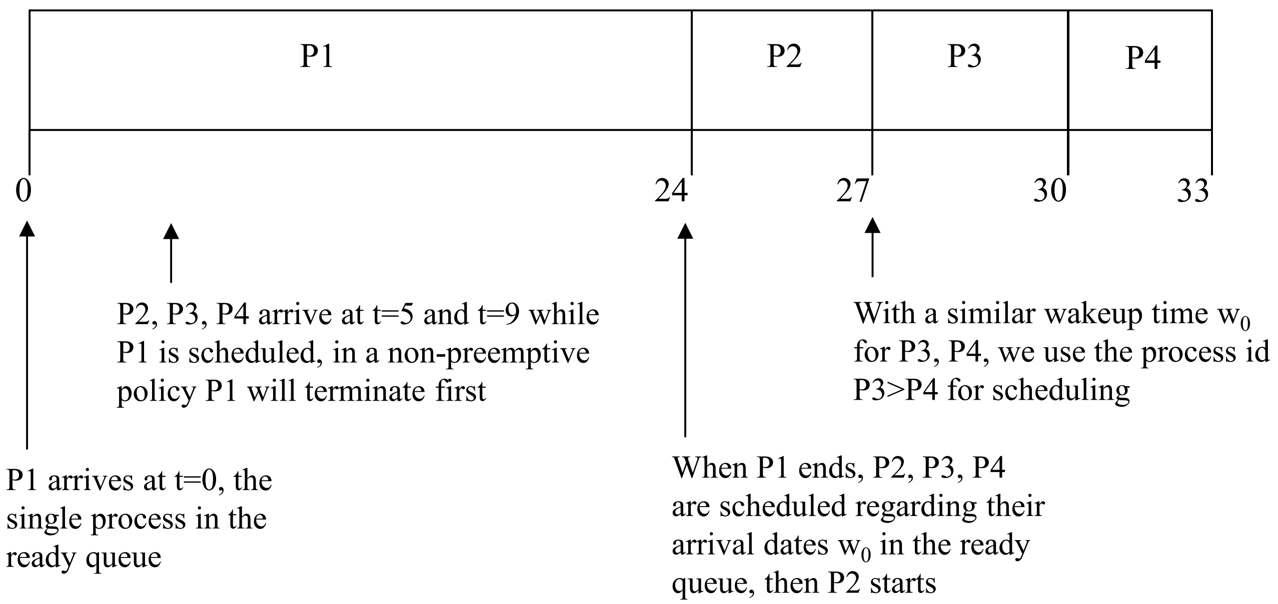
1. Introduction
2. Process description and control
3. Short-term scheduling
   3.1. About short-term scheduling
   3.2. Context switch, quantum and ready queue
   3.3. Process and diagram models
   3.4. Scheduling algorithms
   3.5. Modeling multiprogramming
   3.6. Evaluation of algorithms
4. Soft real-time scheduling

Algorithm	Preemptive	Scheduling criterion	Priority	Predictable capacity	Performance criteria	Taxonomy
First Come First Serve	no	rank in the queue	static	no	Arrival time	Arrival
Priority Scheduling	yes/no	process priority	static	no	Respecting the priority	Priority
Dynamic Priority Scheduling	yes	process priority with aging	dynamic	no	Respecting the priority and avoiding the fairness	
Highest Response Ratio Next	no	response ratio	dynamic	yes	Optimal response time	Optimization
Shortest Job First	yes/no	shortest remaining time	static/ dynamic	yes	Optimal waiting time	
Time prediction	no/yes	shortest predicted time	dynamic	no	Achieving the predictability with the SJF	

Scheduling algorithms
"First Come, First Served (FCFS)"

First Come First Serve (FCS): processes are scheduled regarding their positions in the ready queue (1, 2, 3, …). With equal arrival date (wakeup time) w_0, the process id could be used P1>P2>P3 etc.

Processes	Wakeup (w_0)	Capacity (C)
P1	0	24
P2	5	3
P3	9	3
P4	9	3

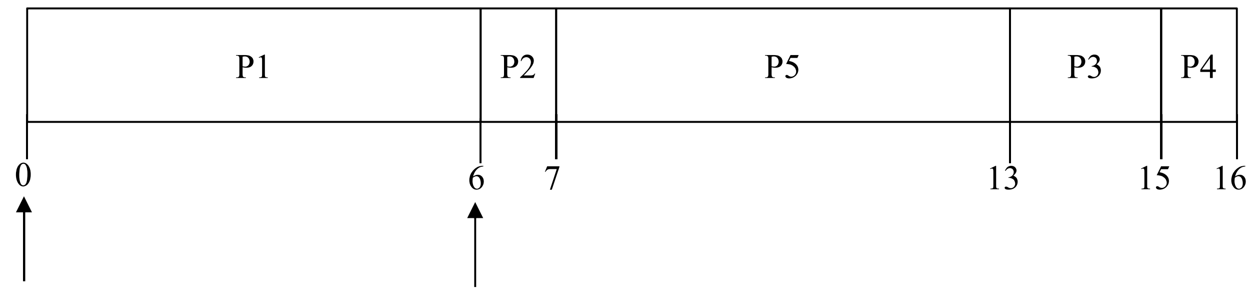


Algorithm	**Preemptive**	**Scheduling criterion**	**Priority**	**Predictable capacity**	**Performance criteria**	**Taxonomy**
First Come First Serve	no	rank in the queue	static	no	Arrival time	Arrival
Priority Scheduling	yes/no	process priority	static	no	Respecting the priority	Priority
Dynamic Priority Scheduling	yes	process priority with aging	dynamic	no	Respecting the priority and avoiding the fairness	
Highest Response Ratio Next	no	response ratio	dynamic	yes	Optimal response time	Optimization
Shortest Job First	yes/no	shortest remaining time	static/ dynamic	yes	Optimal waiting time	
Time prediction	no/yes	shortest predicted time	dynamic	no	Achieving the predictability with the SJF	

Scheduling algorithms
"Priority Scheduling (PS)" (1)

Priority Scheduling (PS): when a process is finished, we shift to the process with the highest priority (i.e. the lowest P value).

Processes	Wakeup (w_0)	Capacity (C)	Priority (P)
P1	0	6	3
P2	1	1	1
P3	2	2	4
P4	3	1	5
P5	4	6	2



P1 alone in the ready queue arrives at t=0

At t=6, P2, P3, P4, P5 are in the ready queue, the scheduling will go on with the priority order P2>P5>P3>P4

Scheduling algorithms
“Priority Scheduling (PS)” (2)

Priority Scheduling (PS): the preemptive case, at any time, we look for the process of the highest priority (i.e. the lowest P value).

Processes	Wakeup (w_0)	Capacity (C)	Priority (P)
P1	0	6	3
P2	1	1	1
P3	2	2	4
P4	3	1	5
P5	4	6	2

t or q		0-1	1-2	2-3	3-4	4-5	5-6	6-7	7-8	8-9	9-10	10-11	11-12	12-13	13-14	14-15	15-16
P1	C(t)	6-5	5-5	5-4	4-3	3-3	3-3	3-3	3-3	3-3	3-3	3-2	2-1	1-0			
P2	C(t)		1-0														
P3	C(t)			2-2	2-2	2-2	2-2	2-2	2-2	2-2	2-2	2-2	2-2	2-2	2-1	1-0	
P4	C(t)				1-1	1-1	1-1	1-1	1-1	1-1	1-1	1-1	1-1	1-1	1-1	1-1	1-0
P5	C(t)					6-5	5-4	4-3	3-2	2-1	1-0						

P2 of highest priority takes the CPU

P5 of highest priority preempts P1

When a process ends, the process with the lowest priority is scheduled

Scheduling algorithms
"Dynamic Priority Scheduling (DPS)"

Dynamic Priority Scheduling (DPS): works with a dynamic priority P(t) and is a preemptive algorithm

1. a process starts with a $P(t=w_0) = P$, its initial priority value
2. when a process is running, P(t) is constant
3. when a process is waiting $P(t+1) = P(t)+1$
4. at any time, the process of highest P(t) takes the CPU
5. if $P_i(t) = P_j(t)$ for two processes i,j, thus we look for $P_i(w_0)$, $P_j(w_0)$
6. when a process recovers the CPU at t_n, we reset $P(t_n) = P(w_0) = P$

Processes	Wakeup (w_0)	Capacity (C)	Priority (P)
P1	0	∞	1
P2	0	∞	3
P3	0	∞	5

t or q		0-1	1-2	2-3	3-4	4-5	5-6	6-7	7-8	8-9	9-10	10-11	11-12	12-13	13-14	14-15	15-16
P1	P(t)	1-2	2-3	3-4	4-5	5-6	1-1	1-2	2-3	3-4	4-5	5-6	6-7	1-1	1-2	2-3	3-4
P2	P(t)	3-4	4-5	5-6	3-3	3-4	4-5	5-6	3-3	3-4	4-5	5-6	3-3	3-4	4-5	5-6	3-3
P3	P(t)	5-5	5-5	5-5	5-6	5-5	5-6	5-5	5-6	5-5	5-5	5-5	5-6	6-7	5-5	5-5	5-6

P3 is running,
P(t) is constant

A context switch
we reset P(t)

Equivalence case,
we look for $P(w_0)$

Equivalence case,
we look for $P(w_0)$

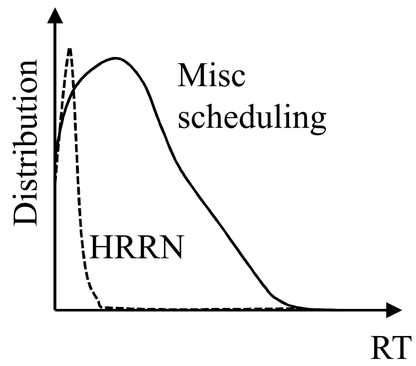
Algorithm	**Preemptive**	**Scheduling criterion**	**Priority**	**Predictable capacity**	**Performance criteria**	**Taxonomy**
First Come First Serve	no	rank in the queue	static	no	Arrival time	Arrival
Priority Scheduling	yes/no	process priority	static	no	Respecting the priority	Priority
Dynamic Priority Scheduling	yes	process priority with aging	dynamic	no	Respecting the priority and avoiding the fairness	
Highest Response Ratio Next	no	response ratio	dynamic	yes	Optimal response time	Optimization
Shortest Job First	yes/no	shortest remaining time	static/ dynamic	yes	Optimal waiting time	
Time prediction	no/yes	shortest predicted time	dynamic	no	Achieving the predictability with the SJF	

Scheduling algorithms
"Highest Response Ratio Next (HRRN)" (1)

For each process, we would like to minimize a normalized turnaround time defined as

$$R_i(t) = \frac{WT_i(t) + C_i}{C_i} = \frac{WT_i(t)}{C_i} + 1$$

with $WT_i(t)$ the waiting time of process i at t and C_i the capacity. Let's note that $1 \leq R_i(t) \leq \propto$



Considering a non-preemptive scheduling we have T(t) = 0 at t<s,
then WT(t) = E(t) – (T(t)=0) = E(t) = t – w_0, R(t) is then

$$R_i(t) = \frac{(t - w_0) + C_i}{C_i} = \frac{(t - w_0)}{C_i} + 1$$

The scheduling is non-preemptive and looks for the highest R(t) value at any context switch.

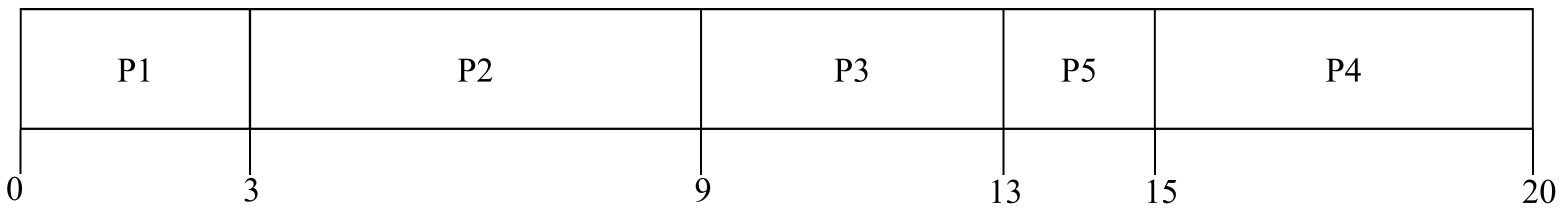
The idea behind this method is to get the mean response ratio low,
so if a job has a high response ratio, it should be run at once to reduce the mean.

Scheduling algorithms
"Highest Response Ratio Next (HRRN)" (2)

For each process, we would like to minimize a normalized turnaround time defined as

$$R_i(t) = \frac{WT_i(t) + C_i}{C_i} = \frac{(t - w_0) + C_i}{C_i}$$

Processes	Wakeup (w_0)	Capacity (C)
P1	0	3
P2	2	6
P3	4	4
P4	6	5
P5	8	2



P1 arrives at t=0, it is the single process in the ready queue and then scheduled

P2 arrives at w_0=2, the single waiting process in the ready queue, it is scheduled next

P3, P4 and P5 are here, we compute the $R_i(t)$, we have $R_3(t) > R_4(t) > R_5(t)$, P3 is scheduled next

$$R_3(t) = \frac{(9-4)+4}{4} = \frac{9}{4} = 2{,}25$$

$$R_4(t) = \frac{(9-6)+5}{5} = \frac{8}{5} = 1{,}6$$

$$R_5(t) = \frac{(9-8)+2}{2} = \frac{3}{2} = 1{,}5$$

P4 and P5 are still waiting, with $R_5(t) > R_4(t)$ P5 is the next process.
We observe a priority inversion between P4, P5 at t=9 and t=13 due to the dynamic R(t)

$$R_4(t) = \frac{(13-6)+5}{5} = \frac{12}{5} = 2{,}4$$

$$R_5(t) = \frac{(13-8)+2}{2} = \frac{7}{2} = 3{,}5$$

P4 the last process is scheduled next

Scheduling algorithms
“Shortest Job First (SJF)”

Shorted Job First (SJF): in the preemptive case, at any time, it looks for the process of the shortest residual capacity C(t) in the ready queue. It is also called Shortest Remaining Time (SRT). The non preemptive version is called the Shortest Process Next (SPN). When a process ends, it looks for the process of the shortest capacity C in the ready queue.

Processes	Wakeup (w_0)	Capacity (C)
P1	0	7
P2	2	4
P3	4	1
P4	5	4

t or q		0-1	1-2	2-3	3-4	4-5	5-6	6-7	7-8	8-9	9-10	10-11	11-12	12-13	13-14	14-15	15-16
P1	C(t)	7-6	6-5	5-5	5-5	5-5	5-5	5-5	5-5	5-5	5-5	5-5	5-4	4-3	3-2	2-1	1-0
P2	C(t)			4-3	3-2	2-2	2-1	1-0									
P3	C(t)					1-0											
P4	C(t)						4-4	4-4	4-3	3-2	2-1	1-0					

A shortest process arises,
we shift the context

When a process ends,
we shift to the process of shortest remaining C(t)

Scheduling algorithms "Time prediction" (1)

One difficulty with the SJF algorithm is the need to know the required residual capacity. When the system cannot guaranty a predictability, we can use the time prediction.

✓For the I/O bound processes, the OS may keep a CPU burst average T_n for each of the processes. This criterion T interpolates a fraction 1/n of the CPU time consumed (and then the residual capacity C(t)).

✓The simplest calculation for T_n would be the following

$$T_{n+1} = \frac{1}{n}\sum_{i=1}^{n} t_i$$

✓To avoid recalculating the entire summation each time, we can rewrite the previous equation as

$$T_{n+1} = \frac{1}{n} t_n + \frac{n-1}{n} T_n$$

✓A common technique for predicting a future value on the basis of a time series is **exponential averaging**

$$T_{n+1} = \alpha \times t_n + (1-\alpha) \times T_n$$

$$T_{n+1} = \alpha \times t_n + \alpha(1-\alpha) \times t_{n-1} + \ldots + \underbrace{\alpha(1-\alpha)^j \times t_{n-j} + \ldots + \alpha(1-\alpha)^n \times T_0}$$

because $\alpha \in$ [0-1], each term has less weight than its predecessor

with,

T_{n+1} is the prediction of the next CPU burst "n+1"
T_n time prediction of the current CPU burst "n"
t_n time value of the current CPU burst "n"
α controls the relative weight (0-1) between the next (T_{n+1}) and the previous (T_n) prediction

Scheduling algorithms
"Time prediction" (2)

A common technique for predicting a future value on the basis of a time series is **exponential averaging**

$$T_{n+1} = \alpha \times t_n + (1-\alpha) \times T_n$$

with,

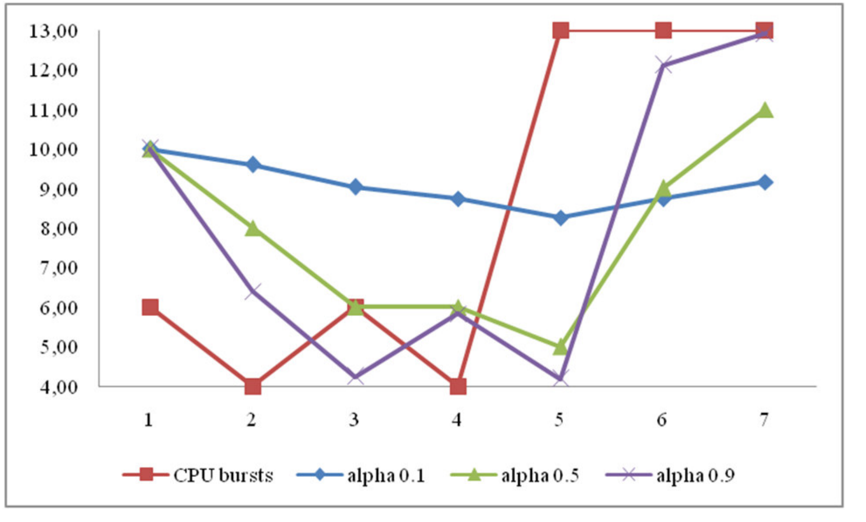
T_{n+1} is the prediction of the next CPU burst "n+1"
T_n time prediction of the current CPU burst "n"
t_n time value of the current CPU burst "n"
α controls the relative weight (0-1) between the next (T_{n+1}) and the previous (T_n) prediction

$\alpha = 0$ $T_{n+1} = T_n$ recent history has no effect
$\alpha = 1$ $T_{n+1} = t_n$ only the most recent CPU burst matters

If first execution (i.e. w_0), T0 is a chosen as a constant (e.g. the overall system average)

t_i	Ti		
	alpha		
	0,1	0,5	0,9
6,00	10,00	10,00	10,00
4,00	9,60	8,00	6,40
6,00	9,04	6,00	4,24
4,00	8,74	6,00	5,82
13,00	8,26	5,00	4,18
13,00	8,74	9,00	12,12
13,00	9,16	11,00	12,91
13,00	9,55	12,00	12,99

$9{,}6 = 0{,}1 \times 6 + 0{,}9 \times 10$ $6{,}4 = 0{,}9 \times 6 + 0{,}1 \times 10$



Scheduling algorithms
"Time prediction" (3)

e.g. Time prediction with the SRT algorithm (SJF preemptive)

i. We consider the case of two processes A, B with the following observed CPU bursts and I/O completion events at a time interval $[t_0, t_0+T]$ At t_0, A, B are in the blocking queue.

ii. We have T0 = 5 and $\alpha = 0.4$ as parameters.

iii. We assume that at any I/O completion event A, B are concurrent for the CPU access (i.e. when B released A is scheduled and vice-versa).

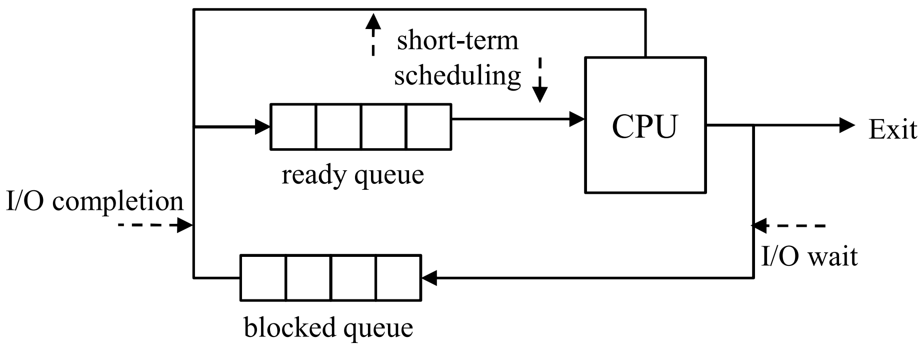
Process A

t_i	Ti
	alpha
	0,4
4,00	5,00
5,00	4,60
3,00	4,76

Process B

t_i	Ti
	alpha
	0,4
3,00	5,00
6,00	4,20
4,00	4,92

I/O completion events	
1	A
2	B
3	A
4	A,B
5	B



Scheduling algorithms
"Time prediction" (4)

e.g. Time prediction with the SRT algorithm (SJF preemptive)

i. We consider the case of two processes A, B with the following observed CPU bursts and I/O completion events at a time interval $[t_0, t_0+T]$ At t_0, A, B are in the blocking queue.

ii. We have T0 = 5 and $\alpha = 0.4$ as parameters.

iii. We assume that at any I/O completion event A, B are concurrent for the CPU access (i.e. when B released A is scheduled and vice-versa).

Process A

t_i	Ti
	alpha
	0,4
4,00	5,00
5,00	4,60
3,00	4,76

Process B

t_i	Ti
	alpha
	0,4
3,00	5,00
6,00	4,20
4,00	4,92

I/O completion events	
1	A
2	B
3	A
4	A,B
5	B

events	<1	1	2		3			4		5		
blocked queue	A,B	B		A		A	A,B		B		A	A,B
ready queue			B(5)		B(5)			A(4.7)		B(4.9)		
CPU		A(5)	A(5)	B(5)	A(4.6)	B(5)		B(4.2)	A(4.7)	A(4.7)	B(4.9)	

- A is released and alone in the ready queue, then scheduled
- B is released while A is scheduled (iii), with $T_B=T_A$ then B waits in the ready queue
- When A ends, B is scheduled
- A is released while B is scheduled (iii), with $T_A \leq T_B$ then B is preempted
- When A ends, B shifts from the ready queue to the CPU then ends and returns to the blocked queue
- Two releases occur at the same time, $T_B \leq T_A$ then A waits in the queue
- B is released while A is scheduled (iii), with $T_B \geq T_A$ B waits in the ready queue

Scheduling algorithms "Time prediction" (5)

e.g. Time prediction with the SRT algorithm (SJF preemptive)

i. We consider the case of two processes A, B with the following observed CPU bursts and I/O completion events at a time interval $[t_0, t_0+T]$ At t_0, A, B are in the blocking queue.
ii. We have T0 = 5 and $\alpha = 0.4$ as parameters.
iii. We assume that at any I/O completion event A, B are concurrent for the CPU access (i.e. when B released A is scheduled and vice-versa).

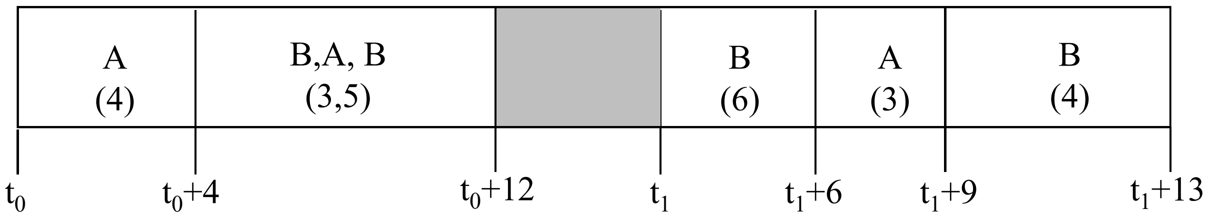
Process A

t_i	Ti
	alpha
	0,4
4,00	5,00
5,00	4,60
3,00	4,76

Process B

t_i	Ti
	alpha
	0,4
3,00	5,00
6,00	4,20
4,00	4,92

I/O completion events	
1	A
2	B
3	A
4	A,B
5	B



Foundation of operating systems for soft real-time scheduling

1. Introduction
2. Process description and control
3. Short-term scheduling
   3.1. About short-term scheduling
   3.2. Context switch, quantum and ready queue
   3.3. Process and diagram models
   3.4. Scheduling algorithms
   3.5. Modeling multiprogramming
   3.6. Evaluation of algorithms
4. Soft real-time scheduling

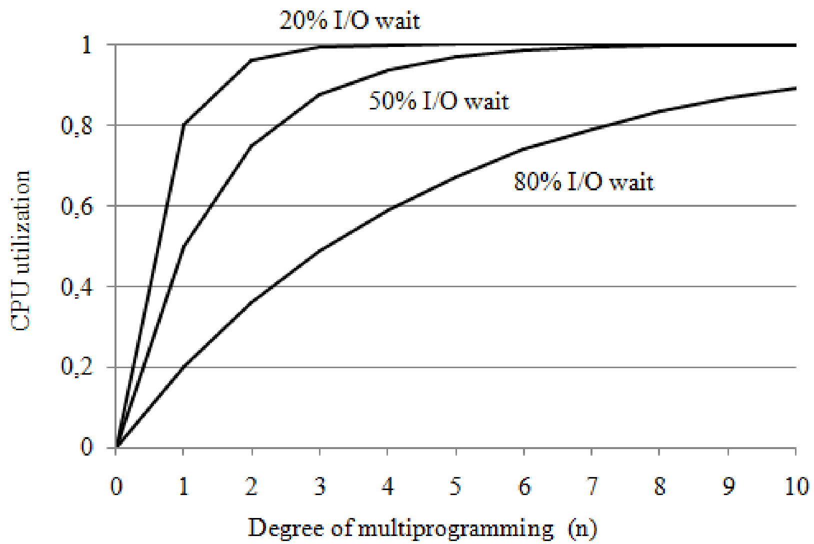
Modeling multiprogramming

Modeling multiprogramming: from a probabilistic point of view, suppose that a process spends a fraction *p* of its time waiting for I/O to complete.

With n processes in memory, the probability that these processes are waiting for I/O (the case where the CPU will be idle) is p^n. The CPU utilization is then given by the formula

$CPU\ utilization = 1 - p^n$ — n is the number of processes; p is their (common) I/O rate

e.g. 80% I/O rate, 4 processes $CPU\ utilization = 1 - 0{,}8^4 = 0{,}5904$



When the I/O rates are different, formula can be expressed as

$CPU\ utilization = 1 - \prod_{i=1}^{n} p_i$ — n is the number of processes; p_i is the I/O rate of process *i*

e.g. P1 (80%), P2(60%), P3(40%) P4(60%) $CPU\ utilization = 1 - (0{,}8 \times 06 \times 0{,}4 \times 0{,}6) = 0{,}8704$

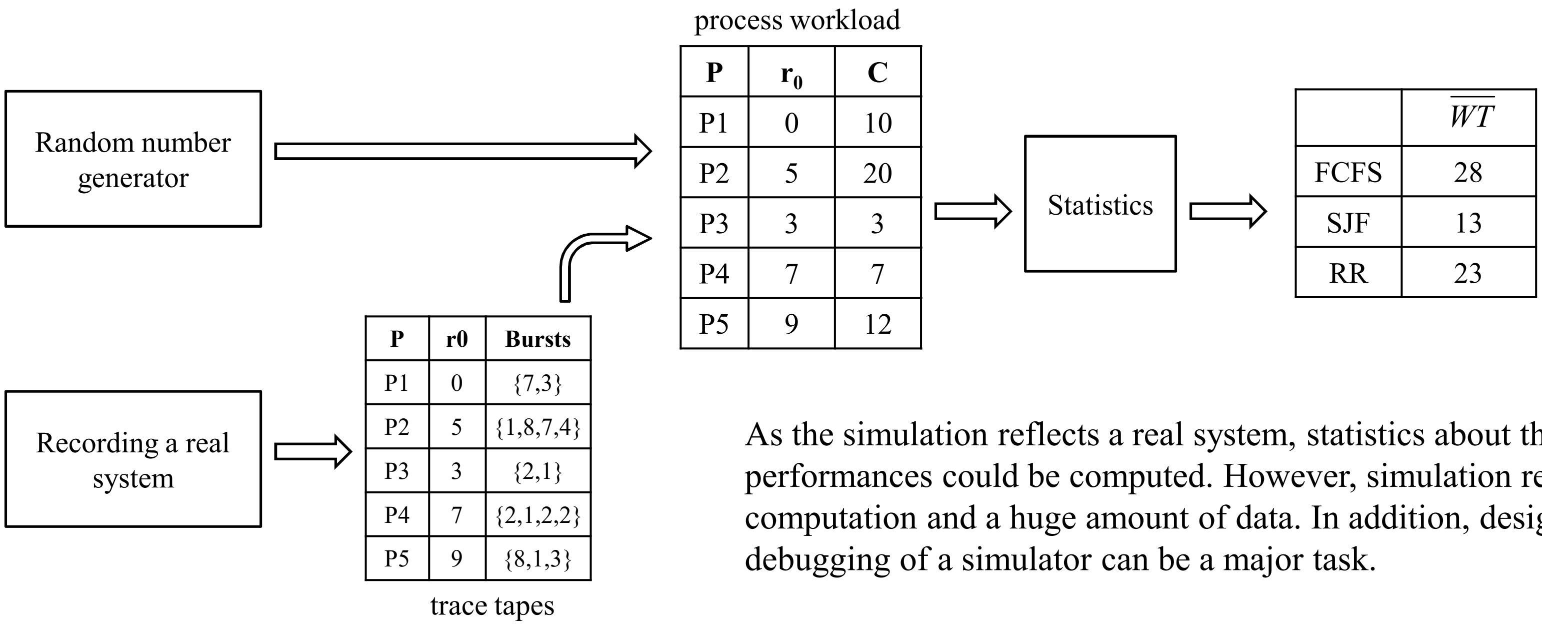
Foundation of operating systems for soft real-time scheduling

1. Introduction
2. Process description and control
3. Short-term scheduling
   3.1. About short-term scheduling
   3.2. Context switch, quantum and ready queue
   3.3. Process and diagram models
   3.4. Scheduling algorithms
   3.5. Modeling multiprogramming
   3.6. Evaluation of algorithms
4. Soft real-time scheduling

Evaluation of algorithms

Simulation aims to handle a model of the OS for evaluation (scheduling algorithm, processes, etc.). The simulator has a variable representing a clock, when increasing the simulator modifies the state of the system.
The data to drive simulation can be generated in two main ways:

- to use synthetic data with a random number generator.
- to record trace tapes by monitoring a real system.



process workload

P	r_0	C
P1	0	10
P2	5	20
P3	3	3
P4	7	7
P5	9	12

P	r0	Bursts
P1	0	{7,3}
P2	5	{1,8,7,4}
P3	3	{2,1}
P4	7	{2,1,2,2}
P5	9	{8,1,3}

trace tapes

	$\overline{WT}$
FCFS	28
SJF	13
RR	23

As the simulation reflects a real system, statistics about the algorithm performances could be computed. However, simulation requires hours of computation and a huge amount of data. In addition, design, coding and debugging of a simulator can be a major task.

Foundation of operating systems for soft real-time scheduling

1. Introduction
2. Process description and control
3. Short-term scheduling
   - 3.1. About short-term scheduling
   - 3.2. Context switch, quantum and ready queue
   - 3.3. Process and diagram models
   - 3.4. Scheduling algorithms
   - 3.5. Modeling multiprogramming
   - 3.6. Evaluation of algorithms
4. Soft real-time scheduling

Soft real-time scheduling (1/2)

Algorithm	Pros	Cons
First Come First Serve	The FCFS can be applied for the batch processing i.e. the real-time task is still the most recent submitted task. As the scheduling policy is no preemptive, the critical section problem doesn't matter.	The wakeup times need to be ordered. The scheduling policy is not preemptive, while running an equal priority matters.
Priority Scheduling	The real-time tasks having high priorities are mixed with time-sharing tasks having low priorities.	The number of real-time tasks must be low to avoid starvation and missed deadlines. The scheduling policy is preemptive, a critical section could be handled but with a priority inversion.
Dynamic Priority Scheduling	The DPS will smooth the starvation for the time-sharing tasks, in the case of a huge CPU consumption of the real-time tasks.	Similar to PS, however, it relaxes the response times of real-time tasks and constraints.
Highest Response Ratio Next	The HRRN is able to respect real-time constraints for a set of tasks having in-balance deadline and capacity. As the scheduling policy is no preemptive, the critical section problem doesn't matter.	The deadlines must be aligned the capacities with the $R_{max}(t)$ value for all the real-time tasks. The algorithm requires a constant execution time for all the tasks.
Shortest Job First Time prediction	The SJF can be applied if the real-time constraint targets the tasks having a low-level capacity. While running, the priority of a task increases. This smooths the priority inversion problem for mutual exclusion. Predictability ensures estimation of the capacity.	As the priority increases while running, a task with a large capacity will require a largest laxity.

Soft real-time scheduling (2/2)

<table>
<tr><th>Algorithm</th><th>w_o</th><th>RT tasks</th><th>Capacity</th><th>Predictability</th></tr>
<tr><td>First Come First Serve</td><td>Known</td><td>A large number</td><td>Variable</td><td>No</td></tr>
<tr><td>Priority Scheduling</td><td rowspan="2">Un-known</td><td rowspan="2">A few</td><td>Small</td><td rowspan="2">No</td></tr>
<tr><td>Dynamic Priority Scheduling</td><td>Large</td></tr>
<tr><td>Highest Response Ratio Next</td><td rowspan="3">Un-known</td><td rowspan="3">A large number</td><td>Variable</td><td rowspan="2">Yes</td></tr>
<tr><td>Shortest Job First</td><td rowspan="2">Short</td></tr>
<tr><td>Time prediction</td><td>No</td></tr>
</table>