Operating Systems
"Uniprocessor scheduling"

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

Lecture available at http://mathieu.delalandre.free.fr/teachings/operating1.html

Operating Systems
“Uniprocessor scheduling”

1. About short-term scheduling
2. Context switch, quantum and ready queue
3. Process and diagram models
4. Scheduling algorithms
    4.1. FCFS scheduling
    4.2. Priority based scheduling
    4.3. Optimal scheduling
    4.4. Time-sharing based scheduling
    4.5. Priority/Time-sharing based scheduling
5. Modeling multiprogramming
6. Evaluation of algorithms

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.

Operating Systems
"Uniprocessor scheduling"

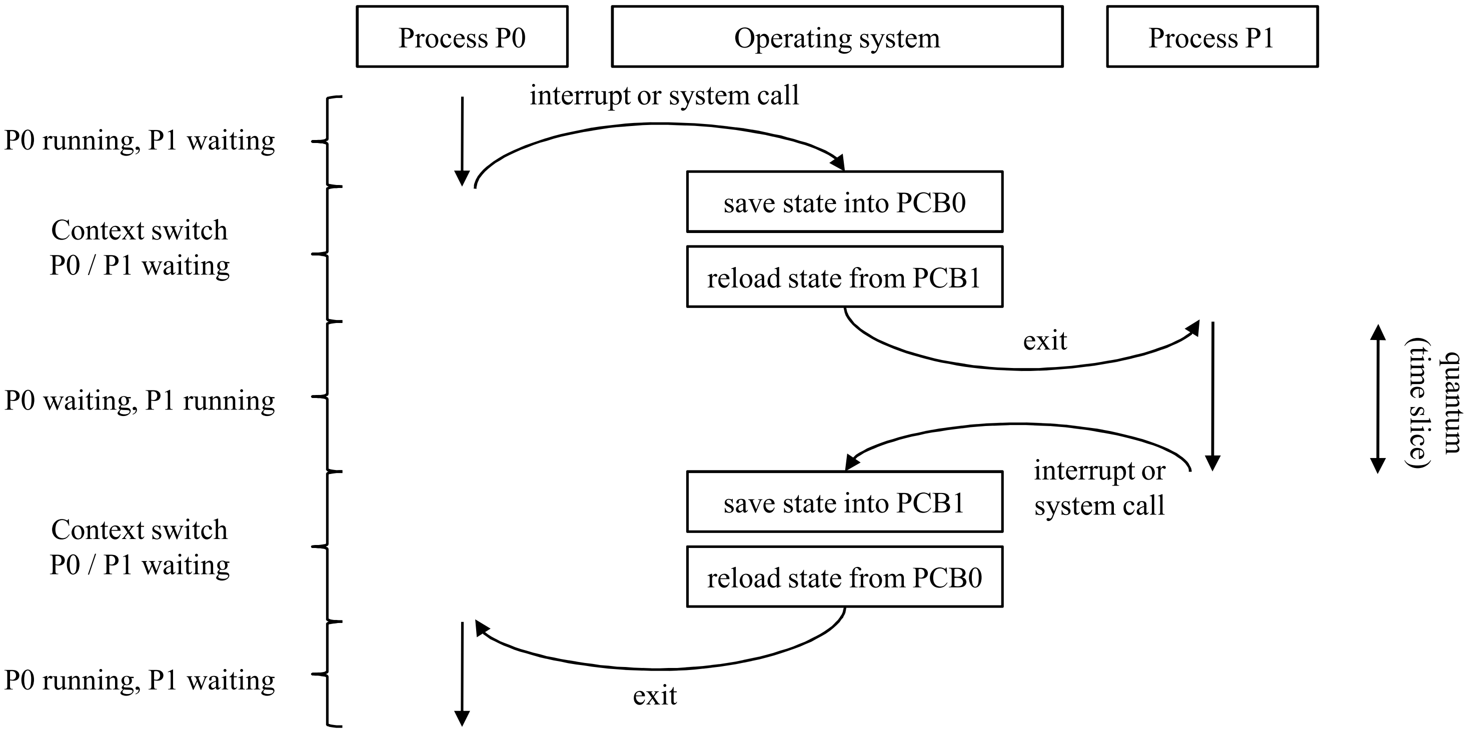
1. About short-term scheduling
2. Context switch, quantum and ready queue
3. Process and diagram models
4. Scheduling algorithms
    4.1. FCFS scheduling
    4.2. Priority based scheduling
    4.3. Optimal scheduling
    4.4. Time-sharing based scheduling
    4.5. Priority/Time-sharing based scheduling
5. Modeling multiprogramming
6. Evaluation of algorithms

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	Process A
4	5003	
5	5004	
6	5005	← A interrupted
7	100	
8	101	
9	102	
10	103	Dispatcher
11	104	
12	105	← B starts
13	8000	
14	8001	Process B
15	8002	
16	8003	← B ends
17	100	
18	101	
19	102	Dispatcher
20	103	
21	104	
22	105	← C starts
23	12000	
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	Dispatcher
30	101	
31	102	
32	103	
33	104	
34	105	
35	5006	← A continues; Process A
36	5007	
37	5008	
38	5009	
39	5010	
40	5011	
41	100	← A ends; Dispatcher
42	101	
43	102	
44	103	
45	104	
46	105	
47	12006	← C continues; Process C
48	12007	
49	12008	
50	12009	
51	12010	
52	1211	← 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	

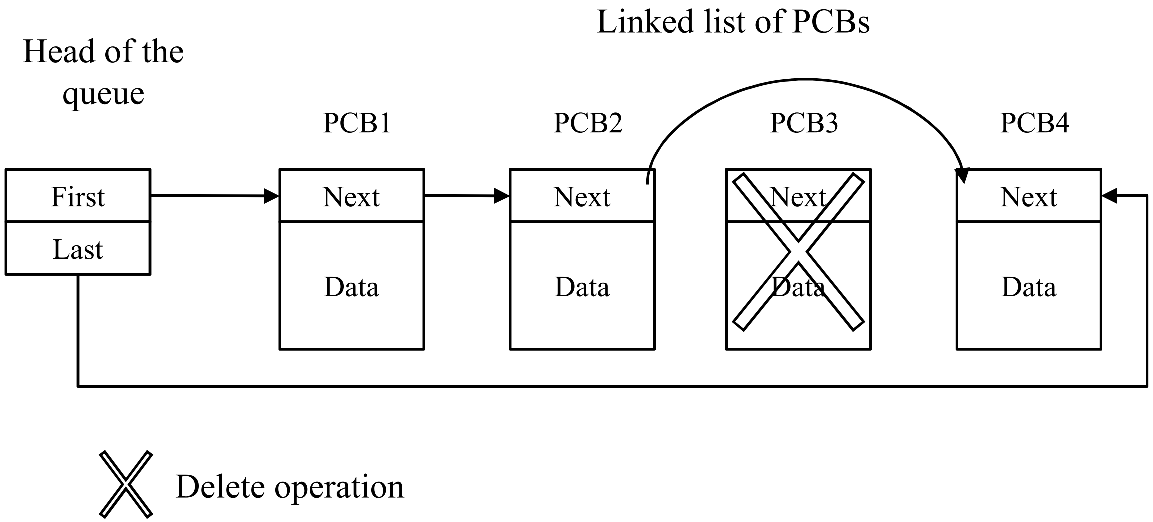
5 quanta / 4 context switches (n-1 quanta)
28 process instruction (6+4+6+6+6)
6×4=24 dispatcher instructions
a maximum of two processes in the ready queue

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.

Operating Systems
"Uniprocessor scheduling"

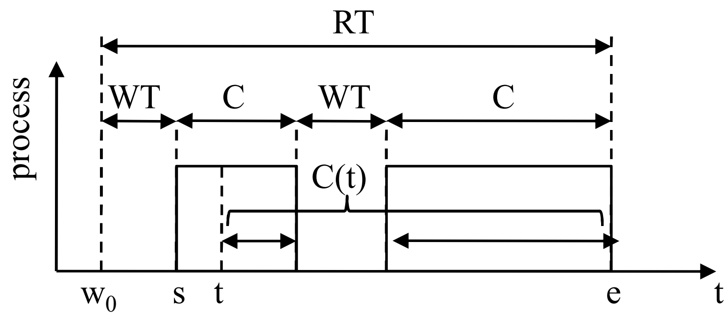
1. About short-term scheduling
2. Context switch, quantum and ready queue
3. Process and diagram models
4. Scheduling algorithms
    4.1. FCFS scheduling
    4.2. Priority based scheduling
    4.3. Optimal scheduling
    4.4. Time-sharing based scheduling
    4.5. Priority/Time-sharing based scheduling
5. Modeling multiprogramming
6. Evaluation of algorithms

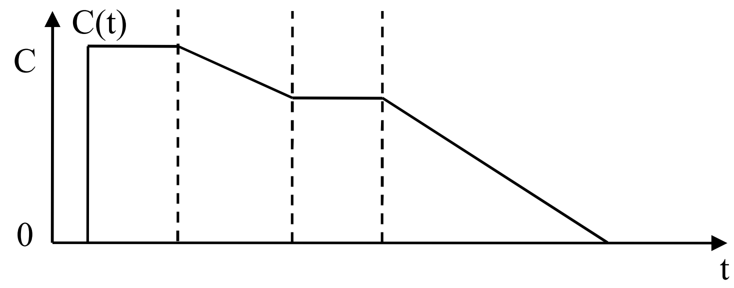
Process and diagram models (1)

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)

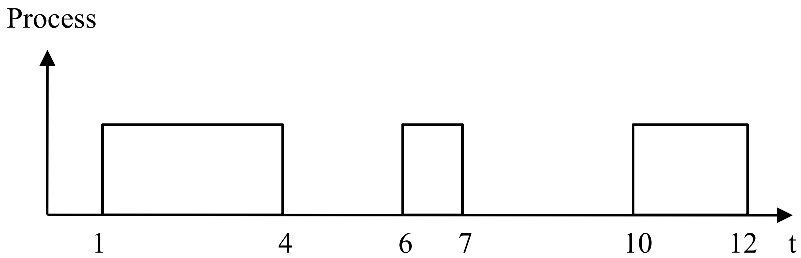
Process model and context parameters

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

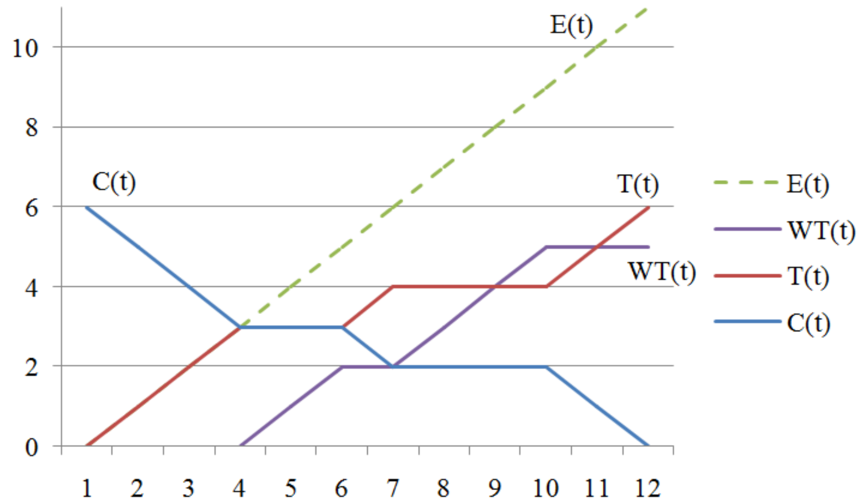
Process parameters

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 $WT(w_0)=0, WT(e)=WT$

context parameters



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)

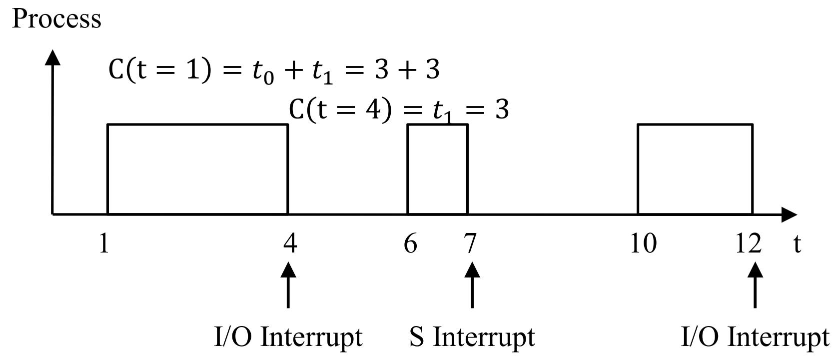
Process model and context parameters

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.

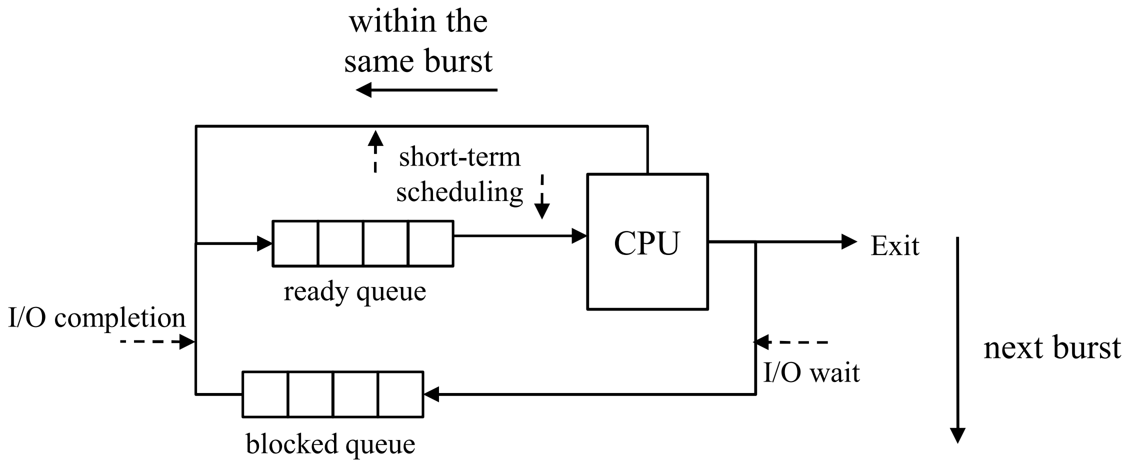
There is a direct and relationship between the durations of the burst t_n to come and the residual capacity C(t) (i.e. any future burst is a fraction of the residual capacity):

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

context parameters



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

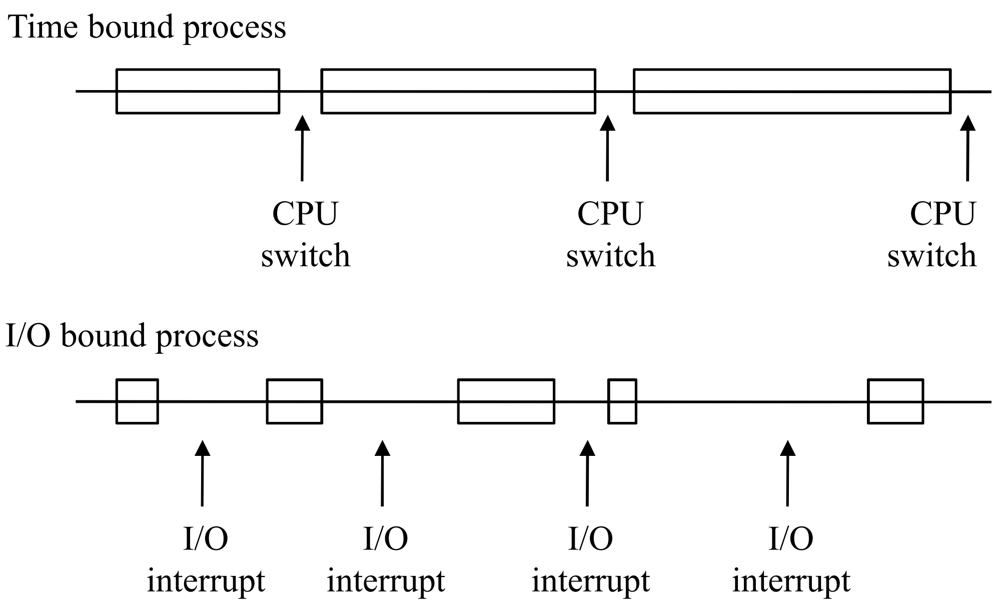


Process and diagram models (4)

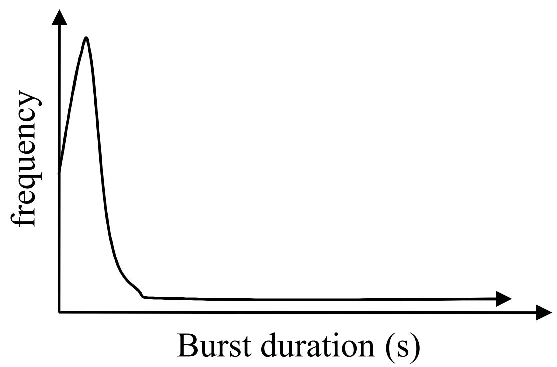
Process behavior: some processes spend most of their time computing (time-bound), while others spend most of their time waiting for I/O (I/O bound).

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.



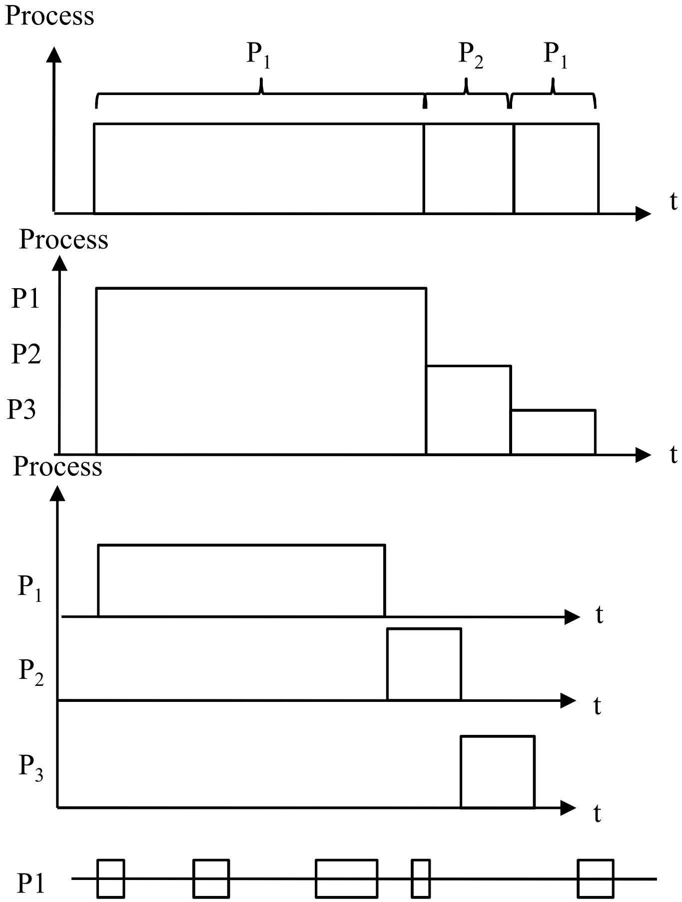
Time measurement is related to the analysis of the duration of CPU bursts. The CPU bursts tend to have a frequency characterized as an exponential. This law varies from process to process and from computer to computer.



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 ….

Operating Systems
"Uniprocessor scheduling"

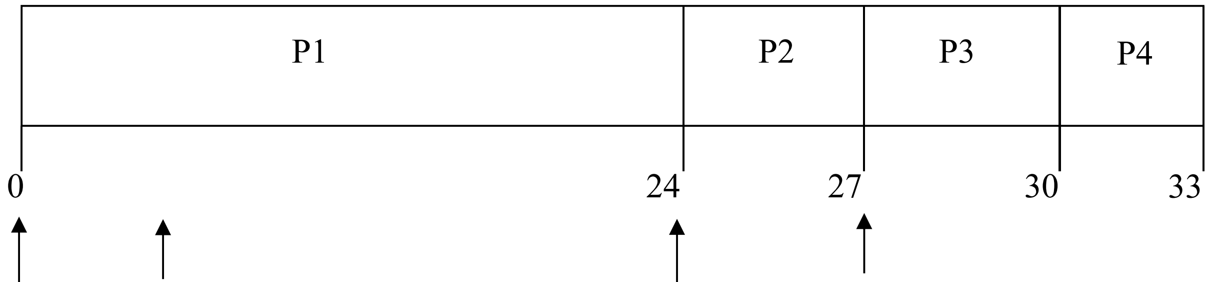
1. About short-term scheduling
2. Context switch, quantum and ready queue
3. Process and diagram models
4. Scheduling algorithms
    - 4.1. FCFS scheduling
    - 4.2. Priority based scheduling
    - 4.3. Optimal scheduling
    - 4.4. Time-sharing based scheduling
    - 4.5. Priority/Time-sharing based scheduling
5. Modeling multiprogramming
6. Evaluation of algorithms

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	
Guaranteed Scheduling	yes	CPU use ratio	dynamic	no	Enforcing the response time	Time sharing
Round-Robin	yes	rank in the queue and round	dynamic	no		
Fair-Share Scheduling	yes	process priority	dynamic	no	Respecting the priority and enforcing the response time	Priority & time sharing
Multilevel feedback queue scheduling	yes	process priority and queue position	static/ dynamic	no		

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



P2, P3, P4 arrive at t=5 and t=9 while P1 is scheduled, in a non-preemptive policy P1 will terminate first

With a similar wakeup time w_0 for P3, P4, we use the process id P3>P4 for scheduling

P1 arrives at t=0, the single process in the ready queue

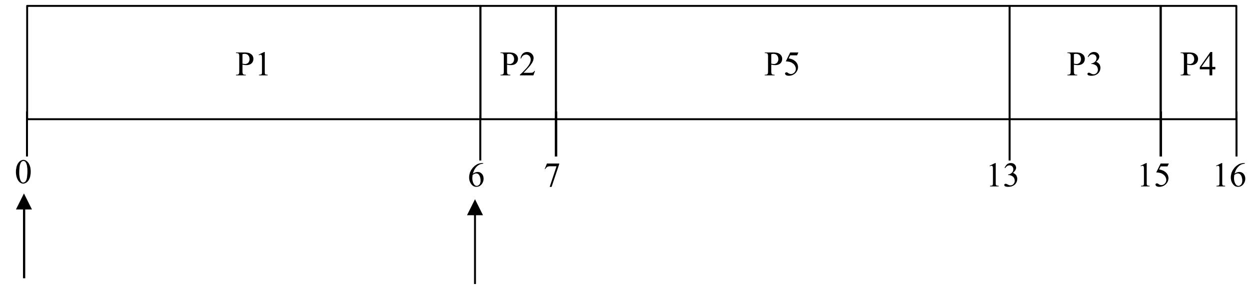
When P1 ends, P2, P3, P4 are scheduled regarding their arrival dates w_0 in the ready queue, then P2 starts

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	
Guaranteed Scheduling	yes	CPU use ratio	dynamic	no	Enforcing the response time	Time sharing
Round-Robin	yes	rank in the queue and round	dynamic	no		
Fair-Share Scheduling	yes	process priority	dynamic	no	Respecting the priority and enforcing the response time	Priority & time sharing
Multilevel feedback queue scheduling	yes	process priority and queue position	static/ dynamic	no		

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 $P(w_0)$

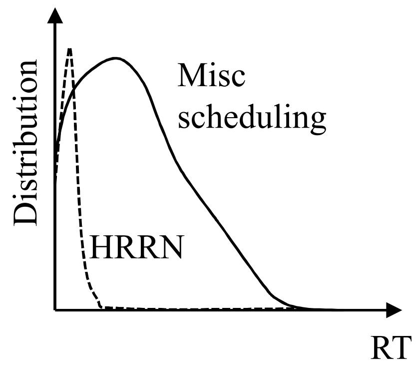
Algorithm	**Preemptive**	**Scheduling criterion**	**Priority**	**Predictable capacity**	**Performance criteria**	**Taxonomy**
First Come First Serve	no	rank in the queue	static	no	Arrival time	Arrival
Priority Scheduling	yes/no	process priority	static	no	Respecting the priority	Priority
Dynamic Priority 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	
Guaranteed Scheduling	yes	CPU use ratio	dynamic	no	Enforcing the response time	Time sharing
Round-Robin	yes	rank in the queue and round	dynamic	no		
Fair-Share Scheduling	yes	process priority	dynamic	no	Respecting the priority and enforcing the response time	Priority & time sharing
Multilevel feedback queue scheduling	yes	process priority and queue position	static/ dynamic	no		

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.

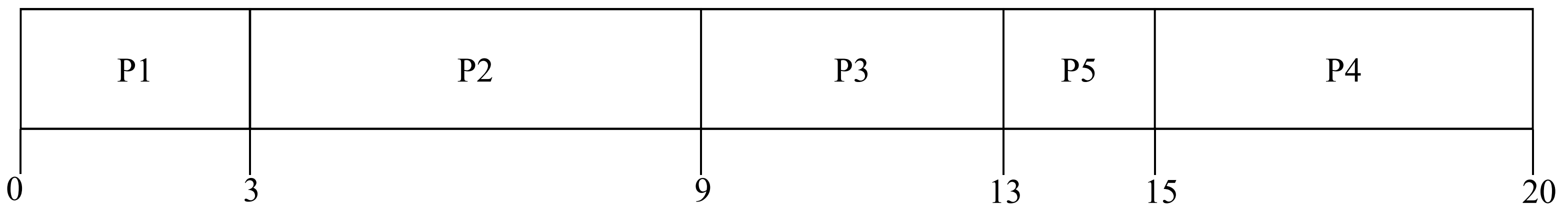
The idea behind this method is to get the mean response ratio low,
so if a job has a high response ratio, it should be run at once to reduce the mean.

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 $\alpha \in [0\text{-}1]$, 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,

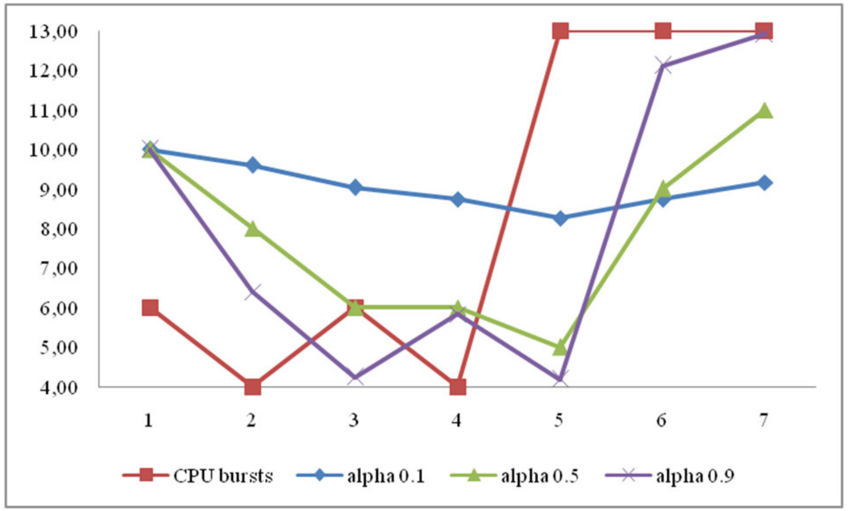
T_{n+1} is the prediction of the next CPU burst "n+1"
T_n time prediction of the current CPU burst "n"
t_n time value of the current CPU burst "n"
α controls the relative weight (0-1) between the next (T_{n+1}) and the previous (T_n) prediction

$\alpha = 0$ $T_{n+1} = T_n$ 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 α = 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).

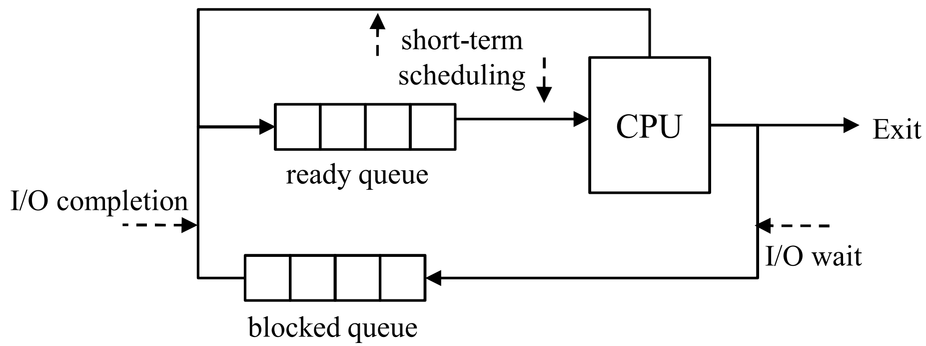
Process A

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).

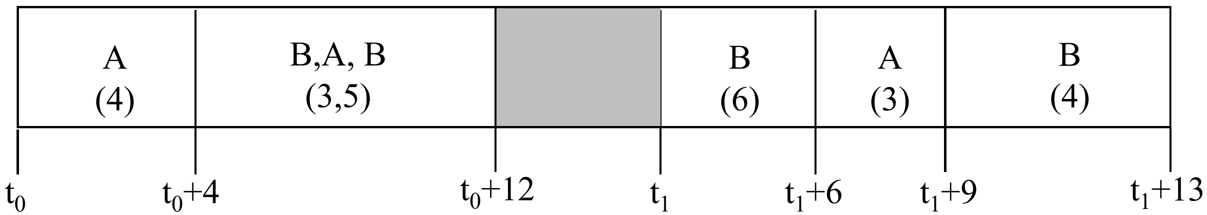
Process A

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



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	
Guaranteed Scheduling	yes	CPU use ratio	dynamic	no	Enforcing the response time	Time sharing
Round-Robin	yes	rank in the queue and round	dynamic	no		
Fair-Share Scheduling	yes	process priority	dynamic	no	Respecting the priority and enforcing the response time	Priority & time sharing
Multilevel feedback queue scheduling	yes	process priority and queue position	static/ dynamic	no		

Scheduling algorithms
"Guaranteed Scheduling (GS)" (1)

With *n* processes running, all things being equal, each one should get 1/n of the CPU utilization. For a process i, the scheduling algorithm:

1. keeps a track of the actual **CPU time consumed**, $F_1(t) = T_i(t) = C_i - C_i(t)$

2. it then computes the **CPU time entitled ratio**, $F_2(t) = \frac{E_i(t)}{n} = \frac{t - w_i}{n}$

3. the **CPU time consumed** is normalized with the **CPU time entitled ratio**, the lowest value has the higher priority.

$$R_i(t) = \frac{F_1(t)}{F_2(t)} = \frac{T_i(t)}{\mathrm{E_i}(t)} \times n = \frac{T_i(t)}{t - w_i} \times n$$

$$R_i(t) \begin{cases} > 1 & T_i(t) > E_i(t)/n \\ = 1 & T_i(t) = E_i(t)/n \\ < 1 & T_i(t) < E_i(t)/n \end{cases}$$

- > 1: P_i got more CPU time than guaranteed.
- $= 1$: P_i got a right faction of the CPU.
- < 1: P_i is in a starvation and has a high priority.

Scheduling algorithms
"Guaranteed Scheduling (GS)" (2)

With *n* processes running, all things being equal, each one should get 1/n of the CPU utilization.

the **CPU time consumed** is normalized with the **CPU time entitled ratio**, the lowest value has the higher priority

$$R_i(t) = \frac{T_i(t)}{t - w_i} \times n$$

Processes	Wakeup (w_0)	Capacity (C)
P1	0	∞
P2	2	∞
P3	4	∞

t or q		0-1	1-2	2-3	3-4	4-5	5-6	6-7	7-8
n		1	1	2	2	3	3	3	3
P1	T(t)	0-1	1-2	2	2-3	3	3	3-4	4
	t-w_0	0-1	1-2	2-3	3-4	4-5	5-6	6-7	7-8
	R(t)	0=(0/0×1)	1=(1/1×1)	2=(2/2×2)	1.3=(2/3×2)	2.2=(3/4×3)	1.8=(3/5×3)	1.5=(3/6×3)	1.7=(4/7×3)
P2	T(t)			0-1	1	1	1-2	2	2
	t-w_0			0-1	1-2	2-3	3-4	4-5	5-6
	R(t)			0=(0/0×2)	2=(1/1×2)	1.5=(1/2×3)	1=(1/3×3)	1.5=(2/4×3)	1.2=(2/5×3)
P3	T(t)					0-1	1	1	1-2
	t-w_0					0-1	1-2	2-3	3-4
	R(t)					0=(0/0×3)	3=(1/1×3)	1.5=(1/2×3)	1=(1/3×3)

When n increases, R(t) increases, we shift to the lowest R(t)

$R_1(t)=R_2(t)=R_3(t)$, we apply a selection on id P1>P2>P3

While a process is scheduled, R(t) increases as T(t) increases
While a process is waiting, R(t) decreases as T(t) is constant

After a while, the algorithm looks for a convergence $R_1(t) \approx R_2(t) \approx R_3(t) \approx 1$

Scheduling algorithms
"Round Robin (RR)"

We assign a quantum set *m* to each process in equal portions and in a circular order (A look-like FCFS), handling all the processes without priority.

i. Every *m* quantum, we shift to the following process in the ready queue.

ii. When a process is ended and a rest of quantum appears, we shift to the next process.

Processes	Wakeup (w_0)	Capacity (C)
P1	0	53
P2	0	17
P3	0	68
P4	0	24

We will use here *m* = 20

t or q		0-20	20-37	37-57	57-77	77-97	97-117	117-121	121-134	134-154	154-162
P1	C(t)	53-33	33-33	33-33	33-33	33-13	13-13	13-13	13-0		
P2	C(t)	17-17	17-0								
P3	C(t)	68-68	68-68	68-48	48-48	48-48	48-28	28-28	28-28	28-8	8-0
P4	C(t)	24-24	24-24	24-24	24-4	4-4	4-4	4-0			

Processes are scheduled regarding their positions in the ready queue

A process is ended before m we shift to the next process

The last process is terminated in some successive steps

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	
Guaranteed Scheduling	yes	CPU use ratio	dynamic	no	Enforcing the response time	Time sharing
Round-Robin	yes	rank in the queue and round	dynamic	no		
Fair-Share Scheduling	yes	process priority	dynamic	no	Respecting the priority and enforcing the response time	Priority & time sharing
Multilevel feedback queue scheduling	yes	process priority and queue position	static/ dynamic	no		

Scheduling algorithms
"Fair-Share Scheduling (FSS)" (1)

Applications may be organized with multiple processes. The FSS scheduling algorithm allocates a fraction of the processor resources to each group. An hybrid scheduling, mixing the round robin & priority scheduling (using a base priority, a exponential iterative reduction rule, a group weighting), assures a fair share of the CPU for each process.

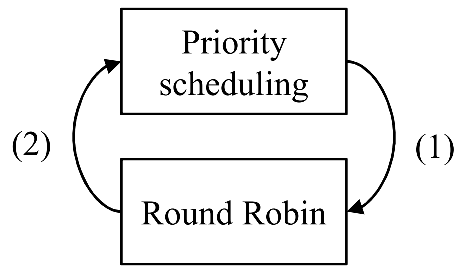
$P_j(i)$	is the priority of process j at beginning of interval i, lower values equal higher priorities
$Base_j$	is the base "or root" priority of process j
$CPU_j(i)$	is the measure of processor utilization by process j through the interval i
$GCPU_k(i)$	is the measure of processor utilization by group k through the interval i
w_k	is the weight assigned to group k, with the constraint $0 \leq w_k \leq 1$ and $\sum_k w_k = 1$

The scheduler applies a round robin and looks for minimization of the criterion $P_j(i)$ at each round.

$$P_j(i) = Base_j + \frac{CPU_j(i)}{2} + \frac{GCPU_k(i)}{4 \times w_k}$$

$$with \quad CPU_j(i) = \frac{CPU_j(i-1)}{2}$$

$$and \quad GCPU_k(i) = \frac{GCPU_k(i-1)}{2}$$



If the $P_j(i)$ are equal $\forall j$,
we apply a selection based on the round robin e.g. P1 > P2 > P3.

(1) If for $\forall j$, two or more min $P_j(i)$ appear
(2) Whenever for $\forall j$, a standalone min $P_j(i)$ is here

Scheduling algorithms
"Fair-Share Scheduling (FSS)" (2)

The scheduler applies a round robin and looks for minimization of the criterion $P_j(i)$ at each round.

$$P_j(i) = Base_j + \frac{CPU_j(i)}{2} + \frac{GCPU_k(i)}{4 \times w_k}$$

$$with \quad CPU_j(i) = \frac{CPU_j(i-1)}{2}$$

$$and \quad GCPU_k(i) = \frac{GCPU_k(i-1)}{2}$$

Processes	Wakeup (w_0)	Priority	Capacity (C)	Group
P1	0	60	∞	1
P2	0	60	∞	2
P3	0	60	∞	2

e.g. $w_1 = w_2 = 0.5$ and $m = 60$

t or q		00-60	60-120	120-180	180-240	240-300	300-360
P1	CPU(t)	0-60	30	15-75	37	18-78	39
	GCPU(t)	0-60	30	15-75	37	18-78	39
	P(t)	60 (60+0+0)	90 (60+15+15)	74 (60+7+7)	96(60+18+18)	78 (60+9+9)	98(60+19+19)
P2	CPU(t)	0	0-60	30	15	7	3-63
	GCPU(t)	0	0-60	30	15-75	37	18-78
	P(t)	60 (60+0+0)	60 (60+0+0)	90 (60+15+15)	74 (60+7+7)	81(60+3+18)	70(60+1+9)
P3	CPU(t)	0	0	0	0-60	30	15
	GCPU(t)	0	0-60	30	15-75	37	18-78
	P(t)	60 (60+0+0)	60 (60+0+0)	75 (60+0+15)	67 (60+0+7)	93(60+15+18)	76(60+7+9)

$P_1(t) = P_2(t) = P_3(t)$, we apply a selection based on the round robin P1 > P2 > P3

$P_2(t) \neq P_3(t)$ with a same $GCPU_2(t)$ and $CPU_2(t) \neq CPU_3(t)$

When P1 is scheduled, $P_1(t)$ increases and $P_2(t) = P_3(t)$ remains constant, the RR policy applies here

Scheduling will go on, P1 will have more chance to get the CPU as it constitutes a single group

Scheduling algorithms
"Multilevel feedback queue scheduling (MLFQ)" (1)

The multilevel feedback queue scheduling algorithm allows processes to move between queues. The idea is to separate the processes according to their CPU bursts.

1. The queues are organized according to priority levels, $\bigcup_{i=0}^{n} RQ_i$

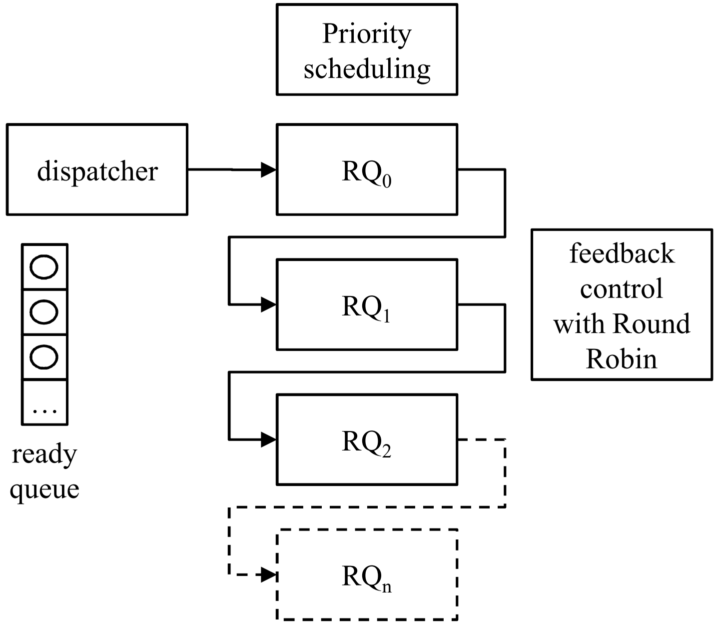
2. When a process first enters in the system, it is placed in RQ_0.

3. In general, a process scheduled from RQ_i is allowed to execute a maximum of $m = 2^{k+i}$ time units (i.e. quantum) before a preemption.

4. After a preemption at level i, a process shifts to the level $i+1$.

5. Within each queue, a simple FCFS mechanism is used.

6. A process at a priority level i can preempt any process at a priority level $> i$.



Scheduling algorithms
“Multilevel feedback queue scheduling (MLFQ)” (2)

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

	m with k=0
RQ_0	$2^{0+0}=1$
RQ_1	$2^{0+1}=2$
RQ_2	$2^{0+2}=4$

t or q		0-1	1-2	2-3	3-4	4-5	5-6	6-7	7-8	8-9	9-10
RQ_0		P1		P2		P3		P4		P5	
RQ_1			P1	P1	P1, P2	P2	P2, P3	P2, P3	P2, P3, P4	P3, P4	P3, P4, P5
RQ_2										P2	P2
P1	C(t)	3-2	2-1	1	1-0						
	P(t)	0-1	1	1	1						
P2	C(t)			6-5	5	5	5-4	4	4-3	3	3
	P(t)			0-1	1	1	1	1	1-2	2	2
P3	C(t)					4-3	3	3	3	3	3-2
	P(t)					0-1	1	1	1	1	1
P4	C(t)							5-4	4	4	4
	P(t)							0-1	1	1	1
P5	C(t)									2-1	1
	P(t)									0-1	1

P1 starts in the queue RQ_0

After a quantum $2^0=1$, P1 shifts to RQ_1

P2 starts and preempts P1 as i=0 < i=1

When P2 shifts to RQ_1, within the FCFS policy P1 is scheduled first

P1 ends before to shift to RQ_2

Same case with P2, P3

Same case with P2, P3 and P4

P5 arrives as P2 shifts to RQ_2 after $2^0+2^1=3$ times units

Scheduling algorithms
“Multilevel feedback queue scheduling (MLFQ)” (3)

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

	m with *k*=1
RQ_0	$2^{1\times0}=1$
RQ_1	$2^{1\times1}=2$
RQ_2	$2^{1\times2}=4$

t or q		10-11	11-12	12-13	13-14	14-15	15-16	16-17	17-18	18-19	19-20
RQ_0											
RQ_1		P3, P4, P5	P4, P5	P4, P5	P5						
RQ_2		P2,	P2, P3	P2, P3	P2, P3, P4	P2, P3, P4	P2, P3, P4	P2, P3, P4	P3, P4	P4	P4
P1	C(t)										
	P(t)										
P2	C(t)	3	3	3	3	3-2	2-1	1-0			
	P(t)	2	2	2	2	2	2	2			
P3	C(t)	2-1	1	1	1	1	1	1	1-0		
	P(t)	1-2	2	2	2	2	2	2	2		
P4	C(t)	4	4-3	3-2	2	2	2	2	2	2-1	1-0
	P(t)	1	1	1-2	2	2	2	2	2	2	2
P5	C(t)	1	1	1	1-0						
	P(t)	1	1	1	1						

After a quantum $2^1=2$, P3 shifts to RQ_2

Same case with P4

P5 ends before to shift to RQ_2

Within RQ_2, P2 can execute on $3 < 2^2$ time units before completion

Scheduling will go on

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	
Guaranteed Scheduling	yes	CPU use ratio	dynamic	no	Enforcing the response time	Time sharing
Round-Robin	yes	rank in the queue and round	dynamic	no		
Fair-Share Scheduling	yes	process priority	dynamic	no	Respecting the priority and enforcing the response time	Priority & time sharing
Multilevel feedback queue scheduling	yes	process priority and queue position	static/ dynamic	no		

Operating Systems
“Uniprocessor scheduling”

1. About short-term scheduling
2. Context switch, quantum and ready queue
3. Process and diagram models
4. Scheduling algorithms
    4.1. FCFS scheduling
    4.2. Priority based scheduling
    4.3. Optimal scheduling
    4.4. Time-sharing based scheduling
    4.5. Priority/Time-sharing based scheduling
5. Modeling multiprogramming
6. Evaluation of algorithms

Modeling multiprogramming

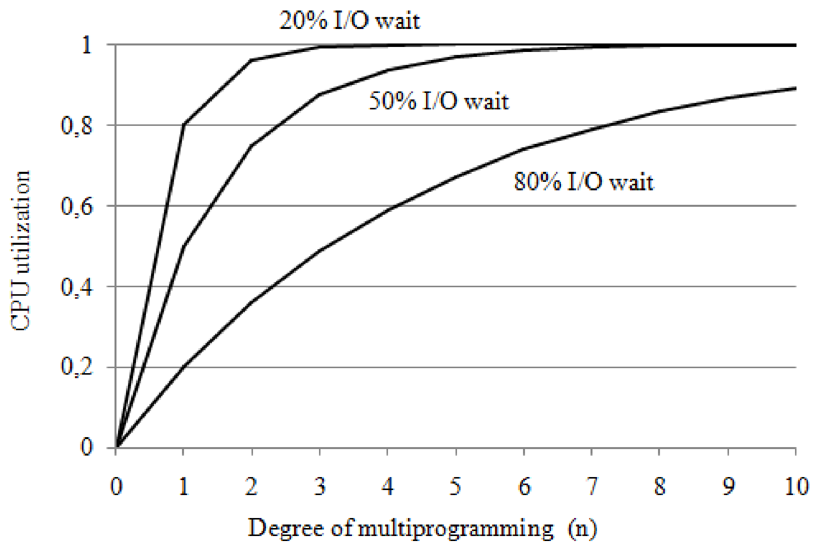
Modeling multiprogramming: from a probabilistic point of view, suppose that a process spends a fraction *p* of its time waiting for I/O to complete.

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%) $CPU\ utilization = 1 - (0{,}8 \times 06 \times 0{,}4 \times 0{,}6) = 0{,}8704$

Operating Systems
“Uniprocessor scheduling”

1. About short-term scheduling
2. Context switch, quantum and ready queue
3. Process and diagram models
4. Scheduling algorithms
    - 4.1. FCFS scheduling
    - 4.2. Priority based scheduling
    - 4.3. Optimal scheduling
    - 4.4. Time-sharing based scheduling
    - 4.5. Priority/Time-sharing based scheduling
5. Modeling multiprogramming
6. Evaluation of algorithms

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.

Random number generator

Recording a real system

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

process workload

P	$\mathbf{r_0}$	**C**
P1	0	10
P2	5	20
P3	3	3
P4	7	7
P5	9	12

Statistics

	$\overline{WT}$
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.