Reti e Sistemi Operativi

- Reti:
    1) Introduzione
    2) Application Layer
    3) Transport Layer
    4) Network Layer
    5) Link Layer

- Sistemi Operativi
    1) Struttura e Servizi
    2) Processi e Thread
    3) Scheduling cpu
    4) Gestione Memoria
    5) File System

Reti: Introduzione

Funzionalità Host

- forwarding: azione locale che mi informa su quale strada devo mandare un pacchetto
- routing: azione globale che attraverso algoritmi inoltra il pacchetto.

Store and forward: prima di poter inviare un pacchetto devo averlo ricevuto nell'interezza

Packet Transmission Delay: Per trasmettere un pacchetto su un link impieghiamo L/R secondi.

Queueing: Succede quando riceviamo pacchetti più velocemente di quanto li inviamo. Perdo informazioni.

Layer Protocol: architettura a strati: struttura gerarchica. Ogni livello offre un servizio sfruttando i servizi del livello precedente.

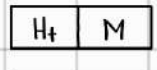
Protocollo TCP/IP:

- application: supportano le applicazioni di rete. (DNS)
- transport: trasferimento di dati tra processi diversi su host differenti. (TCP, UDP)
- network: inoltro e instradamento dei pacchetti tra sorgente e destinazione (IP, protocolli routing)
- link: trasferimento di dati tra nodi adiacenti
- physical: trasferimento bit su cavo. (non vedremo).

Incapsulamento: come avviene lo scambio di informazioni tra source e destination:

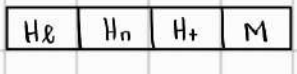
application: scambiano messaggi attraverso i servizi offerti da trasporto

transport: aggiunge header per offrire i servizi. Abbiamo un segmento



network: aggiunge header per offrire i servizi. Abbiamo un datagramma

link: aggiunge header per offrire i servizi. Abbiamo un frame



rete: compie il processo di instradamento.

Reti: Application layer

DNS: Domain Name System: traslazione degli indirizzi host-name in indirizzi IP.

Struttura:

Root DNS Server — Root

.com .org .edu — TLD

amazon.com pbs.org nyu.edu — Server Authoritative

Herated query: Può contattare direttamente i server root, TLD ed altri

Recursive query: Deve passare da un root DNS che comunicherà con gli altri.

Reti: Introduzione

Ritardo trasmissione (latenza) d_{TRANS}: $\frac{L}{R}$ [s]

L = lunghezza pacchetto [b]

R = banda [b/s]

Ritardo propagazione d_{PROP}: $\frac{d}{s}$ [s]

d: distanza tra due nodi [m]

s: velocità propagazione [$\frac{m}{d}$]

Intensità traffico, ritardo accodamento: $\frac{L \cdot a}{R}$

a: tasso medio arrivo pacchetti [$\frac{n° p}{s}$]

Numero pacchetti: $\frac{L}{\ell}$

① $L = 500$ byte, $R = 1$ Gbps, $s = 300 \cdot 10^6$ m/s. Trovare d

$\frac{L}{R} = \frac{d}{s}$ quindi $d = sL/R = 300 \cdot 10^6 \text{ m/s} \cdot \frac{500 \cdot 8 \text{ b}}{1 \cdot 10^9 \text{ b/s}} = 12 \cdot 10^3$ m

② $L = 1500$ byte, $R = 100$ Mbps, trova a tc $\frac{aL}{R} = 1$

$\frac{R}{L} = \frac{100 \cdot 10^6 \text{ b/s}}{1500 \cdot 8 \text{ b}} = 8500$ pacchetti/s valore minimo necessario

Store and forward: no frammentazione d_R: $Q \cdot \frac{L}{R}$ [s]

si frammentazione d_R: $\frac{L}{R} + (Q-1)\frac{\ell}{R}$ [s]

Q n° link

ℓ quantità dati singolo pacchetto

③ $L = 7.5$ Mbit, $Q = 5$, $R = 1.5$ Mbit/s, $\ell = 1500$ bit. d_R?

$d_R = 5 \cdot \frac{7.5 \cdot 10^6 \text{ b}}{1.5 \cdot 10^6 \text{ b/s}} = 25$ s no framm

$d_R = \frac{7.5 \cdot 10^6 \text{ b}}{1.5 \cdot 10^6 \text{ b/s}} + 4 \frac{1500 \text{ b}}{1.5 \cdot 10^6 \text{ b/s}} = 5 + 0.0004 \simeq 5$ ms

Reti: Transport Layer

Transport Layer: Fornisce comunicazione logica tra applicativi, ovvero permette di far comunicare due processi su due macchine come se fossero sulla stessa. Le azioni intraprese sono:

- sender: divide i messaggi in segmenti e passa al layer rete
- receiver: riassembla i segmenti e passa al layer applicazione

Socket: Interfaccia che permette di far comunicare i processi che scambiano messaggi

Multiplexing: Gestisce dati da più socket aggiungendo l'header trasporto.
Demultiplexing: Usa l'header per consegnare i pacchetti ai socket corretti.

Connessione: Due tipi di connessione:

- connectionless: creazione dei socket specificando IP: dest. port. Tutte le comunicazioni arrivano allo stesso socket.
- connection-oriented: socket identificati da src IP: src port e dest IP: dest. port (1 socket = 1 client)

Affidabilità: Definiamo con ACK il corretto ricevimento del pacchetto e con NACK la ricezione non corretta del pacchetto. Questi possono contenere errori che vengono controllati con:

- stop and wait: Ogni segmento è numerato, si usa un timer per gestire le perdite
- sliding window: Si inviano più pacchetti in sequenza, la finestra non chiude prima del 1° ACK. Quando perdiamo un pacchetto possiamo usare:
  - Go-back-N: se a tempo S+T non arriva ACK allora ritorniamo indietro.
  - Selective Repeat: si rinvia solo il pacchetto perso.

  Entrambi presentano un buffer sul lato sender.

TCP e UDP: Protocolli diversi, ma entrambi offrono controllo di errore ed inoltro dei dati alla applicazione.

Reti: Transport Layer

UDP: User Data Protocol, usato in applicazioni real-time non sensibili alla perdita di dati:
- best effort: non garantisce consegna ordinata di pacchetti.
- no congestion controll: congestioni fanno perdere pacchetti
- connectionless: senza handshake, semplice e header ridotto. ma non garantisce servizi su ritardi e latenza.

TCP: Transmission Control Protocol: usato in applicazioni dove la perdita è importante:
- point-to-point: un mittente e un destinatario.
- affidabile: consegna in-order
- controllo congestione e flusso
- handshake

Sequence Number: Controllo di sequenza attraverso due numeri:
- sequence number: 1° byte segmento
- acknowledgments: n° next byte atteso

RTT: Round-Trip-Time: tempo che impiega un pacchetto per "viaggiare". Si usano i parametri:
- Sample RTT: tempo tra trasmissione e ricevimento dell' ACK.
- Estimated RTT: $(1-\alpha)\cdot$ Estimated RTT $+ 2\cdot$ Sample RTT.
- TimeOut Interval: Estimated RTT $+ 4\cdot$ DevRTT
- DevRTT: $(1-\beta)\cdot$ DevRTT $+ \beta\cdot |$ Sample RTT $-$ Estimated RTT $|$

Fast Retransmit: se il mittente riceve 3 ack uguali, rinvia UNACKED con più piccolo sequence number.

Flow control: Modera velocità trasmissione attraverso rwnd, ovvero lo spazio libero del buffer ricevente

Reti: Transport Layer

Connection Managment: Si effettua un three-way-handshake per stabilire parametri e connessione. Si conclude con FINbit=1 e rispondendo con ACK.

Controllo Congestione: Avviene quando più utenti inviano dati che la rete non riesce a gestire.

Può essere gestita da:

- end-to-end: gestito da due host
- network assisted: guidata da feedback del router.

È gestita con:

- AIMD: successo incremento RTT, perdita dimezzo.
- SLOW START: aumento esponenziale di cwnd. Usa ssthresh con valore predefinito.

TCP Reno: automa per gestire il flusso e la congestione tramite 3 stati:

- Slow start
- Congestion Avoidance
- Fast Recovery

Reti: Transport Layer

Utilizzo canale: $\mathcal{U} \simeq \frac{N \cdot L/R}{RTT + L/R}$

Latenza: $2 \cdot RTT + \frac{O}{R}$ [s]

Throughput medio: $\frac{\frac{3}{4} \cdot N \cdot L}{RTT}$

N dimensione finestra

O dimensione oggetto

Dim. media finestra: $1 + \frac{R \cdot RTT}{MSS}$

Tasso perdita pacchetti: $\frac{1}{\frac{3}{8} \cdot N^2}$

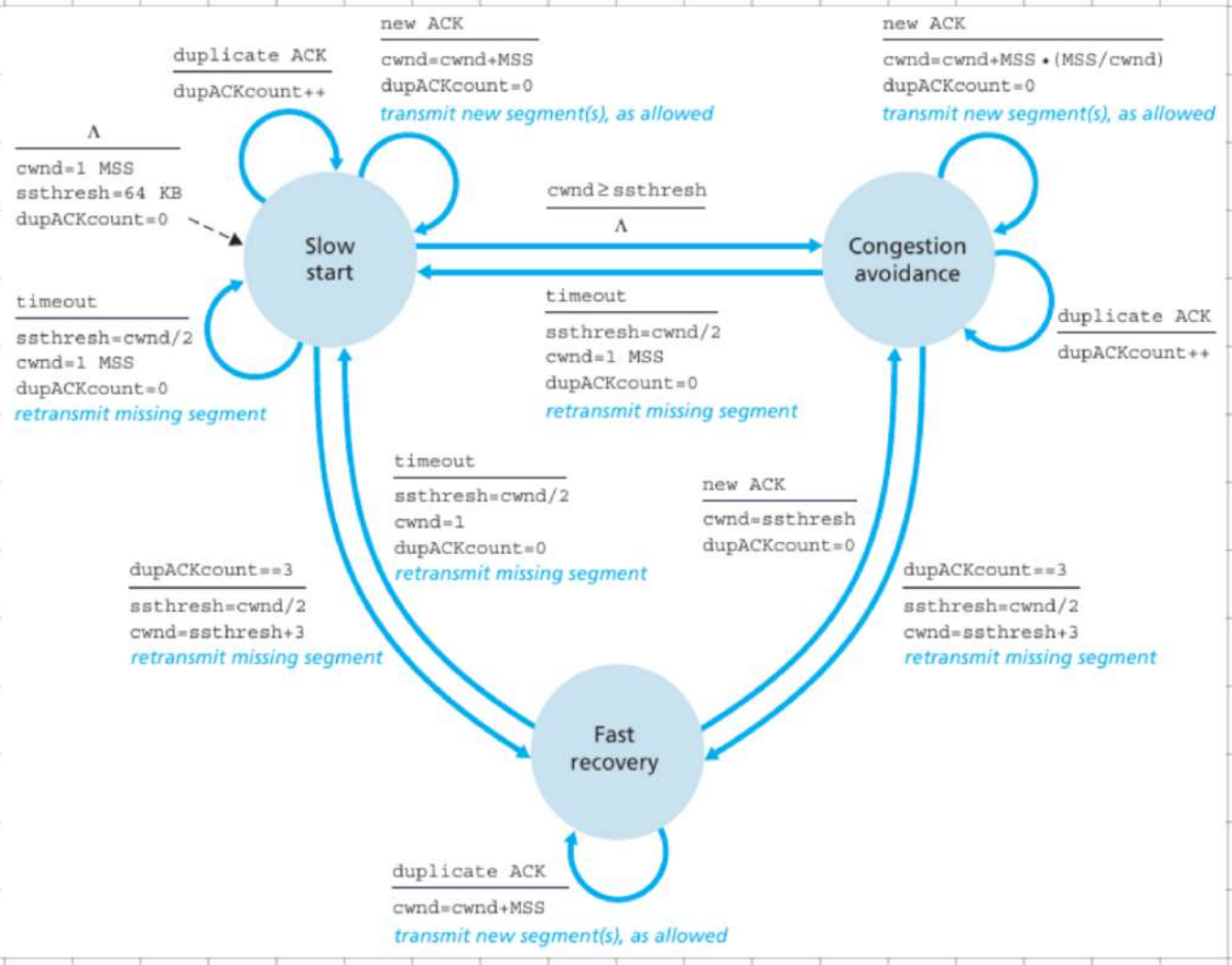
MSS: maximum segment size

cwnd max: $\left\lfloor \frac{RTT \cdot R_{max}}{MSS} \right\rfloor$

cwnd minima: $\left\lceil \frac{\text{cwnd max}}{2} \right\rceil + 3$

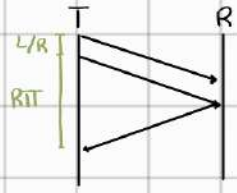
Tempo invio (slow start): $\frac{L}{R} + RTT$

tempo dopo perdita pacchetto x tornare a cwnd max: (cwnd max - cwnd min) · RTT



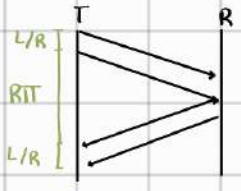
Reti: Transport Layer

① $R = 640$ kbps, $RTT = 40$ ms, $L = 1000$ byte, no riscontro, w?



$\frac{N \cdot L/R}{RTT + L/R} = 1$ $\quad N = 640 \cdot 10^3$ bps $/ 8000$ b $\cdot 40 \cdot 10^{-3} + 1 = 4,2$

② $R = 640$ kbps, $RTT = 40$ ms, $L = 1000$ byte, si riscontro dopo 1000 byte, w?



$\frac{N \cdot L/R}{RTT + 2L/R}$

$w = R/L * RTT + 2 = 640 \cdot 10^3$ bps $/ 8000$ b $\cdot 40 \cdot 10^{-3} + 2 = 6,2$

③ $O = 14000$ byte, ssthresh $= 4$, $R = 1$ Mbps, $RTT = 30$ ms, $MSS = 1000$ byte

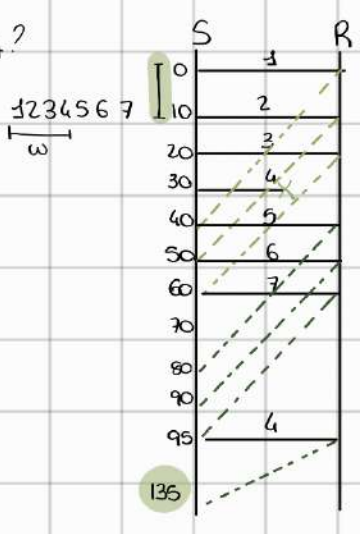
cwnd	B-TX	TF	
1	1000	TF1	$TF1 = MSS/R + RTT = 1000 \cdot 8$ b $/ 10^6$ bps $+ 0.030$ s $= 38$ ms
2	3000	TF1	$MSS/R + RTT < TF2 = cwnd \cdot MSS/R \rightarrow 0.038 < 0.016$ × trasmissione discontinua
4	7000	TF1	$MSS/R + RTT < TF3 = cwnd \cdot MSS/R \rightarrow 0.038 < 0.030$ × trasmissione discontinua
5	12000	TF4	$0.038 < TF4 = 5 \cdot MSS/R \rightarrow 0.038 < 0.040$ ✓ trasmissione continua
6	14000	TF5	$TF5 = 2 * MSS/R = 0.016$ non devo confrontarlo

$T_{tot} \left(\sum_{i=1}^{5} T_i\right) + RTT = 180$ ms

④ $L = 6500$ byte, $w = 4000$ byte, $MSS = 1000$ byte, $R = 800$ Kbps, $RTT = 30$ ms. T_{tot}?

$MSS/R + RTT = 8000$ bit $/ 800 \cdot 10^3$ Kbps $+ 30$ ms $= 10 + 30 = 40$ ms × 1 segmento

6500 byte, quindi 6 segmenti da 1000 byte e 1 da 500 byte



Reti: Transport Layer

⑤ Client invia a server seq n = 112 e ACK = 115, dim = 30 byte. Valori al ritorno? seq n: 115, ACK: 142

⑥ RTT 15 ms, MSS = 2000 byte R = 2 MBps

a. cwnd max: RTT · R = 0.015 s · 2 · 8 · 10^6 bps = 240 Kbit. cwnd max / MSS = 240 Kbit / 2000 bit = 120 segmenti

b. valore partenza cwnd con 3 ack duplicati? Tahoe cwnd = 1 MSS, cwnd Reno = non lo sappiamo

c. valore partenza cwnd con timeout? cwnd Tahoe = cwnd Reno = 1 MSS

d. 25 segmenti, slow start = 10 segmenti, 4° segmento perso per 3 ACK, 19° per timeout.

	step	cwnd	ssthresh	segmenti	
SS	1	1	10	1	
SS	2	2	10	2, 3	
SS	3	4	10	[4], 5, 6, 7	3 ack
fr	4	5 →	2	4, 8	
CA	5	2	2	9, 10	
CA	6	3	2	11, 12, 13	
CA	7	4	2	14, 15, 16, 17	
CA	8	5	2	18, [19], 20, 21, 22	timeout
SS	9	1 →	3	19	
SS	10	2	3	23, 24	
SS	11	4	3	25	

⑦ R = 12 Mbps RTT = 150 ms, MSS = 1500 B

a. cwnd max = $\frac{RTT \cdot R}{MSS}$ = (150 · 10^{-3} s · 12 · 10^6 bps) / 1500 · 8 b = 150 segmenti

b. cwnd min = $\left\lceil \frac{\text{cwnd max}}{2} \right\rceil + 3 =$ 78, cwnd avg = $\left\lfloor \frac{150+78}{2} \right\rfloor$ = 114 segmenti

c. Vavg = (cwnd avg * MSS) / RTT = (114 · 1500 · 8) / 100 · 10^{-3} = 9.12 Mbps

d. tempo a cwndmax dopo loss: (cwnd max - cwnd min) · RTT = 72 · 15 · 10^{-3} = 10.8 s

Reti: Network layer - data

Data plane: azione locale: determina come i datagrammi vengono spostati tra sender e receiver.

Service model: best effort, non è garantita consegna in ordine, latenza minima e cadenza su pacchetti.

Architettura porta Input:
- line termination: livello fisico, riceve bit
- link layer protocol: Ethernet
- lookup forwarding: Attraverso header

Architettura porta output:
- buffering: quando i datagrammi arrivano al fabric più velocemente del link transm. rate.

Switching fabrics: trasferisce protocolli da input a output con switching rate NR per N porte e R rates.

Overhead IP: IP unico protocollo, overhead 20 TCP + 20 IP = 40 bytes.

Indirizzo IP: identificatore 32 bit associato ad ogni interfaccia host o router.

Subnet: sottoreti che si connettono senza passare da router. Ogni indirizzo ha subnet e host part.

Classi: gli IP si possono classificare in due modi:
- Classless InterDomain Routing: nella forma a.b.c.d/x dove x n° bit dedicati a subnet.
- Classful: con notazione a.b.c.d, se $0 < a \leq 127$ classe A: 8 S/24 H. $128 \leq A \leq 191$ 16S/16H, $192 < A < 223$ classe C, 24S, 8H. Non si usa subnet mask.

Ottenimento IP: si ottiene l'indirizzo con:
- Statico: specificato in configurazione
- Dhcp: ottenuti dal server nelle fasi: discover - host cerca server, Server apre a indirizzo, request, ACK.

NAT: unico indirizzo con cui gli host possono comunicare con l'esterno.

Reti: Network layer-control

Control Plane: determina come i datagrammi viaggiano tra router. Attraverso il protocollo di routing si determina il percorso migliore, attraverso dei grafi.

Classificazione algoritmi routing: possono essere globali dove tutti hanno conoscenza sui costi altrui come il link state, o decentralizzato dove si conoscono solo i vicini, come distance vector. Inoltre possono essere statici dove le route cambiano poco o dinamici dove cambiano spesso.

Distance vector: $dist(A,X) = \min\{dist(V,x) + c(A,V)\}$.

Link state: ciascun router comunica il proprio link state. Usa l'algoritmo di Dijkstra.

Routing scalabili: routing con miliardi di destinazioni che non può memorizzare. Si aggregano router in anonymus systems. Intra-AS: tutti i router hanno lo stesso protocollo. Inter-AS: collega i vari AS.

Reti: Network layer-data

① Definire IPv4 pvt classe C e definire n° max host e rete

Classe C: 192-223. 192.168.0.0 - 192.168.255.255 pvt

Abbiamo N.N.N.H quindi 2^8-2 dispositivi per host e $2^{24}-2$ per rete

Indirizzo di rete è 192.168.0.0, maschera: 255.255.255.0, broadcast: 192.168.0.255.

② meno spreco per 70 dispositivi

$2^7-2 = 126 > 70$ ✓ quindi 192.168.0.nhhhhhhh → 255.255.255.128 subnet mask

rete: 192.168.0.0, broadcast 192.168.0.127 broadcast

③ Quale indirizzo host non è valido:

- 10.10.10.77/14 → N.nnnnnnhh.HH → 255.252.0.0 subnet mask.

ind. rete: 10.10.10.77 and 255.252.0.0 → 10.8.0.0 ≠ rete, ≠ broadcast valido

- 133.66.56.96/27. N.N.N.nnnhhhhh → 255.255.255.224, rete: 133.66.56.96/24 non valido!!

96 → 01100000
224 → 111 00000
] 011 00000

- 230.50.50.2/19 valido, no rete perché abbiamo .H nell'ultimo ottetto ≠ 0 rete e 255 broadcast

④ Primo e ultimo host dell'intervallo 172.31.78.88/23.

N.N.nnnnnnh.H, 255.255.254.0. and tra 78 e 254 = 78, ind rete: 172.31.78.0

metto bit h = 1 ottengo 79 e 255 quindi: 172.31.79.255.

Range ottenuto 172.31.78.1 - 172.31.79.254

⑤ Ultimo host della sottorete di 10.117.238.0\20.

10.17.1110 1110.0 and 1.1.1111 0000.0 = 10.117.224.0 ind. rete subnet. broadcast: 10.117.1110 1111.255

↓

10.117.239.254

Reti: Network Layer - data

© ind 10.1.0.0 subnet 255.255.248.0 n° possibili sottoreti

248 = 11111000 → N.N.nnnnnhhh.H → $2^5 = 32$ pos. sottoreti. possibili host x sottorete: $2^{11} - 2 = 2096$

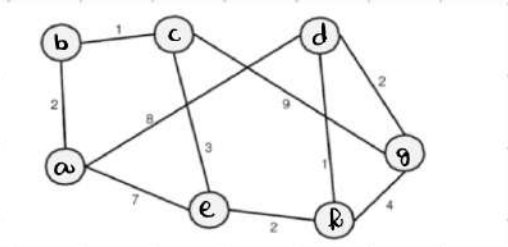
Le sottoreti saranno 10.1.0.0 rete, 10.1.7.255 broadcast

10.1.8.0 rete, 10.1.15.255 broadcast

10.1.16.0 rete, 10.1.23.255 broadcast

Reti: livello Rete - piano controllo

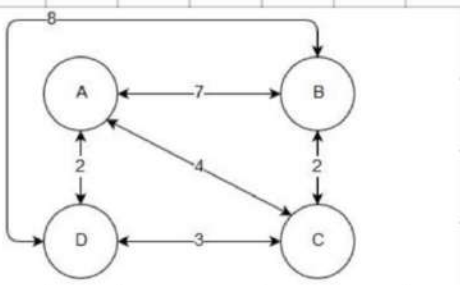
① Dijkstra



- passo
- aggiorno
- vecchio

Passo	1 a	2 b	3 c	4 d	5 e	6 f	7 g
1	a	a,2	∞	a,8	a,7	∞	∞
2	a,b	/	b,3	a,8	a,7	∞	∞
3	a,b,c	/	/	a,8	c,6	∞	c,12
4	a,b,c,e	/	/	a,8	/	e,8	c,12
5	a,b,c,e,d	/	/	/	/	e,8	d,10
6	a,b,c,e,d,f	/	/	/	/	/	d,10

② Distance vector



A	A	B	C	D
A	0	7	4	2
B				
C				
D				

B	A	B	C	D
A				
B	7	0	2	8
C				
D				

C	A	B	C	D
A				
B				
C	4	2	0	3
D				

D	A	B	C	D
A				
B				
C				
D	2	8	3	0

D	A	B	C	D
A	0	~~7~~ 6	4	2
B	~~7~~ 6	0	2	~~8~~ 5
C	4	2	0	3
D	2	~~8~~ 5	3	0

aggiorno
aggiorno
non aggiorno
aggiorno

$T_{tot} = T_i + T_r = 4 + 3 = 7$

Reti: livello Rete - piano controllo

③ Tabella Routing

La tabella di routing di un router contiene dei valori nel seguente formato: *indirizzo di rete, maschera di sottorete, indirizzo del next-hop*. Vengono di seguito riportate le righe della tabella già configurate:

- 10.1.0.0 255.255.255.0 192.168.2.2
- 10.1.0.0 255.255.0.0 192.168.3.3
- 0.0.0.0 0.0.0.0 192.168.1.1

Dati in ingresso ad una generica interfaccia del router i seguenti pacchetti con indirizzo di destinazione, indicare a quale next-hop vengono instradati:

- 10.1.1.10
- 10.1.0.14
- 10.2.1.3
- 10.1.4.6
- 10.1.0.123
- 10.6.8.4

Longest prefix-match: guardo chi ha la subnet più simile:

10.1.1.10 255.255.0.0 255.255.255.0
✓ ✓ ✓
✓ ✓ ✗

10.2.1.13 default route

10.1.4.6
✓ ✓ ✓
✓ ✓ ✗

10.1.0.4
✓ ✓ ✗
✓ ✓ ✓

Reti: Link Layer

Servizi: Trasferisce i datagrammi da un nodo a quello adiacente e diventa frame.

Tipi link: collegamento di 3 tipi per lo scambio dei dati: wired, wireless, LANs.

Protocolli di accesso: può essere point-to-point come ethernet o broadcast con link condiviso.

Tipi di broadcast: 3 tipi: channel partitioning: partizione del canale ad uso esclusivo di un singolo nodo.
random access e a turni.

Slotted ALOHA: di tipo RA, ogni nodo ha slot di tempo nel quale può trasmettere.
Se si presenta collisione, si ritrasmette fino al successo. Efficienza: 37%
Pure ALOHA: senza slot e sincronizzazione. aumento collisioni, + semplice. Efficienza: 18%.

CSMA: ascolto prima di invio. Se vuoto trasmetto frame intero, altrimenti trasm. differita.
CSMA/CD: con collision detected. Migliore dell' ALOHA.

Polling: protocollo a turni, abbiamo un nodo master che gestisce i dumb. Overhead, latenza, 1 punto fail.
Token Passing: Un token è passato tra i nodi per usare canale. Stessi difetti.

Ethernet: connectionless e unreliable (no handshake e ACK).

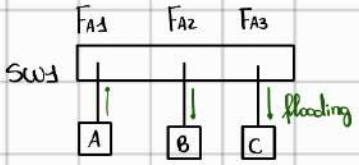
Switch: dispositivo che svolge store & forward dei frame, inoltro pacchetti a MAC, plug & play, evita collisioni.

Switching: registra MAC, interroga la switch table, se trova indirizzo invia frame altrimenti lo inoltra
a tutti i collegamenti.

Reti: Link Layer

Interfaccia	*Dispositivo collegato*
Fa1	Host A
Fa2	Host B
Fa3	Host C

Assumendo che i dispositivi terminali abbiano un'unica interfaccia di rete, e che nessun dispositivo abbia ancora trasmesso alcun frame, determinare il contenuto della MAC address table di SW1 e quali sue interfacce sono coinvolte da un inoltro se l'host A trasmette un frame all'host B.



mac address table sw1: Fa1 mac_addr_A

interfacce coinvolte: Fa2, Fa3

Dispositivo switch SW1:

Interfaccia	*Dispositivo collegato*
Fa1	Host A
Fa2	Host B
Fa3	SW2

Dispositivo switch SW2:

Interfaccia	*Dispositivo collegato*
Fa1	SW1
Fa2	Host C

Ipotizzando che nessun dispositivo abbia ancora trasmesso alcun frame, indicare quali porte degli switch sono interessate da un inoltro quando avvengo in sequenza le seguenti trasmissioni:

1. Da A verso B
2. Da A verso C
3. Da B verso C

1) A→B: sw1: Fa1 macA — sw2: Fa1 macA

2) A→C: sw1: Fa1 macA — sw2: Fa1 macA

3) B→C: sw1: Fa1 macA, Fa2 macB — sw2: Fa1 macA, macB

Si consideri la topologia di rete LAN Ethernet riportata in figura 2. Determinare il contenuto dei campi che definiscono l'indirizzamento a livello 2 e a livello 3 quando:

- A trasmette una richiesta ARP per trasmettere al dispositivo B
- A riceve una risposta ARP alla richiesta del punto precedente
- A trasmette al dispositivo B

1) dest MAC addr: FF:FF:FF:FF:FF:FF — src MAC addr: macA
dest IP: ip B — src IP: ip A

2) dest mac address: mac A — src mac address: mac R1
dest IP: ip A — src IP: ip R1

3) dest mac addr: mac R1 — src mac addr: mac A
dest ip: ip B — src IP: ip A

Reti: Link Layer

Si consideri una rete Ethernet a 10 Mbps; effettuare una stima del tempo di trasmissione di un oggetto di dimensione 5 Mbyte tenendo conto degli overhead introdotti dai livelli della pila protocollare. In particolare, Ethernet per il livello 2, IP per il livello 3, TCP e UDP per il livello 4.

$L = 5$ Mbyte $= 40 \cdot 10^6$ bit

$R = 10 \cdot 10^6$ byte

- Ethernet: 26 byte + 12 byte (IFS) = 38 byte

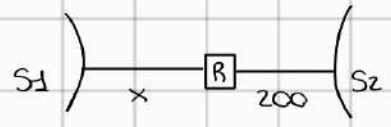
max payload: 1500 / (38 + 1500) = 4.1 secondi

- IP/TCP: 20 byte + 20 byte = (1500 - 40)/(38 + 1500) = 4.2 secondi
- IP/UDP: 20 byte + 8 byte = (1500 - 28)/(38 + 1500) = 4.17 secondi

Si considerino due segmenti di rete Ethernet ad accesso condiviso con velocità pari a 10 Mbps. I due segmenti sono collegati da un dispositivo ripetitore (R) che si limita a ritrasmettere quanto ricevuto da un segmento, in uscita a tutte le sue interfacce. Sul primo segmento (S1) è collegata una stazione posta ad X metri da R mentre sul secondo segmento (S2) ne è collegata una a distanza di 200 metri. Considerando che:

- S1 presenta un ritardo RTT di 0,1026 bit/m
- S2 presenta un ritardo RTT di 0,0866 bit/m
- R introduce un ritardo di elaborazione pari a 8 µs

Valutare, al variare di X, cosa succede se al tempo t = 0 µs S2 comincia a trasmettere un frame, mentre S1 comincia, nello stesso istante, il carrier sense per valutare se il canale è libero.



$D_1 = (x \cdot 0.0513 \text{ bit/m})/10 \text{ ms}$ D_2: $0.0433 \text{ bit/m} \cdot 200 \text{ m} = 8.66 \text{ bit}/10 \text{ ms} = 0.866 \text{ ms}$.

$/2$

$D_{tot} = D_R + D_2 + D_1 = 8 + 0.866 + x \cdot 0.0513$

se $t_1 + IFS < t_2 + D_{tot}$

$0 + 9.6 < 0.866 + x \cdot 0.0513$

$x > 143 \text{ m}$

Sistemi Operativi: Struttura e Servizi

Sistema Operativo: insieme di programmi che gestisce gli elementi fisici del computer.
Fornisce un'ambiente omogeneo attraverso regole alla quale le applicazioni devono attenersi.
Fornisce una piattaforma di sviluppo per i programmi applicativi.

Maledizione delle generalità: Quando un so cerca di funzionare su dispositivi diversi con diversi scopi, le performance saranno scarse.

Struttura SO: Un so comprende almeno
- Kernel utilizzato e crea le chiamate di sistema delle applicazioni.
- Middleware: gestisce le chiamate con framework e librerie per renderle più semplici. (API)
- Programmi di Sistema: Utilizzano i servizi offerti dal middleware e vengono usati dall'utente

Chiamate di sistema: Funzioni invocabili in un determinato linguaggio attraverso la subroutine system call interface legge l'ID, genera trap, gestisce e torna in user mode.

API: Application Programm Interface sono:
- sono standardizzate
- sono stabili rispetto alle versioni del so. (chiamate di sistema)
- offrono servizi più ad alto livello e semplici

Catena programmativa: Un programma sorgente viene compilato in un file oggetto che deve poter essere caricato a partire da qualsiasi locazione di memoria fisica. Il linker combinano più file oggetto per formare un file eseguibile, anch'esso rilocabile.
Il loader carica in memoria i file eseguibili nel momento in cui devono essere eseguiti e effettuano la rilocazione e il linking delle librerie dinamiche

Sistemi Operativi: Struttura e Servizi

Librerie dinamiche: Non vengono linkate a tempo di compilazione ma quando il programma è caricato:
- possono essere condivise da più programmi
- possono essere modificate senza dover ricompilatore
- snellisce gli eseguibili

Modalità funzionamento: Permette al so di proteggersi e di proteggere i programmi in esecuzione:
- user mode la cpu non può accedere alla memoria del Kernel
- Kernel mode istruzioni privilegiate.

ABI: Application Binary Interface insieme di convenzioni attraverso le quali codice binario di una applicazione si interfaccia con le API.

Eseguibili binari: ci sono 3 modi per avere programmi portabili:
- scrivo in linguaggio portabile — eseguibile = sorgente
- scrivo in linguaggio con run-time portabile — eseguibile = bytecode
- scrivo in linguaggio con compilatore portabile e API standard — eseguibile = binario

Programmi di sistema: Permettono agli utenti di avere un ambiente più conveniente per l'esecuzione dei programmi, il loro sviluppo e la gestione delle risorse del sistema.

Kernel: É un programma di dimensioni elevate e complesso, deve operare velocemente e un bug può causare crash.

Sistemi Operativi: Strutture e Servizi

Organizzazione: Esistono diverse strutture per il Kernel:

- monolitico: singolo file binario statico: semplice e veloce quindi efficiente, ma complesso
- a strati: il livello n usa n-1: indipendenza tra stati, astrazione, ma overhead.
- microkernel: sposta quanti più servizi ok di fuori del Kernel, quindi dimensioni ridotte, estensibile, pochi bug ma overhead
- a moduli: componenti dinamicamente caricabili che parlano attraverso interfacce. Ibrido tra microkernel e a strati, minore overhead ma minore isolamento.
- ibridi: combinano diversi approcci.

Politica: Dice quando una certa operazione viene effettuata. Influiscono sulle caratteristiche del so

Meccanismo: Spiega come una certa operazione è effettuata. Stabili

Interprete dei comandi: Permette agli utenti di immettere in maniera testuale le istruzioni che il so deve eseguire (Shell). Possono essere built-in o programmi di sistema.

Sistemi Operativi: Processi e Thread

Processo: Entità astratta definita dal SO allo scopo di eseguire un programma. (≠ programma, entità passiva). L'obiettivo del SO è di mantenere occupate le CPU dando l'illusione che ogni programma abbia il suo processore dedicato.

Multiprogrammazione: Impedire ad un programma fermo di mantenere la CPU

Nello specifico, il SO mantiene in memoria i processi da eseguire:

- se la CPU non è impegnata, le viene assegnato un processo. Quando quest'ultimo non può evolvere, la CPU viene assegnata ad un altro processo.
- richiede che tutte le immagini dei processi siano in memoria. Lo swapping viene usato per spostare dentro e fuori le immagini dalla memoria virtuale.

Multitasking: far sì che un programma interattivo reagisca velocemente agli input.

Nello specifico fa sì che la CPU venga sottratta periodicamente tra i vari processi in esecuzione: così facendo i processi proseguono in maniera parallela, evitando monopolio della CPU.

Creazione processi: Solo un processo può crearne un altro, chiamato figlio, generando un albero dei processi.

Terminazione dei processi: Devono richiedere al SO di essere terminati. Un padre può aspettare o meno la terminazione di un figlio. Alcuni SO impediscono ai figli di terminare dopo il padre. Se un figlio termina ma il padre non lo sta aspettando, si dice zombie. Se un padre termina, il figlio è detto orfano.

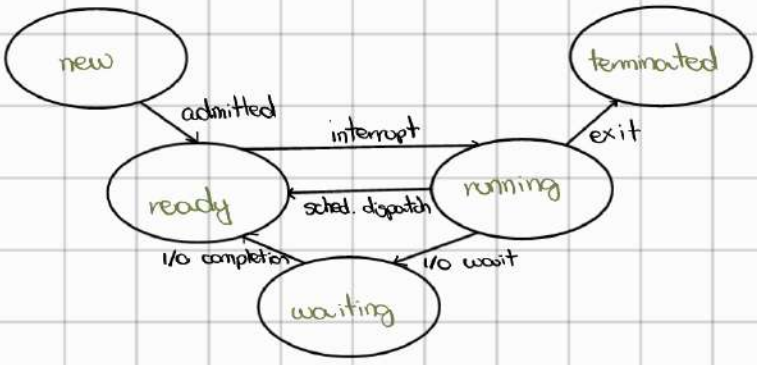
Struttura processo: Ogni processo ha una immagine diversa:

- stato registri processore
- stato memoria centrale (immagine): formata da text e data section da dim. costante e stack e heap da dimensioni variabili.
- stato del processo
- risorse del SO in uso.

Sistemi Operativi: Processi e Thread

Stati processo:

- new: creato ma non può eseguire
- ready: attesa assegnazione a CPU
- running
- waiting: in attesa di eventi
- terminated



PCB: Process Control Block, contiene tutte le informazioni relative ad un processo

Scheduling processo: Sceglie il processo da eseguire tra
- ready queue: processi in stato ready
- wait queues: code per processi in wait, divisi per il tipo di evento d'attesa.

Context switching: Effettuato da dispatcher, salva il processo da interrompere nel suo PCB e carica il nuovo contesto dal suo PCB.

Comunicazione interprocesso: I processi possono essere indipendenti o cooperare se il suo comportamento influenza o è influenzato da altri. Per cooperare sono necessarie le primitive di comunicazione interprocesso (IPC)

Pipe: canali di comunicazione tra processi. Possono essere uni/bi - direzionali, half- full - duplex

Multithreading: Più istruzioni possono essere eseguite contemporaneamente, quindi un processo può avere più percorsi (thread) di esecuzione concorrenti. I thread di un processo condividono la memoria globale, memoria del codice e le risorse ottenute. Ogni thread ha il proprio stack.

Sistemi Operativi: Processi e Thread

Implementazione thread: I thread possono essere implementati a due livelli:

- utente, ovvero quelli offerti dalle librerie dei thread
- Kernel, usati per strutturare il Kernel in maniera concorrente.

Modelli multithreading: ci sono diversi modelli utente-Kernel

- molti-a-uno: è usabile su ogni so ma una chiamata bloccante blocca tutti i thread, non sfrutta i core
- uno-a-uno: maggior concorrenza e sfrutta multicore, ma meno performante e stressa Kernel
- molti-a-molti: associazione tramite scheduler: unisce i due vantaggi ma difficile da implementare

TCB: Stesso scopo del PCB

Lightweight process: Interfaccia offerta da Kernel per le librerie per usare i thread.

Attivazione scheduler: Il Kernel comunica con le librerie tramite upcall. La libreria può rispondere aggiornando la propria struttura dati o effettuare operazioni di scheduling.

Sistemi Operativi: Scheduling CPU

Scheduler: Selezione tra i processi nella ready queue ed alloca un core libero ad esso.
Tali assegnamenti possono essere effettuati quando un processo cambia stato.

Riassegnamento: possono essere nonpreemptive, dove un core è sempre liberato quando un processo vi rinuncia, mentre le preemptive un core può essere liberato forzatamente.

Burst CPU: sequenza operazioni della cpu.
Burst I/O: attesa completamento operazione I/O.

FCFS: la CPU viene assegnata al primo processo. semplice ma tempo attesa medio lungo

SJF: viene assegnata al processo con CPU burst più breve.

RR: circolare, ogni processo ha una quantità di tempo + politica FIFO.

Priorità: ad ogni processo viene assegnato un valore, viene eseguito quello con priorità più alta.
Potrebbe verificarsi starvation, ovvero un processo con priorità bassa. Risolto con aging

Sistemi Operativi: Scheduling CPU

①

P	B
P_1	21
P_2	13
P_3	2
P_4	6
P_5	4

FCFS:

$\bar{t}$ attesa: $(0 + 21 + 34 + 36 + 42)/5 = 26,6$

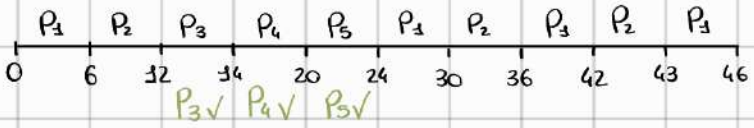
$\bar{t}$ completamento: $(21 + 34 + 36 + 42 + 46)/5 = 35,8$

SJF:

$\bar{t}$ attesa: $(25 + 12 + 0 + 6 + 2)/5 = 9$

$\bar{t}$ completamento: $(46 + 25 + 2 + 12 + 6)/5 = 18,2$

RR: quanto = 6



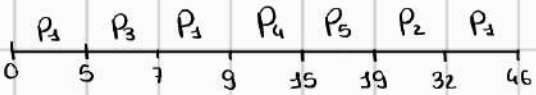
$\bar{t}$ attesa: P_1: $46 - 21 = 25$; P_2: $43 - 13 = 30$; P_3: $14 - 2 = 12$; P_4: $20 - 6 = 14$; P_5: $24 - 4 = 20$ → $/5 = 20,2$

$\bar{t}$ completamento: $(46 + 43 + 14 + 20 + 24)/5 = 29,4$

②

P	B	$\bar{t}$
P_1	21	0
P_2	13	10
P_3	2	5
P_4	6	9
P_5	4	12

SRTF:



$\bar{t}$ attesa: P_1: $46 - 21 = 25$; P_2: $32 - 23 = 9$; P_3: $7 - 7 = 0$; P_4: $15 - 15 = 0$; P_5: $19 - 16 = 3$ → $/5 = 7,4$

$\bar{t}$ completamento: P_1: $46 - 0 = 46$; P_2: $32 - 10 = 22$; P_3: $7 - 5 = 2$; P_4: $15 - 9 = 6$; P_5: $19 - 12 = 7$ → $/5 = 16,6$

Sistemi Operativi: Scheduling cpu

③

P	B	p
P_1	21	4
P_2	13	5
P_3	2	3
P_4	6	2
P_5	4	1

PR

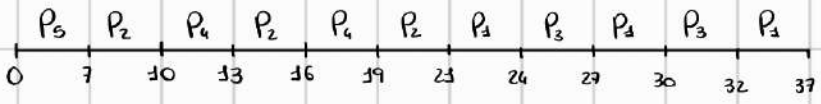
T. attesa: $(0+4+10+12+33)/5 = 11,8$

T. completamento: $(33+46+12+10+4)/5 = 21$

④

P	B	p
P_1	11	3
P_2	8	2
P_3	5	3
P_4	6	2
P_5	7	1

PR+RR = quanto=3



T. attesa: $P_1: 37-11=26$, $P_2: 23-8=13$, $P_3: 32-5=27$, $P_4: 19-6=13$, $P_5: 7-7=0$ $\Big\} /5 = 15,8$

T. completamento: $(37+23+32+19+7)/5 = 23,2$

Sistemi Operativi: Gestione Memoria

Allocazione contigua: Dobbiamo allocare memoria per i processi. Contigua è semplice, dove la mem. centrale è divisa in SO e processi utente, dove si caricano le immagini.

Associazione indirizzi: Il SO carica un programma in momenti diversi in diverse aree non contigue. L'associazione è detta binding e può non essere creata, con indirizzi rilocabili o assoluti. Il binding può essere fatto:

- compilazione se sappiamo l'indirizzo
- caricamento non sappiamo indirizzo e non viene spostato in memoria
- esecuzione: non sappiamo indirizzo e può essere spostato in memoria

Indirizzi logici e fisici: I logici sono generati dalla CPU mentre i fisici sono quelli della memoria centrale. I logici vengono trasformati in fisici tramite MMU: memory managment unit, mod utente.

Base e limite: Il processore ha due registri: base contiene + piccolo indirizzo fisico a cui può accedere, limite determina l'intervallo ammesso.

MMU con reg. di rilocazione: Il registro base è reg. di rilocazione e un indirizzo fisico diventa logico + rilocazione. Se il logico supera limite si genera eccezione.

Allocazione contigua a partizioni variabili: Ogni processo ottiene partizione di memoria attraverso un buco: possiamo inserire i processi con 3 strategie:

- first-fit sceglie il primo sufficiente
- best-fit sceglie il buco più piccolo
- worst-fit sceglie il buco più grande

Frammentazione: può essere:

- esterna memoria sufficiente ma sparsa tra buchi troppo piccoli.
- interna se la mem. allocata, una partizione contiene memoria inutilizzata

Sistemi Operativi: Gestione Memoria

Paginazione: La memoria fisica viene divisa in frames di dim. fissa, mentre la dinamica in pagine con dim. uguali a frame. La tabella delle pagine associa frame a pagine

Traduzione degli indirizzi: Un indirizzo logica è divisa in:

- n° pagina: indice nella tabella delle pagine
- offset: all'interno della pagina.
- con 2^m indirizzi logici e page grandi 2^n ⟶ n° di pagine 2^{m-n}

Supporto SO: Deve mantenere page table di ciascun processo e tabella dei frame di ogni frame. La MMU utilizza due registri:

- page table base register: indirizzo fisico dell'inizio della tabella
- page table length register: dimensione delle tabella delle pagine.

TLB: Translation Lookaside Buffer: cache apposita per evitare i due accessi di PTBR e PTLR. Bisogna flushare ogni context switch. Si possono mantenere le entry con un address space identifier.

Tempi effettivi accesso memoria:

- TLB hit: n° pagina trovato
- TLB miss: n° pagina non trovato
- Effective Access Time: $ma \cdot hr + 2ma(1-hr) = ma(2-hr)$, dove ma = memory access, hr = hit-ratio

Politiche di Sostituzione: se vi è una TLB miss ma TLB pieno, occorre sostituire una entry con una strategia:

- LRU: last recently used
- RR: round robin
- Random

ci sono vincolare TLB entry da non sostituire mai.

Sistemi Operativi: Gestione Memoria

Protezione della memoria: Il PTLR permette di tagliare la tabella in maniera da ridurre indirizzi o si può usare validity bit per creare indirizzi logici inaccessibili.

Pagine condivise: affinché possano condividere memoria, due frame devono essere nella stessa page table.

Struttura della tabella delle pagine: esistono varie soluzioni per strutturarle in maniera ottimale:

- gerarchiche: evitano allocamento a mem. contigua. + livelli, ogni pagina ha nº e offset. Ha supporto a pagine di dimensioni diverse ma aumento numero accessi.
- hash con gestione delle collisioni
- invertite un entry per ogni frame anziché per pagina. Abbiamo solo una tabella per processo ma maggior tempo d'accesso.

Swapping: Tecnica che permette di eseguire più processi di quanti la memoria fisica ne riesca a contenere.

Swapping standard: molto oneroso perché si sposta l'intero processo al backing store.

Swapping con paginazione: sposta solo sottoinsieme delle pagine di un processo che permette di fare una nuova entrata.

Paginazione su richiesta: Demand paging. implementa memoria virtuale. avere permette a un processo di eseguire anche se una parte di esso è in memoria fisica.

Page fault: Interrupt se si accede a pagina con validity bit=0

Swap Space: Memorizza le pagine fuori dalla memoria nel backing store.

Sistemi Operativi: Gestione Memoria

Gestione Page Fault: Il so decide se ciò che genera errore è non valido o non in memoria:
- non valido: abort
- non memoria: carica da backing store: trova frame libero, scheduling, context switch, aggiorno pt.

Politiche caricamento pagine:
- pura: caricate se servono
- prefatching: caricate in un intorno di pagine.

Grado di multiprogrammazione: n° max di processi che lo scheduler può mandare in esecuzione.
Determina quindi l'occupazione totale di memoria fisica da parte dei processi. |multiprog|=|imm. in mem.|
I processi in memoria sono ready + running + waiting.

Controllo grado: Scheduler a lungo termine decide se ammettere un processo dopo la sua creazione.
a medio termine si occupa dello swapping, a breve termine non influisce sul grado.

Sostituzione pagina: quando si libera un frame inutilizzato, si aggiorna la pt, ma non si fa page out dei read-only e con bit di modifica a 0.

Algoritmi di allocazione: Determina quanti frame assegnare ad ogni processo. Può essere:
- fissa: n° frame non cambia durante esecuzione.
- variabile: n° frame può cambiare durante esecuzione.
- uniforme: m frame e n processi, si assegna m/n frame con m%n come buffer.
- proporzionale: con m frame al processo i con s_i di mem. virtuale, ottiene frame $a_i = s_i / \sum_{i=1}^{n} s_i \cdot n$

Algoritmi sostituzione: determina qual'è frame vittima, confronto tramite page fault:
- globale: tra tutti i frame del processo. Efficiente ma perdi pagine a causa di altri processi
- locale: solo nel frame che genera page fault. Stesso n° pagine per processo ma sottoutilizzo memoria.

Sistemi Operativi: Gestione Memoria

Thrashing: Quando un processo trascorre più tempo nella gestione dei page fault che nell'esecuzione.
Avviene quando $\sum_{i=1}^{n} \text{località}_i > \text{memory}_i$.

Rimedi: si abbassa la multiprogrammazione con due tecniche: working set e frequenza page fault.

Working set: si calcola località del processo, Δ = finestra working set rispetto al n° di riferimenti:

- $\Delta \ll$ w.s. non include località interamente
- $\Delta \gg$ sovrappone più località
- $\Delta \to \infty$ w.s. include tutte le pagine usate dal processo.

Frequenza page fault: si stabilisce un range accettabile nella freq. dei page fault:

- PFF < soglia minima → processo perde frame
- PFF > soglia massima → processo acquista frame che, se non disponibili, vanno liberati.

Sistemi Operativi: File System

File system: come il SO memorizza dati e programmi.

File: Unità di memorizzazione logica. I suoi attributi sono memorizzati nella directory.

Tabella file aperti: Il SO mantiene una tabella di file aperti per ogni processo.

Lock dei file: associa ai dei file dei lock per coordinare più processi:
- condiviso più processi possono acquisirlo o esclusivo, solo uno.
- mandatory il SO regola l'accesso al file o advisory, no regolamento.

Metodi di accesso: può essere sequenziale, dove si fa I/O il record successivo alla pos. corrente, diretto dove si accede a n-esimo record, indicizzato basato su chiave.

Directory tabella di simboli che traduce il nome del file negli elementi contenuti.

Struttura: può essere:
- ad un livello: una sola directory per tutti i file
- 2 livelli: una dir contiene una dir per utente
- ad albero: ogni dir contiene altre dir. Si possono specificare path relative.
- a grafo aciclico: si può assegnare più di un nome allo stesso file
- a grafo generico: si introduce garbage collector.

Liste controllo accessi: ad ogni file/directory associato ACL dove specifica chi può accedere.
Ci sono 3 tipi di utente: proprietario, gruppo o pubblico

Sistemi Operativi: File System

Strutture dati: servono per realizzare delle operazioni. Open apre o crea file, close elimina.

Realizzazione delle directory: può essere
- lista lineare con nomi di file e puntatori ai blocchi dei dati. Onerosa.
- tabella hash con funzione hash per trovare indice da nome. Collisioni

Allocazione contigua: ad ogni file è associato un insieme contiguo di blocchi. Semplice e performante, ma bisogna sapere dim. file, trovare spazio contiguo e frammentazione.

Allocazione concatenata: ogni file è composto da una lista concatenata di blocchi.

Allocazione indicizzata: ogni file ha blocco indice come un array.