



**SAPIENZA**  
UNIVERSITÀ DI ROMA

**Facoltà di Ingegneria dell'Informazione, Informatica e  
Statistica  
Dipartimento di Informatica**

# **Sistemi Operativi 1**

**Autore:**  
Simone Lidonnici

7 settembre 2024

# Indice

<b>E</b>	<b>Esercizi</b>	<b>1</b>
E.1	Esercizi sulle memorie . . . . .	1
E.1.1	Allocazione di processi in blocchi . . . . .	1
E.1.2	Memoria divisa in pagine . . . . .	2
E.1.3	Pagina e offset di un indirizzo . . . . .	2
E.1.4	Controllare i page fault . . . . .	3
E.2	Calcolo dei tempi . . . . .	4
E.2.1	Tempo medio di waiting . . . . .	4
E.2.2	Tempo di accesso in memoria . . . . .	6
E.3	Dischi magnetici . . . . .	7
E.3.1	Tempo di trasferimento dati . . . . .	7
E.3.2	Scheduling del disco . . . . .	7

# E

## Esercizi

### E.1 Esercizi sulle memorie

#### E.1.1 Allocazione di processi in blocchi

Dato un processo  $P$  che necessita di una memoria libera di  $N$  KiB per essere allocato in modo contiguo, avendo blocchi liberi  $B_1, B_2, \dots, B_n$  di dimensione  $M_1, M_2, \dots, M_n$  il processo verrà allocato in blocchi diversi in base alla politica:

- **Best-Fit:** blocco più piccolo tale che  $M_i > N$
- **Worst-Fit:** blocco più grande tale che  $M_i > N$
- **First-Fit:** primo blocco tale che  $M_i > N$

#### Esempio:

Si supponga che un processo  $P$  necessiti di un'area di memoria libera pari a 115 KiB per essere allocato in modo contiguo in memoria principale. Se la lista dei blocchi di memoria libera contiene i seguenti elementi:

A	B	C	D	E	F
300 KiB	600 KiB	350 KiB	200 KiB	750 KiB	125 KiB

In base alla politica  $P$  verrà allocato:

- Best-Fit: F perché è il blocco minore in cui  $M_F > 115$  KiB
- Worst-Fit: E perché è il blocco maggiore in cui  $M_E > 115$  KiB
- First-Fit: A perché è il primo blocco in cui  $M_A > 115$  KiB

## E.1.2 Memoria divisa in pagine

### Memoria divisa in pagine

Data una memoria fisica di grandezza  $M$  con indirizzamento con word size  $w$  e la memoria sia divisa in pagine di grandezza  $S$ , la dimensione della page table, cioè il numero di entry è:

$$T = \frac{M}{S}$$

Invece il numero di bit necessari per indirizzare le word sarà:

$$\text{n° bit word} = \log_2\left(\frac{M}{w}\right)$$

Il numero di bit necessari per l'indice di pagina e per l'offset sono:

$$\text{n° bit offset} = \log_2\left(\frac{S}{w}\right)$$

$$\text{n° bit indice pagina} = \log_2\left(\frac{M}{S}\right)$$

#### Esempio:

Si supponga di avere una memoria  $M$  di capacità pari a 8 KiB (8192 B). Assumendo che l'indirizzamento avvenga con lunghezza di parola (word size) pari al singolo byte e che  $M$  utilizzi una gestione paginata con blocchi di dimensione pari a 128 B, quale dimensione (intesa come numero di entry) ha la corrispondente page table  $T$ ?

$$T = \frac{8192}{128} = 64$$

## E.1.3 Pagina e offset di un indirizzo

### Pagina e offset di un indirizzo

Data una memoria fisica di grandezza  $M$  con frame di grandezza  $F$ , l'indirizzo  $A$  avrà numero di pagina  $p$  e offset:

$$p = \lfloor \frac{A}{F} \rfloor$$

$$\text{offset} = A \% F$$

#### Esempio:

Si consideri una memoria  $M$  di capacità pari a 100 B con frame di dimensione pari a 10 B. Dato l'indirizzo del byte  $A=37$ , quale sarà l'indirizzo di pagina e l'offset?

$$p = \lfloor \frac{37}{10} \rfloor = 3$$

$$\text{offset} = 37 \% 10 = 7$$

### E.1.4 Controllare i page fault

Data una memoria con  $k$  frame e un processo con  $n$  pagine virtuali, per calcolare il numero di page fault bisogna prima controllare quali sono le pagine caricate in memoria (al massimo  $k$  insieme) e vedere se la pagina richiesta in quel momento è caricata in memoria, se non lo è si verifica un page fault. Quando una pagina viene richiesta e non è già caricata, se la memoria non è piena viene caricata senza sovrascriverne nessun'altra.

Nel caso in cui la memoria è piena quale pagina verrebbe sovrascritta dipende dal protocollo:

- **First In First Out (FIFO)**: viene sovrascritta la pagina che è caricata in memoria da più tempo
- **Last Recent Used (LRU)**: viene sovrascritta la pagina che non viene chiamata da più tempo
- **MIN (OPT)**: viene sovrascritta la pagina che non verrà acceduta per il lasso di tempo maggiore nel futuro

#### Esempio:

Data una memoria composta da 3 frame fisici e un processo composto da 5 pagine virtuali: A, B, C, D, E, si calcoli il numero di page fault che si verificano a fronte delle seguenti richieste da parte del processo: B, C, C, B, A, E, B, A, E, D, B. Si assuma che nessuna pagina del processo sia inizialmente caricata in memoria e che si utilizzi un algoritmo LRU di sostituzione delle pagine.

Pagina chiamata	Pagine in memoria	Page fault
B	/ / /	✓
C	/ / B	✓
C	/ B C	
B	/ B C	
A	/ C B	✓
E	C B A	✓
B	B A E	
A	A E B	
E	E B A	
D	B A E	✓
B	A E D	✓

In questo caso quindi il numero di page fault è 6.

## E.2 Calcolo dei tempi

### E.2.1 Tempo medio di waiting

#### Tempo medio di waiting

Data una coda di  $n$  processi, ognuno con un istante di arrivo  $T_{\text{arrival}}$  e un tempo di completamento CPU burst, il tempo di waiting di un determinato processo  $i$  si calcola:

$$T_{\text{waiting}}^i = T_{\text{completion}}^i - T_{\text{arrival}}^i - \text{CPU burst}^i$$

In cui  $T_{\text{completion}}$  è l'istante in cui termina il processo  $i$ .

Da cui il tempo medio di waiting dell'intera coda di  $n$  processi sarà:

$$\bar{T}_{\text{waiting}} = \frac{1}{n} \sum_{i=1}^n T_{\text{waiting}}^i$$

Per calcolare l'istante di completamento di un processo bisogna controllare la coda e cambia in base al protocollo usato:

- **First Come First Served (FCFS)**: viene eseguito per intero il processo con  $T_{\text{arrival}}$  minore e poi viene eseguito il successivo
- **Round Robin (RR)**: si inizia l'esecuzione dei processi in ordine secondo la coda ma fermandoli e passando al successivo ogni tot di tempo, detto **time slice**  $q$ . Ogni volta che un processo viene fermato viene aggiunto alla fine della coda.
- **Shortest Job First (SJF)**: si esegue il processo con il CPU burst minore e ci sono due tipi possibili:
  - **Non-Preemptive**: una volta che un processo è iniziato viene portato a termine
  - **Preemptive**: ogni volta che un nuovo processo arriva, se ha CPU burst minore rispetto a quello rimanente al processo in corso, viene eseguito questo nuovo processo

Nel caso in cui ci siano delle attività I/O di durata  $T_{\text{I/O}}$  il tempo di waiting di quel determinato processo diventa:

$$T_{\text{waiting}}^i = T_{\text{completion}}^i - T_{\text{arrival}}^i - \text{CPU burst}^i - T_{\text{I/O}}^i$$

**Esempio 1:**

Data una coda dei processi:

Job	$T_{\text{arrival}}$	$T_{\text{burst}}$
A	0	3
B	2	7
C	6	4
D	7	5

1. FCFS:

A			B							C				D				
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19

Il tempo di waiting sarà:

- $T_{\text{waiting}}^A = 3 - 0 - 3 = 0$
- $T_{\text{waiting}}^B = 10 - 2 - 7 = 1$
- $T_{\text{waiting}}^C = 14 - 6 - 4 = 4$
- $T_{\text{waiting}}^D = 19 - 7 - 5 = 7$

$$\bar{T}_{\text{waiting}} = \frac{0 + 1 + 4 + 7}{4} = 3$$

2. RR con  $q = 4$ :

A			B				C				D				B			D
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19

Il tempo di waiting sarà:

- $T_{\text{waiting}}^A = 3 - 0 - 3 = 0$
- $T_{\text{waiting}}^B = 18 - 2 - 7 = 9$
- $T_{\text{waiting}}^C = 11 - 6 - 4 = 1$
- $T_{\text{waiting}}^D = 19 - 7 - 5 = 7$

$$\bar{T}_{\text{waiting}} = \frac{0 + 9 + 1 + 7}{4} = 4.25$$

**Esempio 2:**

Data una coda dei processi:

Job	$T_{\text{arrival}}$	$T_{\text{burst}}$
A	0	6
B	1	3
C	2	7
D	3	4

## 1. SJF Non-Preemptive:

A						B			D				C						
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20

Il tempo di waiting sarà:

- $T_{\text{waiting}}^A = 6 - 0 - 6 = 0$
- $T_{\text{waiting}}^B = 9 - 1 - 3 = 5$
- $T_{\text{waiting}}^C = 20 - 2 - 7 = 11$
- $T_{\text{waiting}}^D = 13 - 3 - 4 = 7$

$$\overline{T}_{\text{waiting}} = \frac{0 + 5 + 11 + 7}{4} = 5.5$$

## 2. SJF Preemptive:

A		B			D				A					C					
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20

Il tempo di waiting sarà:

- $T_{\text{waiting}}^A = 13 - 0 - 6 = 7$
- $T_{\text{waiting}}^B = 4 - 1 - 3 = 0$
- $T_{\text{waiting}}^C = 20 - 2 - 7 = 11$
- $T_{\text{waiting}}^D = 8 - 3 - 4 = 1$

$$\overline{T}_{\text{waiting}} = \frac{7 + 0 + 11 + 1}{4} = 4.75$$

**E.2.2 Tempo di accesso in memoria****Tempo di accesso in memoria**

Data una memoria fisica con tempo di accesso in memoria  $t_{\text{MA}}$ , tempo di gestione di un page fault  $t_{\text{FAULT}}$  che accade con probabilità  $p_{\text{FAULT}}$ , il tempo di accesso medio sarà:

$$t_A = (1 - p)t_{\text{MA}} + p \cdot t_{\text{FAULT}}$$

La probabilità è scritta come un numero  $p \in [0, 1]$ , per convertirlo in percentuale bisogna moltiplicarlo per 100.



## E.3 Dischi magnetici

### E.3.1 Tempo di trasferimento dati

#### Tempo di trasferimento dati

Dato un disco magnetico in cui si devono trasferire dei dati, con tempo di posizionamento Seek, delay di rotazione ROT e tempo di trasferimento TT, il tempo totale di trasferimento dei dati sarà:

$$T = \text{Seek} + \text{ROT} + \text{TT}$$

In cui TT dipende dalla velocità di trasferimento TR e dalla quantità di dati trasferiti D:

$$\text{TT} = \frac{D}{\text{TR}}$$

#### Esempio:

$$T = 40 \text{ ms}$$

$$\text{Seek} = 18 \text{ ms}$$

$$\text{ROT} = 7 \text{ ms}$$

$$\text{TR} = 5 \text{ Gb/s}$$

$$40 \text{ ms} = 18 \text{ ms} + 7 \text{ ms} + \frac{D}{5 \text{ Gb/s}} \implies D = 15 \text{ ms} \cdot 5 \text{ Gb/s} = 15 \text{ ms} \cdot 5 \text{ Mb/ms} = 75 \text{ Mb} \\ = 9.375 \text{ MB}$$

### E.3.2 Scheduling del disco

Dato un disco con  $k$  tracce (numerate da 0 a  $k$ ) la cui testina è inizialmente posizionata sulla traccia  $x_0$  ed effettuiamo delle richieste  $x_1, x_2, \dots, x_n$  la testina percorrerà un certo numero di tracce per soddisfarle tutte.

L'ordine in cui le richieste vengono soddisfatte dipende dall'algoritmo:

- **First Come First Served (FCFS)**: la testina si muove semplicemente in ordine in base alle richieste
- **Shortest Seek Time First (SSTF)**: la testina ogni volta che si deve muovere lo fa verso la richiesta più vicina alla posizione in cui si trova in quel momento
- **SCAN**: la testina si muove da un'estremo all'altro (da 0 a  $k$  e poi torna indietro) del disco e esegue le richieste durante il passaggio
- **LOOK**: la testina si muove da un'estremo all'altro ma cambiando direzione quando raggiunge la richiesta più esterna
- **C-SCAN**: la testina si muove da un'estremo all'altra ma leggendo solo solo in una direzione (da  $k$  a 0 e poi torna indietro senza leggere per poi rileggere solo durante il tragitto tra  $k$  e 0)
- **C-LOOK**: la testina si muove da un'estremo all'altra ma leggendo solo solo in una direzione ma cambiando direzione quando raggiunge la richiesta più esterna

**Esempio:**

Avendo un disco con 100 tracce effettuiamo le richieste 65, 40, 18, 78 e la testina è inizialmente posizionata sulla traccia 30 (i numeri sulle frecce indicano il numero di tracce percorse durante quello spostamento):

- FCFS:  
 $30 \xrightarrow{35} 65 \xrightarrow{25} 40 \xrightarrow{22} 18 \xrightarrow{60} 78 \implies \text{n}^\circ \text{ passi} = 35 + 25 + 22 + 60 = 142$
- SSTF:  
 $30 \xrightarrow{10} 40 \xrightarrow{22} 18 \xrightarrow{47} 65 \xrightarrow{13} 78 \implies \text{n}^\circ \text{ passi} = 10 + 22 + 47 + 13 = 92$
- SCAN:  
 $30 \xrightarrow{12} 18 \xrightarrow{18} 0 \xrightarrow{40} 40 \xrightarrow{25} 65 \xrightarrow{13} 78 \implies \text{n}^\circ \text{ passi} = 12 + 18 + 40 + 25 + 13 = 108$
- LOOK:  
 $30 \xrightarrow{12} 18 \xrightarrow{22} 40 \xrightarrow{25} 65 \xrightarrow{13} 78 \implies \text{n}^\circ \text{ passi} = 12 + 22 + 25 + 13 = 72$
- C-SCAN:  
 $30 \xrightarrow{12} 18 \xrightarrow{18} 0 \xrightarrow{100} 100 \xrightarrow{22} 78 \xrightarrow{13} 65 \xrightarrow{25} 40 \implies \text{n}^\circ \text{ passi} = 12 + 18 + 100 + 22 + 13 + 25 = 190$
- C-LOOK:  
 $30 \xrightarrow{12} 18 \xrightarrow{60} 78 \xrightarrow{13} 65 \xrightarrow{25} 40 \implies \text{n}^\circ \text{ passi} = 12 + 60 + 13 + 25 = 110$