

STRUTTURA DEI SISTEMI OPERATIVI

- 3 TIPOLOGIE PRINCIPALI DI SISTEMI OPERATIVI:

- ① SISTEMI OPERATIVI BATCH: APPLICAZIONI CPU BOUND
 - ② SISTEMI OPERATIVI TIME-SHARING: APPLICAZIONI I/O BOUND
 - ③ SISTEMI OPERATIVI REALTIME: RISPECTO DEI VINCOLI TEMPORALI (DEADLINE)
- PRINCIPALI COMPONENTI DI UN SISTEMA OPERATIVO

① GESTORE DEI PROCESSI (SCHEDULER)

RIPARTISCE L'USO DEL PROCESSORE TRA I VARI PROGRAMMI CARICATI IN MEMORIA E LI ESEGUE CONCURRENTEMENTE

② GESTORE DELLA MEMORIA PRINCIPALE

NECESSITA' DI MANTENERE IN RAM CONTEMPORANEAMENTE PIU' PROGRAMMI. PARTE DELLA MEMORIA DEVE ESSERE INFILATA AL S.O., QUESTA E' SUDDIVISA IN 2 PARTI: "PARTE RESIDENTE" CON IL BOOTSTRAP E IL SOTTOSISTEMA DELLE FUNZIONI PIU' UTILIZZATE. LA 2^o PARTE VIENE UTILIZZATA PER CARICARCI LE ALTRE FUNZIONI DALLA MEMORIA DI MASSA.

③ GESTIONE DEI DISPOSITIVI PERIFERICI

SI OCUPA DELLA COORDINAZIONE DEGLI ACCESI DEI PROCESSI ALLE PERIFERICHE IN modo DA EVITARE INTERFERENZE E NASCENDERE LA COMPLICATEZZA E LA DIVERSITA' DEI VARI DISPOSITIVI HARDWARE

④ GESTIONE DEGLI ARCHIVI (FILE SYSTEM)

SI OCUPA DI ~~SPOLARIZZARE~~ SEMPLIFICARE I PROBLEMI DI NOMEZIAZIONE SU MEMORIA DI MASSA, DEI CRITERI DI INDIVIDUAZIONE DEI FILE ALL'INTERNO DEL SISTEMA, GARANTIRE PRIVACY E PROTEZIONE

⑤ INTERPRETE DEL LINGUAGGIO DI CONTROLLO

LA SHELL

- PRINCIPALI MODELLI STEVINERI

I PRIMI SISTEMI (BATCH) ERANO COSTRUITI A UN UNICO PROGRAMMA (SISTEMI MONOUNICA). DOPO VENnero CREATI I SISTEMI MODULARI, CHE SONO DIVISI IN MODULI DISPARSI.

IL SISTEMA VIENE ORGANIZZATO GIGARICCIAMENTE IN DIVERSI LIVELLI DI ASTRATTOZIONE IN CUI I PRODOTTI DI UN LIVELLO UTILIZZANO ESCLUSIVAMENTE LE FUNZIONALITÀ OFFERTE DA LIVELLO DI LIVELLO PIÙ BASSO E FORNISCONO LE LORO FUNZIONALITÀ AI LIVELLI DI LIVELLO PIÙ ALTI.

STRUCTURE A MICROKERNEL: L'INSIEME DEI MECCANISMI COSTITUISCE IL MICROKERNEL DEL SISTEMA, CHE È L'UNICO COMPONENTE AD AGIRE IN STATO PRIVILEGIATO, TUTTE LE STRATEGIE VICEVERSA FANNO PARTE DEI PROGETTI DI SISTEMA CHE GIRANO COME NORMARI PROCESSI.

CLASSIFICARE LE ARCHITETTURE: TASSONOMIA DI FLYNN

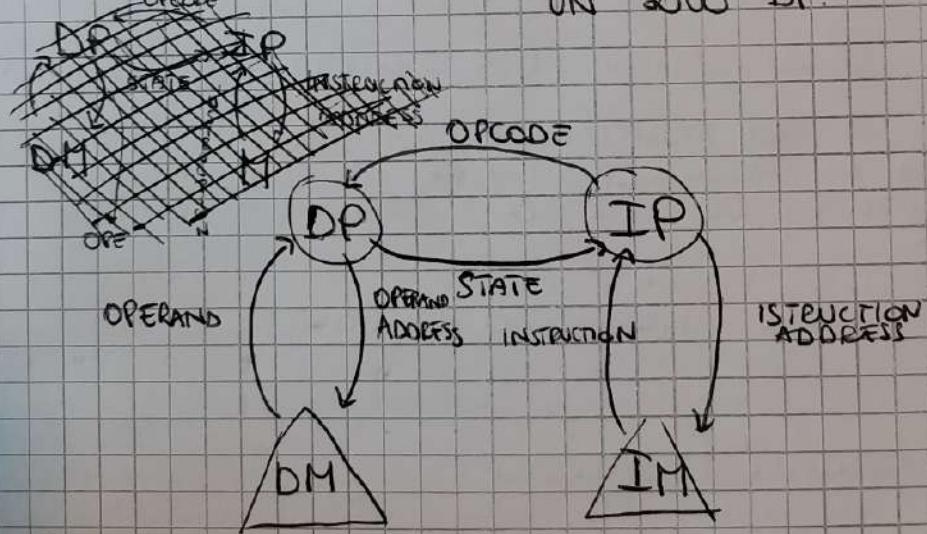
CLASSIFICA IL SISTEMA DI ELABORAZIONE DA 2 PUNTI DI VISTA:

- 1 - LA CAPACITÀ DI AVERE PIÙ MUSSI DI ISTRUZIONI
- 2 - LA CAPACITÀ DI AVERE PIÙ MUSSI DI DATI

SINGLE INSTRUCTION	MULTIPLE INSTRUCTIONS
SIMD	MIMD
MD MULTIPLE DATA STREAM	
SD SINGLE DATA STREAM	

- MACCHINE [SIMD]

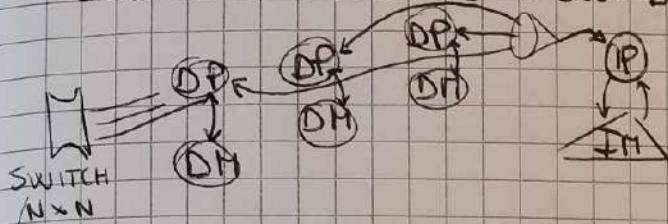
TRADIZIONALE MACCHINA SEQUENZIALE USATA DA NUMERI DI CALCOLATORI CONVENTIONALI. UN'UNICA ISTRUZIONE È ESEGUITA A OGNI STEP TEMPORALE. C'È UN SOLO IP E UN SOLO DP.



DIVISO LA MEMORIA IN 2: DATA MEMORY E INSTRUCTION MEMORY.

- MACCHINA SIMD

PIU' UNITA' DI ELABORAZIONE ESEGUVONO CONTEMPORANEAMENTE LA STESSA ISTRUZIONE SU FLUSSI DI DATI DIVERSI.



CONTEMPORANEAEMENTE

LA STESSA ISTRUZIONE ESEGUITA SU DATI DIVERSI.

LA STESSA ISTRUZIONE VIENE ESEGUITA SU DATI DIVERSI.
I DATI SONO DISTRIBUITI SULLE VARIETÀ DM

C'E UN SOLO PROGRAMMA IN ESECUZIONE, SONO I DATI AD ESSERE PARALLELLIZZATI. ESEGUITO IL PROBLEMA SOLO UNA VOLTA

LE MEMORIE A COSE NORMALE SONO PRIVATE. E' UN EVENTUALE TRAVERSAMENTO CHE PERMETTE DI CONEGGIERE UN'ACCERCHIATA.

- C'E REGOLARITA' NELLA COMUNICAZIONE: IP DANDO L'ISTRUZIONE, TUTTI I DP LA ESEGUVONO E Poi RESTITUISCONO INSIEME LO STATO.

ES - SUPERCOMPUTER VETTORIALI

QUALE' IL SISTEMA OPERANO DI UNA MACCHINA SIMD? NON CI SONO NUOVI PROBLEMI SUA GESTIONE DATI IN ENTRA.

NEMENO PER LA GESTIONE DEI PROCESSI.

LE PERIFERICHE DOVONO ESSERE ATTACcate ALLE DN.

- MACCHINE MISD

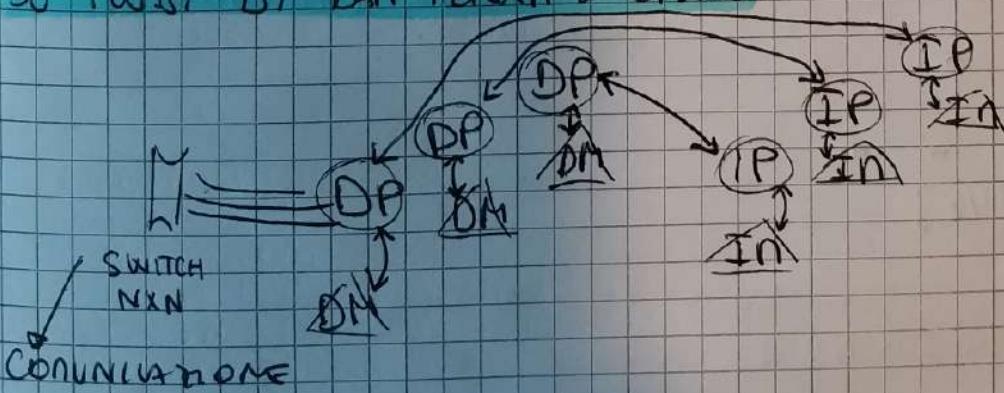
PIU' FLUSSI DI ISTRUZIONI ESEGUITI CONTEMPORANEAMENTE SU UN UNICO FLUSSO DI DATI. ESISTONO? NO.

SE VEDO IL PROCESSORE COME PIPELINE, ALLORA AVRE' UNA PARTE CTF DECODIFICA L'ISTRUZIONE, UNA CHE LA ESEGUE...
NON IMPORTANTE.

A LIVELLO DI COMPUTAZIONE
ACCORGE AUTONOMAMENTE DI POTER OMAGGIARE IL CODICE IL SUPERCOMPUTER SI

- MACCHINE MIMO

PIU' UNITA' DI ELABORAZIONE, PIU' ISTRUZIONI ESEGUITI CONTEMPORANEOAMENTE SU FLUSSI DI DATI PERIAMI E' DIVERSI.



- IN PRatica OGNI COPPIA IP DP COSTITUISCE UNA MACCHINA SIMD.

TRA I NODI NON ESISTE MEMORIA CONDIVISA.

[DM-MIMO]

ES - MULTICOMPUTER, RETI DI CALCOLATORI
 (INTERNET CON UN CERTO GRADO DI ATTENZIONE PUÒ EFFETTUARE
 SCAMBIO DI INFO TRAMITE MESSAGE PASSING. ELEVATA SICUREZZA.
 IL PROBLEMA È LA RETE, CERCAVI PC AI PERMETTE DI CONNETTERE?
HIGH AVAILABILITY E LOAD BALANCING)

IN CASO DI GUARDA LA
 COMPUTAZIONE MIGRA DA
 UN NODO ALL'ALTRO.

TASK MUOVONO NEI NODI CHE
 HANNO IL MINOR LATICO

[SM-MIMO]

LE DUE POSSONO ESSERE ATTACcate INVECE CHE ASSOCiate AL
 UNO SWITCH, IN QUESTO CHE TUTTE LE DUE SONO CONDIVISE,
 SI CHIAMANO MULTIPROCESSORI

TIPOLOGIE DI INTERCONNESSIONE

- BUS

GRADO: 1 (PER CONNECARE UN SISTEMA A UN BUS BASTA GRADO 1)

DIAMETRO: 1 (DISTANZA MASSIMA TRA UNA QUALSIASI COPPIA DI NODI
 INTERMINI DI LINK DA ATTIVARE)

#LINK: 1

FORTE COMPETIZIONE PER ACCEDERE AL LINK



GRADO: NUMERO
 DI LINK CHE
 DEVO CREARE PER
 ATTACCARE UN
 NUOVO NODO

- LINEAR ARRAY



GRADO: PER IL 1° È UNSO NODO E' 1, PER GLI ALTRI 2

DIAMETRO: N-1

COMPETIZIONE MINIMA (I NODI DEVONO ESSERE CAPACI DI FARE
 ROUTING)

NO TOLERANZA AI GUASTI

COMUNICAZIONI IN CONTEMPORANEA: $N/2$ (100% MAX)

- RING



GRADO: 2

DIAMETRO: $\lceil N/2 \rceil$

#LINK: N

TOLERANZA AI GUASTI: 1

- CONNESSIONE COMPLETA

GRADO: $N-1$ (PER OGNI 1 NODI)

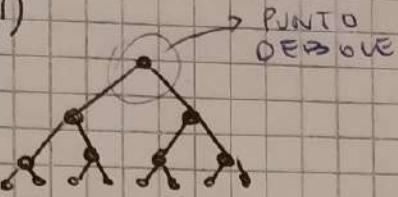


DIAMETRO: 1

TOT DI LINK: $N^* \left(\frac{N-1}{2}\right)$

- B-TREE

ALTEZZA: $\log_2 N$



GRADO: RADICE 2, FOGLIE 1, ALTRI NODI 2

DIAMETRO: $2^* (h-1)$

LINK: $N-1$

LA TOLERANZA AI GUASTI E' MOLTO VACUALE A SECONDA DI CUI SI GUASTA.

INOLTRE LA COMUNICAZIONE E' PIU' INTESA NELLA MODO CHE NI AVVICINO TUTTA RADICE.

- STAR

GRADO: $N-1$ PER IL NODO

CENTRALE, 1 PER OGNI ALTRI



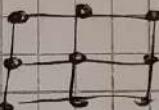
DIAMETRO: 2

LINK: $N-1$

TOLERANZA AI GUASTI DIPENDE DAL NODO CENTRALE

- MESH

GRADO: VERGICI 2 "CENTRALI"
NELL'UN 3, RESTANTI 4



DIAMETRO: $2^* (\sqrt{N} - 1)$

LINK: $2^* N - 2^* \sqrt{N}$

BUONA RESISTENZA AI GUASTI.



GRADO: 4 PER

OGNI 1 NODI

DIAMETRO: $2^* \lfloor \frac{\sqrt{N}}{2} \rfloor$

$2^* N$

NOTEVOLMENTE RESISTENTE
AI GUASTI.

- IPERUBBO (INDISEGNABILE)

COME MISURARE L'EFFICIENZA?

$$\text{SPEED-UP } S = \frac{T_4}{T_N} \rightarrow$$

ESECUZIONE CON UN SOLO
PROCESSORE

→ SPEEDUP $S > 1$

ESECUZIONE CON
NULIPROCESSORE

$$\text{EFFICIENZA } E = \frac{S}{N}$$

$$\text{IDEALE } E=1$$

$$\text{REALTA' } E < 1$$

$$\text{IDEALE } S=N$$

$$\text{IN REALTA' } S < N$$

(NUOVE SINO SPESO $S=N$)
HANNO POCO I/O

LEGGE DI AMDAHL

UN PARALLELISMO PERFETTO NON E' MAI RAGGIUNGIBILE PERCHE'
SARANNO SEMPRE PRESENTI PARTI DI SOFTWARE PER FORZA
SERIALE.

$$\text{LIMITE SUPERIORE: } \frac{T_4}{T_{\text{SER}}}$$

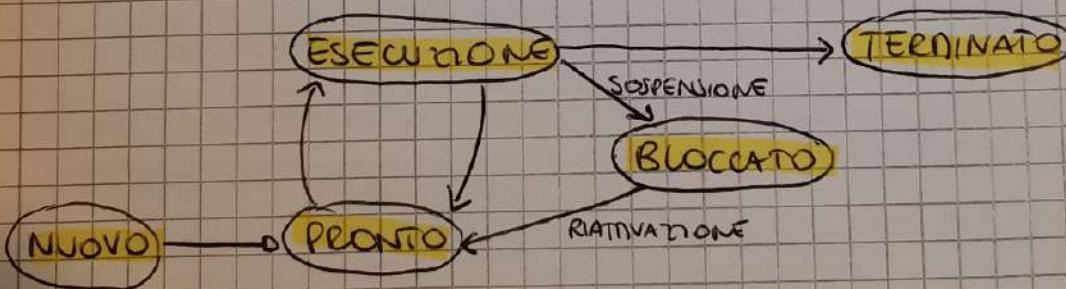
GESTIONE DEI PROCESSI

RICHIAMI:

- **PROCESSO:** E' LA SEQUENZA DI EVENTI CUI DA WORD UN ELABORATORE
QUANDO OPERA SOTTO IL CONTROLLO DI UN PARTICOLARE
PROGRAMMA.

UN PROCESSO E' UN'ISTANZA DI UN PROGRAMMA IN ESECUZIONE.

- STATI DI UN PROCESSO:



AD OCCHI
ESTERNI E'
COME SE OGNI
PROCESSO
POSSEDESSE
UNA SUA CPU
VIRTUALE

- DESCRITTORE DI PROCESSO:

- NAME DEL PROCESSO (PREZZO IN DENDA TABEWA)
- STATO DEL PROCESSO
- MODALITA' DI SERVIZIO DEI PROCESSI (AD ESEMPIO UNA PRIORITA')
DIPENDE DAL CRITERIO DI UNO SCHEDULING

- INFORMAZIONI SUA GESTIONE DELLA MEMORIA
- CONTESTO DEL PROCESSO: TUTTE LE INFO CONTENUTE NEI REGISTRI
- UTILIZZO DELLE RISORSE
- PUNTATORE AL PROCESSO SUCCESSIVO.

- CANGIO DI CONTESTO

1. SALVATAGGIO DEL CONTESTO DEL PROCESSORE IN ESECUZIONE NEL SUO DESCRITTORE (SUO - STATO)
2. INSERIMENTO DEL DESCRITTORE IN CODA PRONTI / BLOCCATI
3. SELEZIONE DI UN NUOVO PROCESSO TRA THOSE IN CODA PRONTI (SCHEDULING) E CARICA NELLO STATO DI TALE PROCESSO NEL REGISTRO "PROCESSO IN ESECUZIONE"
4. CARICA NEL NUOVO CONTESTO NEL REGISTRO DEL PROCESSORE (CARICA - STATO)

IL CAMBIO DI CONTESTO COMPORTA L'AGGIORNAMENTO DI VIELE STRUTTURE DATI: MEMORIA, DISPOSITIVI I/O, FILE APERTI...
 => PERDITA DI TEMPO E ENERGIA.

- INTERAZIONE TRA I PROCESSI (SI PARLA QUINDI DI PROCESSI CONCERNENTI)

OVERLAPPING: OGNI PROCESSO POSSIEDE UNA SUA UNITÀ DI ELABORAZIONE

INTERLEAVING: I PROCESSI CONDIVIDONO LA STEMA UNITÀ DI ELABORAZIONE => QUINDI C'È CONCORRENZA

PROCESSI
INDIPENDENTI



IL SUO COMPORTAMENTO
E' RIPRODUCIBILE, ANCHE
SE ESEGUITO IN
MONADA DIVERSI

PROCESSI
INTERAGENTI



SI INFLUENZANO VICENDOVESENTE
DURANTE L'ESECUZIONE
PER COOPERAZIONE O
CONCORRENZA.
COMPORTAMENTO NON
RIPRODUCIBILE

COOPERAZIONE: P_1 (PRODUTTORE) PRODUCE CICLICAMENTE UN MESSAGGIO CHE DEVE ESSERE LETTO DA P_2 (CONSUMATORE)
IN UN SINGOLO BUFFER CONDIVISO.
C'È SOLO UNA SECONDA VALUTA: INS - PREMUTO - INS - PRE...

E' IMPORTANTE L'ORDINE

COOPERAZIONE E CONCERNTE

CONCERNTE: P_1 E P_2 CONDIVIDONO LA VARIABILE
CONTATORE E LA INCREMENTANO OGNI TOT.

UN SOLO PROCESSO MIA VOLTA DEVE AVERE ACCESO ALLA
RISORSA COMUNE! E' IMPORTANTE LA MUTUA ESCLUSIONE

- ESECUZIONE DI UNA PRIMITIVA DEL NUCLEO

A OGNI PROCESSO E' ASSOCIASTA UNA PIA GESTITA TRAMITE I REGISTRI
SP. SUPPONIAMO INOLTRE CHE VI SIANO 2 INSIEMI DI
REGISTRI R_1, \dots, R_n E R'_1, \dots, R'_n , SP E SP' ASSOCIANO RISPECTIVAMENTE
A PROCESSI E NUCLEO.

L'ESECUZIONE DI UNA PRIMITIVA DA PARTE DI UN PROCESSO P
CONSISTE NELL'ESECUZIONE DI UN'ISTRUZIONE JVC (SUPERVISORICA)
CHE GENERA UN'INTERVENTO DI TIPO SINCRONO CON PRIMATAGGIO
DI PC E PS RITORNANDO A P IN CIMA ALLA PIA DEL NUCLEO
E CARICANDO IN PC E PS RISPECTIVAMENTE DELL'INDIRIZZO
DELLA PROCEDURA E DI UNA PAROLA DI STATO PROPIA DELLA OBBIETTIVITÀ
NUCLEO.

SE LA FUNZIONE RICHIESTA NON E' BLOCCANTE PER IL PROCESSO,
NEANTERTE RET IL NUCLEO RIPRENDENDO I REGISTRI PC E PS' (PS CONTENE
CON I VALORI CHE ERANO STASI NESSI IN PIU'.
ALTRIMENTI CARICA STATO + RET

PC CONTAG
L'INDIRIZZO
DELL'PROX
ISTRUZIONE DA
ESEGUIRE.

- SCHEDULING

- **SHORT TERM SCHEDULER:** E' LA PARTE DI NUCLEO CHE
SI OCCUPA DI SCEGLIERE QUALE PROCESSO DOVRÀ COME PRONTO
ANDRÀ IN ESECUZIONE

- **LONG TERM SCHEDULER:** E' UN SCHEDULER CHE SI OCCUPA
DI SCEGLIERE TRA NM I PROGRAMMI CARICATI IN MEMORIA
DI MASSA CELENI DA TRASFERIRE IN MEMORIA CENTRALE
CONTROVALA ANCHE IL NUCLEO DI PROCESSI GIÀ CREATO E
NON ANCORA TERMINATI

- **MEDIUM TERM SCHEDULER:** SI OCCUPA DI SWIPE OUT
E SWIPE IN DA MEMORIA CENTRALE A MEMORIA DI
MASSA E VICEVERSA.

PREEMPTIVE

POSSENGO REVOLARE LA
CPU AL PROCESSO IN
ESECUZIONE AD ESEMPIO
A UNA DI UN
INTERVENTO

NON PREEMPTIVE

NON POSSONO REVOLARE LA
CPU AL PROCESSO UNA
VOLTA PARON. DECIDONO SOLO
IL PROX PROCESSO DA MANDARNE IN
ESECUZIONE DOPO LA TERMINAZIONE
SOSPENSIONE DEL PRIMO

PASSAGGIO TRA I 2 AMBIENTI (NUCLEO E PROCESSO)

- ① INTERRUZIONI ESTERNE O ASINCRONE
- ② CHIAMATE DI SISTEMA (INTERRUZIONI INTERNE O SINCRONE) \Rightarrow Int

- CRITERI PER LA VALUTAZIONE DI UNO SCHEDULER

SISTEMI BATCH

① UTILIZZO CPU

② TURNAROUND TIME (VALORE MEDIO DEL TEMPO TRA QUANDO IL PROCESSO ENTRA PER LA PRIMA VOLTA IN CODA PRONTA E QUANDO TERMINA)

③ THROUGHPUT RATE:
NUERO MEDIO DI
PROCESSI COMPLETATI
NELL'UNITÀ DI TEMPO.

SISTEMI TIME-SHARING

②

TEMPO DI ATTESA:

TEMPO TOT CHE IL PROCESSO ATTENDE IN CODA PRONTA PER DI ENTRARE IN ESECUZIONE

① TEMPO DI RISPOSTA

(INTERVALLO TRA L'ISTANTE IN CUI P ENTRA IN CODA PRONTA E L'ISTANTE IN CUI RICEVE LA PRIMA RISPOSTA)

SISTEMI REAL-TIME

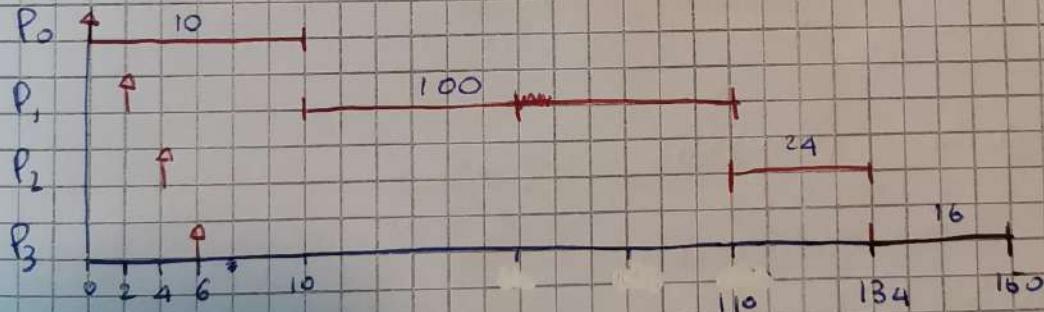
OPPURE

MINIMIZZARE IL NUM. DI PROCESSI CHE NON RISPETTANO IL DEADLINE (SOFT)

GARANTIRE CHE TUTTI I PROCESSI RISPETTINO IL DEADLINE. (HARD)

① ALGORITMO FCFS (FIRST COME FIRST SERVED)

- NO PREEMPTION



TURNAROUND:

$$T_0 = 10 - 0 = 10 \quad T_1 = 110 - 2 = 108 \quad T_2 = 134 - 4 = 130 \quad T_3 = 150 - 6 = 144$$

$$T = 98 \quad (\text{MEDIUM AVERAGE})$$

TEMPO DI ATTESA A

$$A_0 = 0 \quad A_1 = 10 - 2 = 8 \quad A_2 = 110 - 4 = 106 \quad A_3 = 134 - 6 = 128$$

$$A = 60.5$$

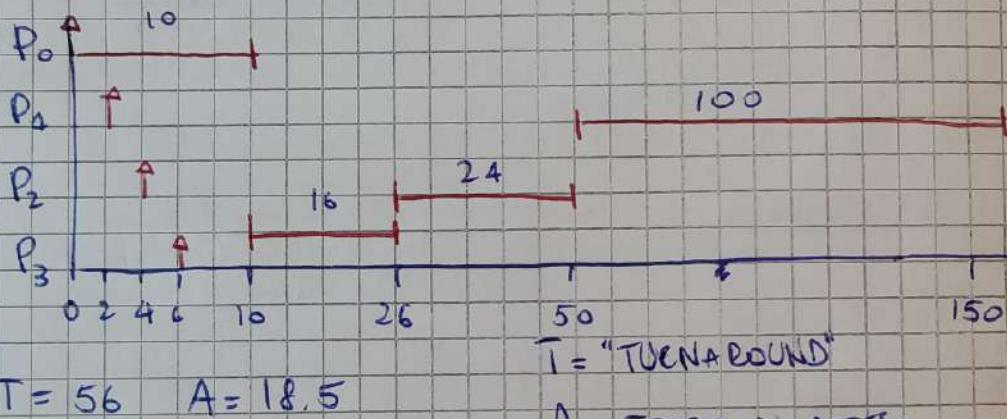
NOTARE CHE T E A SONO ALEATORI: AVENDI ESEGUITO P_0 PER UNSO ARCA OTTENNO VALORI SIGNIFICATIVAMENTE DIPENDENTI (ATTUALMENTE) DAL ORDINE CON CUI I PROCESSI ARRIVANO IN CODA PRONTI.

② SJF (SHORTEST JOB FIRST)

L'ALGORITMO ASSEGNA LA CPU AL PROCESSO CHE RICHIERA IL MINOR TEMPO DI ESECUZIONE FRA QUELLI IN CODA PRONTI

2 PROBLEMI:

- LA CONOSCENZA DEI TEMPI DI ESECUZIONE DI UN PROCESSO A PRIORI E' IMPRESCIA
- SCHEDULAZIONE: SE CONDANNANO AD ESEGUIRE PROCESSI BREVI RISCHIO DI NON ESEGUIRE NEI QUELLI LUNghi



STIMA DEL CPU BURST

NON SEMPRE IL VALORE DEL CPU BURST E' NOTO A PRIORI. METODO DELLA "MEDIA ESPONENZIALE" PERMETTE DI TENERE CONTO DELLA STORIA DEI VALORI MISURATI NEI PRECEDENTI INTERVALLI DI ESECUZIONE

$$S_{n+1} = \alpha t_n + (1-\alpha) S_n$$

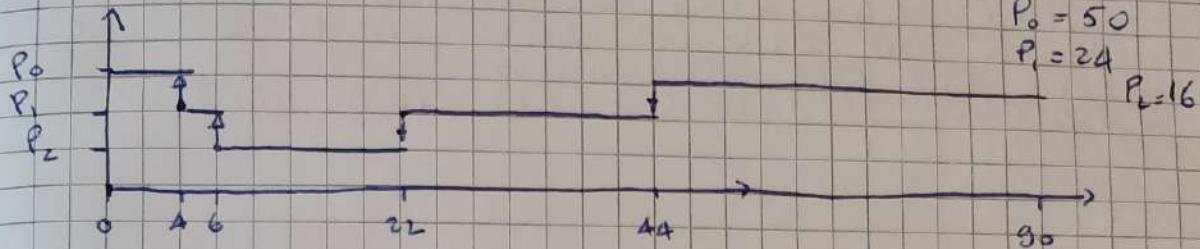
\downarrow E' il valore della stima

FATTORE DURATA DEL CPUBURST
 $0 < \alpha < 1$ n-ESIMO

- $\alpha = 0 \rightarrow$ TUTTE STIME UGUALI A QUELLA INIZIALE, NON CONSIDERANO t_n IN PRATICA
- $\alpha = 1 \rightarrow$ NON SI TIENE CONTO DELLA STORIA DELLE STIME
- TIPICAMENTE $\alpha = \frac{1}{2}$

③ SRTF (SHORTEST REMAINING TIME FIRST)

E' LA VERSIONE PREFERIBILE DI SJF. SE DURANTE L'ESECUZIONE DEL PROCESSO P_0 ENTRA IN CODA PRONTI UN PROCESSO P_1 PIU' BREVE, ESSO VA IN ESECUZIONE.



TURNAROUND

$$\frac{30 + (44-4) + (22-6)}{3} = 48.6$$

T_A

$$\frac{(44-4) + (22-6)}{3} = 18.6$$

- I VALORI NON TENGONO CONTO DEL MAGGIORE OVERHEAD

④ ROUND ROBIN

(ci sono più cambi di processo)

CONCEPITO PER SISTEMI A PARTIZIONE DI TEMPO. OGNI PROCESSO ENTRA IN ESECUZIONE PER QUALCINE DECINA DI MILISECONDI A ROTAZIONE.

E' OPPORTUNO, PER NON AUMENTARE ECCESSIVAMENTE L'OVERHEAD, CHE IL TEMPO PER LA COMPUTATION DEL CONTESTO sia << DEL TEMPO IN CUI LA CPU VIENE ALLOCATA.

- NON C'E STARVATION
- SE AUMENTA RUOLO IL NUMERO DI PROCESSI, ROUND ROBIN ≈ FCFS
- IL TEMPO DI RISPOSTA ALCIORE TANTO PIU' ABBASSO IL QUANTO DI TEMPO, MA AUMENTA L'OVERHEAD
- FAVORISCE I PROCESSI I/O BOUND

⑤ SCHEDULAZIONE A BASE PRIORITÀ CON CODE MULTIPLE

LA PRIORITÀ DI UN PROCESSO E' TANTO PIU' ALTA QUANTO MINORE E' IL VALORE NUMERICO CHE LA CARATTERIZZA PROBLEMA PIU' CRITICO: STARVATION. UN PROCESSO A BASSA PRIORITÀ RISCHIA DI NON ESSERE MAI ESEGUITO.

2 CODE:

LA CODA DEI PROCESSI PRONTI VIENE SCOMPAGNA IN 2 DIVERSE CODE:

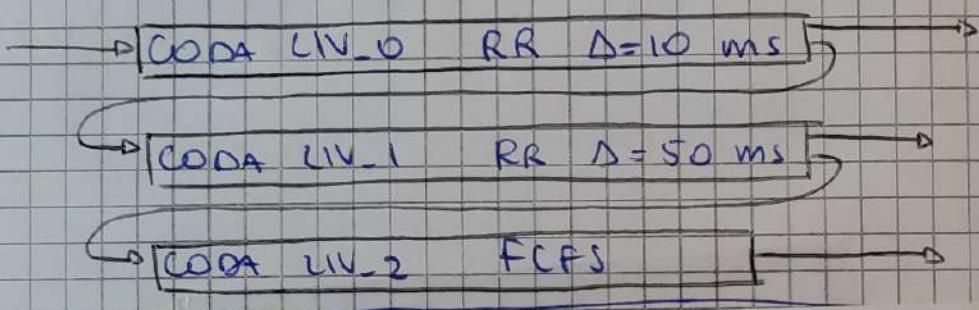
LIV-0: PROCESSI I/O BOUND. TECNICA: ROUND-ROBIN
LIV-1: PROCESSI CPU BOUND. TECNICA: FCFS

PROBLEMI:

- STARVATION SE CONTINUANO AD ARRIVARE PROCESSI CIVICO
 - SE UN PROCESSO UV-1 PRENDE IL PROCESO MAGGIOR
LO USA PER TANTO TEMPO.
- SOLUZIONE: OGNI VOLTA CHE UN NUOVO PROCESSO ENTRA IN CODA UV-0 DEVO AVERE UN'INTERRUZIONE INTERNA CHE RIFA' SCHEDULING.
IL PROCESSO UV-1 DEVE ESSERE PESO IN TESTA ALLA CODA UV-0

3 CODE:

UN PROCESSO AUTOMATICAMENTE SI COLLOCÀ NELLA CODA PIÙ ADEGUATA ALLE SUE CARATTERISTICHE DI INTERAMENTO



⑥ SCHEDULAZIONE DI SISTEMI IN TEMPO REALE

- HARD DEADLINE SYSTEMS: IL SUPERAMENTO DELL'Deadline E' CATASTROFICO. I PROCESSI SONO SPESO PERIODICI

PROCESSI:

- ISTANTE DI RICHIESTA r (MOMENTO IN CUI IL PROCESSO ENTRA IN PLATO)
- deadline d
- tempo di esecuzione c → DATO CHE 1- E' UNA STIMA E 2- PUO' VARIARE DA ESECUZ. A ESECUZ. PRENDO C_{MAX}
- SUPPONIAMO DI SPOSTARE LA DEADLINE PIÙ AVANTI POSSIBILE:
LA DEADLINE COINCIDERA CON L'INIZIO DELLA NUOVA ESECUZIONE
DELLO STESSO PROCESSO, OVVERO
 $d := r_i + t$

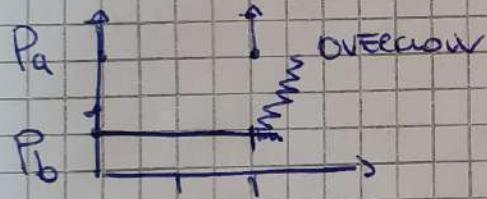
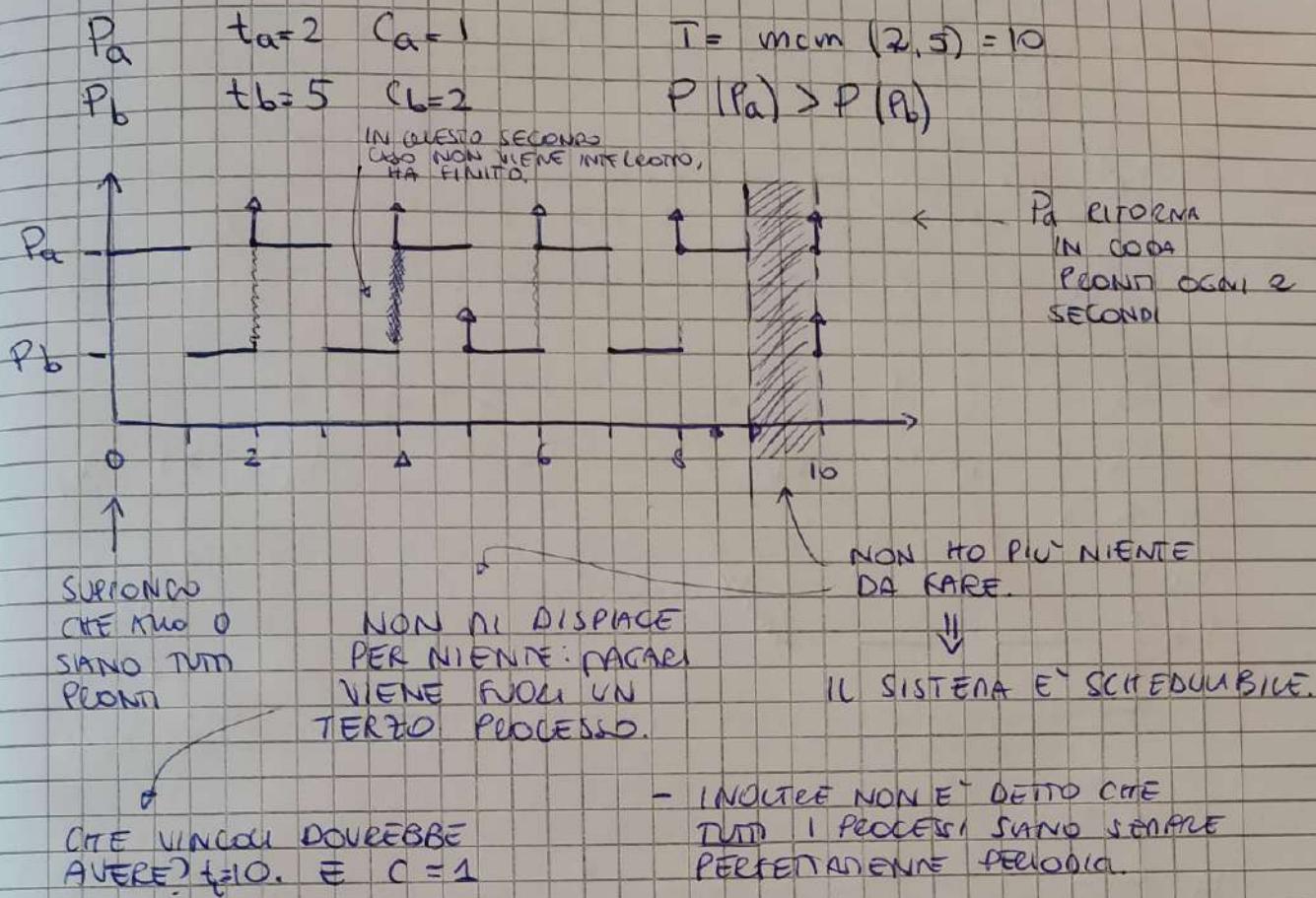
$$\text{INOLTRE } T = \text{mcm}(T_i)$$

ALGORITMO RATE MONOTONIC (E' PREEMPTIVE)

- CONVIENE ASSEGNAME LA PRIORITÀ AI PROCESSI IN BASE ALLA DURATA DEL LORO PERIODO.
IN PARTICOLARE

PRIORITÀ MIGLIA \Leftrightarrow PERIODO PICCOLO.

ESEMPPIO:



SOLO PER GLI ALGORITMI A PRIORITA' **STATICA**

RATE MONOTONIC E' **OTTIMO** PER I SISTEMI IN TEMPO REALE.
(SE NON FUNZIONA CON RM, NON FUNZIONA CON NIENTE ALTRO.
SE riesco a schedularlo con RM allora potrebbe ESSERE SCHEDULABILE ANCHE CON ALTRI ALGORITMI!)

FATTORE DI UTILIZZAZIONE

$$U \leq \sum_{i=1}^n \frac{C_i}{T_i}$$

- SI PUO' FACILMENTE VERIFICARE CHE SE $U > 1$ L'INSIEME DI PROCESSI NON E' SCHEDULABILE PERCHÉ RAPPRESENTA L'UTILIZZO TOTALE DELLA CPU PER UNITÀ DI TEMPO.

$$n = \frac{T}{T_i} \quad n \text{ E' IL NUMERO DI VOLTE PER CUI IL PROCESSO ENTRA IN ESECUZIONE IN UN PERIODO$$

CARICO TOTALE: $\sum_{i=1}^n C_i$

$$n_1 C_1 + n_2 C_2 + \dots + n_N C_N = \frac{T}{T_1} C_1 + \dots + \frac{T}{T_N} C_N = T \left(\sum_{i=1}^n \frac{C_i}{T_i} \right) = TU$$

DA QUI RISULTA CHE SE $U > 1$ ALLORA IL TEMPO RICHIESTO DA TUTTI I PROCESSI IN T SUPERÀ T

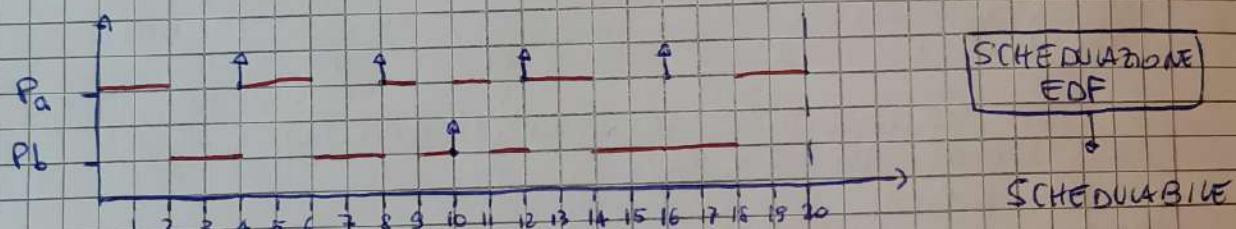
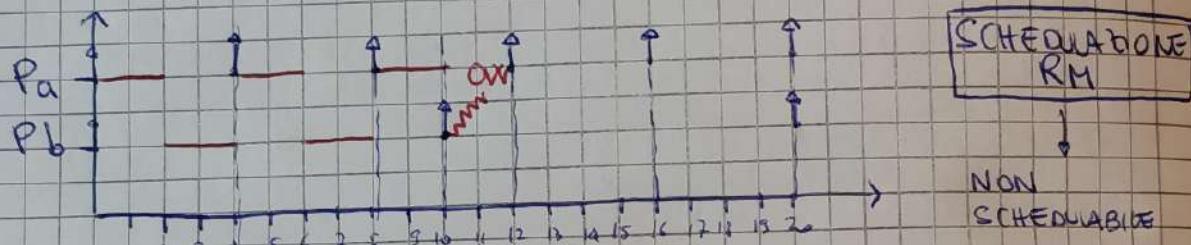
$U < 1$

NOTA: NON E' VERO CHE SE U RISULTA <= 1 ALLORA
L'INSIEME DI PROCESSI E' NECESSARIAMENTE SCHEDUABILE

Esempio: SI PUO' DEDURRE CHE U <= n $(2^{\frac{n}{2}} - 1) \approx 0.7$
(E' UNA CS, UN INSIEME DI PROCESSI PUO' ESSERE
SCHEDUABILE ANCHE SE NON E' RISPETTATA)

→ PER OTTENERE RISULTATI MIGLIORI: PRIORITA' DINAMICO

$$\begin{array}{ll} P_a & t_a = 4 \quad C_a = 2 \\ P_b & t_b = 10 \quad C_b = 5 \end{array} \quad U = \frac{2}{4} + \frac{5}{10} = 1 \quad T = 20 \quad P(P_a) > P(P_b)$$



ALGORITMO EDF (EARLIEST DEADLINE FIRST) (OTTIMO)

- SEGUE UN CRITERIO DI ASSEGNAZIONE DI TIPO DINAMICO
- IN OGNI Istante Assegna la Priorita' Più Alta al Procedo con Deadline Più Prossima.

ALGORITMO MIGLIORE DI RM, MA RICHIEDE MAGGIORI OVERITEND.

3 Soprannomi

- ① PERIODICITA' DEI PROC
- ② SPOSTAMENTO IN DIREZIONE DELLA DEADLINE
- ③ NON SI CONSIDERANO GLI OVERHEAD DEGLI ATCO

DOVUTO DA COSA?

- OGNI MOVENTO DÀ RISULTATO
LE PRIORITA' DEI PROCESSI

SINCRONIZZAZIONE DEI PROCESSI

- IN PRESENZA DI PROCESSI CONCORRENTI CON INTERAZIONE IL COMPORTAMENTO RISULTA ESSERE NON RIPRODUCIBILE.
I PROCESSI SONO ESEGUITI SUL UNICA UNITÀ DI ELABORAZIONE DI VIELE (SISTEMA MULTIPROCESSORE)

TIPI DI INTERAZIONE

(1) **COOPERAZIONE:** AD ESEMPIO UN PROCESSO PRODUTTORE PRODUCE DATI CHE DEPOSITA IN UN BUFFER DOVE SONO PRELEVATI DA UN ALTRO PROCESSO.

=> PROBLEMA DELLA SINCRONIZZAZIONE

(2) **COMPETIZIONE:** 2 PROCESSI ESEGUONO OPERAZIONI DI STAMPA SU UN'UNICA STAMPANTE. E' USATO DA VINCOLI DI REALE DISPONIBILITÀ DELLE RISORSE STESE

=> PROBLEMA DI MUTUA ESCLUSIONE

MODelli

(1) **MODELLO AD AMBIENTE GLOBALE:** CONCERNTE TIPO DI INTERAZIONE TRA PROCESSI AVVIENE TRAMITE LA MEMORIA COMUNE. I PROCESSI COMPETISCONO PER L'UTILIZZO DI RISORSE COMUNI E LE UTILIZZANO PER SCAMBARE INFO. (ARCHITETTURA MULTIPROCESSORE)

=> SEMAFORI CON SEM_WAIT E SEM_SIGNAL

(2) **MODELLO AD AMBIENTE LOCALE:** CONCERNTE TIPO DI INTERAZIONE TRA PROCESSI AVVIENE TRAMITE SCAMBIO DI MESSAGGI. (RETI DI ELABORATORI SENZA MEMORIA COMUNE). (C'È MAGGIORE OVERHEAD IN GENERALE).

=> send E receive

MUTUA ESCLUSIONE

SUPPONIAMO DI AVERE UNA STRUTTURA DATI DI QUESTO TIPO:

```
#include <stack.h>
stack[n];
int top = 1;

void Inserimento(Ty) {
    top++;
    stack[top] = y;
}

Ty Preleva() {
    Ty temp;
    temp = stack[top];
    top--;
    return temp;
}
```

CON UN'ESECUZIONE DI QUESTO TIPO

```
t0 : top++;          (P1)
t1 : temp=stack[top]; (P2)
t2 : top--;          (P2)
t3 : stack[top]=y;   (P1)
```

2 PROBLEMI:

• VIENE PRELEVATO UN DATO SBAGLIATO
• VIENE SOVRASCRITTO UN DATO UTILE
NOTA: ERRORE ALEATORIO, CI SONO DUE SUCCESSIONI.

2 PROCESSI POTREBBERO ENTRARE,
LEGGERE OCCUPATO=0 E ENTRARE.

- 1° SOLUZIONE ERRATA

prologo: while (occupato == 1);
occupato = 1;
(SEZIONE CRITICA)

epilogo: occupato = 0;

2 PROBLEMI:

1) IL PROLOGO NON È ATONICO. SANO NELLA STESSA CONDIZIONE DI PRIMA

2) IL PROCESSO PRENDE CICLI DI CPU INUTILI IN ATESA ("BUSY WAITING")

SOLUZIONE 1

USARE L'ISTRUZIONE TSL

TSL R, X // IL VALORE CONTENUTO NELLA LOCATIIONE X VIENE COPIATO NEL REGISTRO R E VIENE SCRIVO * IN X UN VALORE ≠ 0.

⇒ lock(x):

```
TSL reg, X  
CMP reg, 0  
JNE lock  
RET
```

un lock(x):

```
MOV X, 0  
RET
```

TSL BLOCCA
GLI ALTRI
PROCESSORI
(IN PARCOURSE)
(BLOCCA IL
BUS DELLA
MEMORIA)

Ci sono comunque alcuni problemi:

- 1) Busy waiting
- 2) Applicabile solo in sistemi multiprocessore
- 3) Servono sezioni critiche molto brevi

Però ho almeno il problema della 2° soluzione, perché i primi 2 processi ponavano leggere occupato = 0 uno dopo l'altro a causa di un'interruzione TEC while è occupato = 1. Ora no, perché TSL ha ruvo in un clock.

SOLUZIONE: SEMAFORI

```
void wait(s)
{
    if (s.value == 0)
        // il processo viene sospeso e
        // il suo descrittore inserito in s.queue;
    else s.value = s.value - 1;
}
```

```
void signal(s)
{
    if (<esiste almeno un processo nella coda s.queue>)
        // il descrittore del primo di questi viene estratto
        // da s.queue e il suo stato modificato in pronto;
    else s.value = s.value + 1;
}
```

- DEVO NASCITERE LE INTERRUZIONI DURANTE WAIT E SIGNAL.
• INFAT SONO FORNITE DAL NUCLEO DEL SISTEMA E SONO CHIAMATE TRAMITE CHIAMATE DI SISTEMA.

CONSIDERA I PROCESSI SIANO ESEGUITI SU PROCESSORI DIVERSI, PER GARANTIRE CHE WAIT E SIGNAL OPERINO IN NODO, PURAMENTE ESCLUSIVO BISOGNA AGGIUNGERE lock(x) E unlock(x)

lock(x);
wait (mutex);

unlock(x);

<SEZIONE CRITICA>

lock(x);
signal (mutex);
unlock(x);

- SOLUZIONE AL PROBLEMA DELLA COMUNICAZIONE

① UN PRODUTTORE E UN CONSUMATORE

PROCESSO PRODUTTORE:

{
do {

<produzione del nuovo messaggio>;
wait (spazio_disponibile);
<deposito del messaggio nel buffer>;
signal (messaggio_disponibile);
} while (!fine);
}

Processo consumatore

{
do {

wait (messaggio_disponibile);
<prelievo del messaggio dal buffer>;
signal (spazio_disponibile);
<consumo del messaggio>

} while (!fine);
}

② PIÙ PRODUTTORI E PIÙ CONSUMATORI, BUFFER DA N POSIZIONI

PRODUTTORE

do {

<produzione_messaggio>
wait (spazio_disponibile);
wait (mutex);
<deposito messaggio>
signal (mutex);
signal (messaggio_disponibile);

} while (!fine)

CONSUMATORE

do {

wait (messaggio_disponibile);
wait (mutex);
<prelievo del messaggio>
signal (mutex);
signal (spazio_disponibile);
<consumo del messaggio>

} while (!fine)

→ PERCHÉ NON PUÒ VERIFICARSI DEADLOCK?

③ "readers-writers problem"

DEVO FARLE IN MODO CHE PIÙ CONSUMATORI POSSANO LEGGERE NELLO STESSO MOMENTO, E CHE SOLO UNO SCRIVORE POSSA ACCEDERE ALLA RISORSA NELLO STESSO MOMENTO.

SHARED DATA:

- DATA SET
- SEMAFORO mutex INITIALIZATO A 1
- SEMAFORO wrt INITIALIZATO A 1
- read count INITIALIZATO A 0

IPOTESI:
INIZIO CON
UN MESSAGGIO
NEL BUFFER

PRODUTTORE

```
do {
    wait (wrt);
    < SCRIVETE DEL MESSAGGIO >
    signal (wrt);
}
while (!fine);
```

MI SERVE mutex PERCHÉ
DEVO GARANTIRE LA TUTTA
ESCLUSIONE SULLA RISORSA
CIRCA CONDIVISA, OVVIO
read count.

CONSUMATORE

```
do {
    wait (mutex);
    read count++;
    if (read count == 1)
        wait (wrt);
    signal (mutex);
    < EFFETTUO LA LETTURA >
    wait (mutex);
    read count--;
    if (read count == 0)
        signal (wrt);
    signal (mutex);
}
while (!fine);
```

Solo
il primo
da
lettura
fa la
wait (wrt)

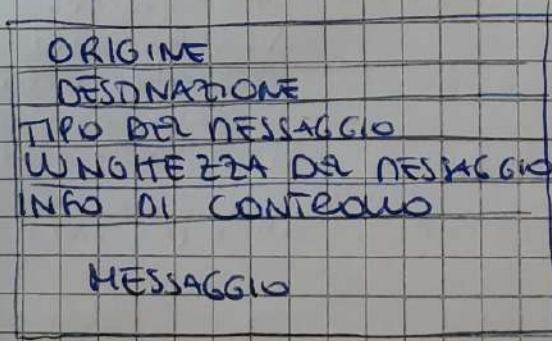
Sul wrt
posso nego
n scrittori
in attesa,
un solo
lettore

SEND E RECEIVE

CON WAIT E SIGNAL RIDONO I PROBLEMI DI CONCORRENZA E COOPERAZIONE TRA PROCESSI OPERANTI IN UN AMBIENTE A NORDIA CONUNE.

IN UN AMBIENTE A NORDIA VOLTE NUOVA AVVIENE TRAMITE SCAMBIO DI MESSAGGI. PUÒ ESSERE USATO ANCHE IN SISTEMI MULTIPROCESSORE A NORDIA CONDIVISA O IN SISTEMI A SINGOLO PROCESSORE

FORMATO DEL MESSAGGIO:



LASCIANO I SEMAFORI
COSÌ CON HONO
IMPLEMENTARE:

SE NATA CORA wrt
(C'È UNO SCRITTORE)
DARGLI LA PREFERENZA
CUI LETTORI

NOTA: OGNI VOLTA CHE
DIANO UNA PRIORITÀ SI
CREA IL PROBLEMA DELL'
STARVATION

COMUNICAZIONE DIRETTA

→ SIMMETRICA: send (P2, messaggio)
receive (P1, messaggio)
(MITTENTE E DESTINATARIO NON SONO SIMETRICALMENTE A VICENDA)

ASIMMETRICA:

send (P2, messaggio)
receive (fd, messaggio)

(IL MITTENTE INVIA UN MESSAGGIO A P2, MENTRE IL RICEVITORE NON SA DA CHI GLO ARRIVI IL MESSAGGIO FINO A CHE NON ESEGUE LA receive)

```
processo produttore P
pid C=...;
main()
{
    msg M;
    do {
        produci(&M);
        ...
        send (C, M);
    } while (!fine);
```

Figura 3.7 Comunicazione diretta simmetrica

```
processo produttore P
pid C=...;
main()
{
    msg M;
    do {
        produci(&M);
        ...
        send (C, M);
    } while (!fine);
```



```
Processo consumatore C
...
main()
{
    msg M;
    pid id;
    do {
        receive(&id, &M);
        ...
        consuma(M);
    } while (!fine);
```

COMUNICAZIONE INDIRETTA

→ I MESSAGGI NON SONO INVIAI DIRETTAMENTE AI PROCESSI, MA DEPOSITATI IN, E PRELEVATI DA, UNA STRUTTURA DATI DETTA PORTA.

send (mailbox, messaggio);
receive (mailbox, messaggio);

- SINCRONIZZAZIONE TRA PROCESSI COMUNICANTI

SEND → ASINCRONA → IL PROCESSO PROSEGUE NEGLI SUA ESECUZIONE

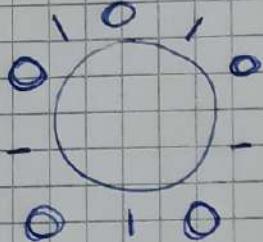
SINCRONA → BLOCCA IL MITTENTE FINO AL RICEVIMENTO DEL MESSAGGIO DAL DESTINATARIO

RPC
(REMOTE PROCEDURE CALL)

→ BLOCCA IL MITTENTE FINO A CHE IL MESSAGGIO NON È STATO CONSUMATO DAL DESTINATARIO

RECEIVE → BLOCCANTE → SE NON CI SONO MESSAGGI
 IL PROCESSO VIENE BLOCCATO
 → NON BLOCCANTE → IL PROCESSO PROSEGUE
 ANCHE SE NON CI SONO MESSAGGI.

DINING PHILOSOPHERS PROBLEM



OGNI FILOSOFO HA BISOGNO DI 2 CHOPSTICK
 (UNA SUA DX E UNA SUA SX) PER
 MANGIARE.

SHARED DATA:

- BOWL OF RICE
- SEMAPHORE OF CHOPSTICK [5] = 1

POTREI PENSARE UNA
 SOLUZIONE DEL GENERE:

```

do {
    wait (chopstick[i]);
    wait (chopstick[(i+1)%5]);
    <eat>
    signal (chopstick[i]);
    signal (chopstick[(i+1)%5]);
    <think>
}
while (!Fine)
    
```

C'È UN PROBLEMA DI
 DEADLOCK: SE TUTTI
 E 5 PRENDERESSERO
 INSieme IL 1° CHOPSTICK
 (VFCITO) NESSUNO
 POTREBBE MANGIARE.

NOTA: IL CODICE È SICURAMENTE BENE,
 E' LO SCEDULER E LE PRIORITÀ
 DEI PROCESSI IL "PROBLEMA".

MONITOR

È UN'ABSTRAZIONE DI ALTO LIVELLO CHE GARantisce UN
 MECCANISMO CONVENIENTE E FUNZIONALE PER LA SINCRONIZZAZIONE
 DEI PROCESSI.

monitor monitor-name

```

{
    // shared variable declarations
    procedure P1(..){...}
    procedure Pn(..){...}
    initialization code(..){...}
}
    
```



X.wait() \Rightarrow IL PROCESSO E' SEMPRE SOSPESO FINO A CHE
QUALCUNO NON INVOCI...

X.signal() \Rightarrow FA RIPRENDERE INO DEI PROCESSI CHE HA FATTO X.wait().
SE NON CE NE E' NESSUNO, BONA.

2 TIPOLOGIE:

1) SIGNAL AND WAIT: SE P FA SIGNAL E Q ERA IN WAIT, P SI
BUCCIA DAL MONITOR E ATTENDE FINO A CHE Q NON ESCE

2) SIGNAL AND CONTINUE: SE P FA SIGNAL E Q ERA IN WAIT, Q
ATTENDE CHE P ESCA DAL MONITOR

SOLUZIONE AI DINING PHILOSOPHERS (COL ~~MONITOR~~ MONITOR)

```
monitor DiningPhilosophers
{
    enum { THINKING, HUNGRY, EATING } state [5];
    condition self [5];

    void pickup (int i) {
        state[i] = HUNGRY;
        test(i);
        if (state[i] != EATING) self [i].wait;
    }

    void putdown (int i) {
        state[i] = THINKING;
        // test left and right neighbors
        test((i + 4) % 5);
        test((i + 1) % 5);
    }

    void test (int i) {
        if ( (state[(i+4) % 5] != EATING) &&
            (state[i] == HUNGRY) &&
            (state[(i+1) % 5] != EATING)) {
            state[i] = EATING;
            self[i].signal();
        }
    }

    initialization_code() {
        for (int i = 0; i < 5; i++)
            state[i] = THINKING;
    }
}
```

Each philosopher / invokes the operations pickup() and putdown() in the following sequence:

DiningPhilosophers.pickup (i);

EAT

DiningPhilosophers.putdown (i);

No deadlock, but starvation is possible

QUA CI ARRIVO SE NON SONO
ENTRATO UNA DENTRO.
QUINDI SIGNIFICA CHE
DEVO ATTETTERMI IN
ATTESA SUA VARIAZIONE
self[i]

QUINDI SIGNIFICA CHE IL
MIO FILOSOFO VOLTEBBE
MANGIARE MA IN CONCILIO
MODO O CECINO MA (UA DX
O QUANDO UA SUA SX STA
GIA' MANGIANDO

QUANDO RACCUO PUTDOWN TESTO SIA
A DX CHE A SX.
QUINDI DO UNA SIGNAL SE
UNO DI QUESTI A DX/SX
PUO' PARTEC.

- LA DIFFERENZA CON #
E' CHE IO NON MI
IMPOSSESSO / NU DURENTE
DI UNA BACCERNA SE
NON HO LA POSSIBILITA'
DI PRENDERE ANCHE LA
2°. RINANCO HUNGRY E
BASTA.

■ Signal and wait - P waits until Q leaves monitor or waits for another condition
 ■ Variables
 RAPPRESENTA IL POSSESSO DEL MONITOR
 semaphore mutex; // (initially = 1)
 semaphore next; // (initially = 0)
 int next_count = 0;
 ■ Each procedure F will be replaced by
 wait(mutex);
 body of F;
 if (next_count > 0)
 signal(next);
 else
 signal(mutex);

QUA CI SONO PROCESSI BLOCCATI DENTRO IL MONITOR

FINITA LA PROCEDURA DEVO PASSARE IL MONITOR A CHI STA GIA DENTO, O A CHI E' FUORI AD

ASPERARE. CHI STA DENTRO IL MONITOR?

I PROCESSI CHE SI SONO BLOCCATI DENTRO IL MONITOR PER QUALCHE RAGIONE

DENTRO LA PROCEDURA, QUANDO SONO DENTRO AL MONITOR, RACCUO X_WAIT().
 DICO CHE LA WAIT E' BLOCCANTE,
 RACCUO LA SIGNAL A QUALCUNO CHE C'E' DENTRO IL MONITOR O A CHI ASPIRA PIU'.
 (LA SIGNAL SOLO IL PROCESSO IN CODA PIOMBI, QUINDI ALLA FINE LI SUSPENDO CON LA WAIT)

SE C'E' QUALCUNO IN CODA SU QUESTA VARIABILE, BENE, ATTENDO.
 NON RACCUO NIENTE.

HO RISVEGUATO QUALCUNO, QUINDI DEVO SUSPENDERLO /O.

MI METTO IN CODA FIANCO AL MONITOR

DEADLOCK

[EN]: 2 O PIU' PROCESSI, QUANDO OGNI UNO POSSIEDE ALMENO UNA RISORSA E FA RICHIESTA PER UN'ALTRA.

MODELLIZZAZIONE DEL PROBLEMA:

- R₁, R₂...R_m SONO LE RISORSE (CPU CYCLES, MEMORY SPACE, I/O)
- EACH R_i HAS W_i INSTANCES
- OGNI PROCESSO UTILIZZA LE RISORSE COME SEGUTE:

- 1) RICHIESTA
- 2) UTILIZZO
- 3) RILASCIO.

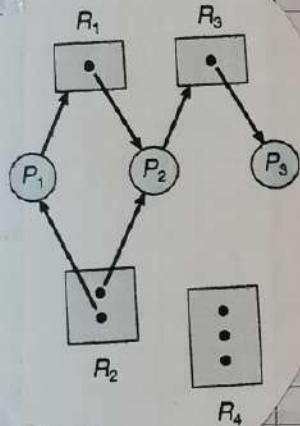
PROBLEMA DEL DEADLOCK: HO UN

SISTEMA MUOGLIO PROGRAMMATO E QUINDI PIU' PROCESSI COMPETONO PER L'ACCESO A DUE RISORSE

[CN]

- ① LE RISORSE POSSONO ESSERE UTILIZZATE DA UN SOLO PROCESSO A UNA VOLTA (**ONE EXCLUSION**)
- ② I PROCESSI TRATTENGONO LE RISORSE CHE GIÀ POSSESSANO MENTRE RICHIEDONO RISORSE ADDIZIONALI (**HOLD AND WAIT**)
- ③ LE RISORSE GIÀ ASSEGNAME AL PROCESSI NON POSSONO ESSERE REVOCATE (**NO PREEMPTION**)
- ④ ESISTE UN INSIENE DI PROCESSI P_1, P_2, \dots, P_k TUTTI CHE P_i È IN ATESA DI UNA RISORSA POSSEDUTA DA P_{i+1} , CHE È IN ATESA DI UNA RISORSA DI $P_{i+2} \dots$ (CHE È IN ATESA DI UNA RISORSA DI P_i)

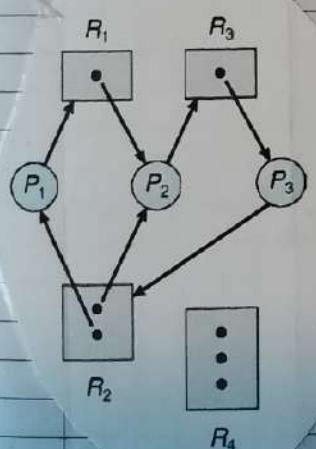
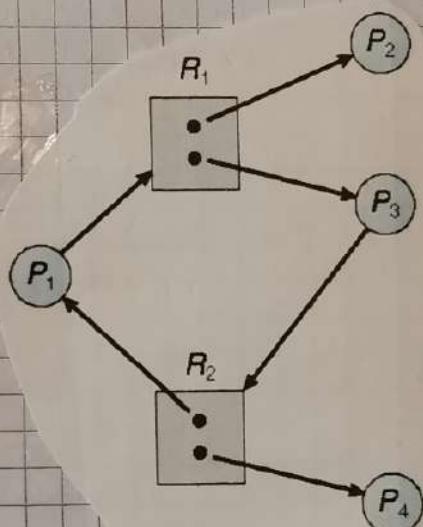
(LA ④ È UNA CNS SE OGNI RISORSA HA UNA FOCA INSTANTA)



NO DEADLOCK:

VIENE MENO LA ②
 P_3 TRAMENDE UNA RISORSA MA NON NE RICHIERA' UN'ALTRA.

VIENE MENO ANCIE LA ④



DEADLOCK:

C' SONO ADDIETTURA 2 CICLI:

$R_2 P_1 R_1 P_2 R_3 P_3 R_2$

E $R_2 P_2 R_3 P_3 R_2$

CYCLE BUT NO DEADLOCK:

ESISTE UN CICLO:

$P_1 R_1 P_3 R_2 P_1$

MA SIA R_1 CHE R_2 SONO TRAMENDE ANCHE DA 2 PROCESSI (P_2, P_4) CHE NON SONO IN STATO QUINDI PRIMA O POI LE RIVISERANNO.

DEADLOCK PREVENTION (PREVENZIONE STANZA)

• CONSISTE NELL'ASSICURARE CHE NELLA MIGLIORE SICUREZZA OGNI PROGRAMMA UNA VOLTA A CN VENGA MENO.

- ① LA NUOVA ESCLUSIONE NON PUÒ ESSERE EVITATA, OMNIAVIETE
- ② HOLD AND WAIT:

SI PUÒ IMPORRE CHE UN PROCESSO RICHIESTA UN MINIMO DI RISORSE CHE GLI SERVIRANNO. SE SONO TUTTE DISPONIBILI PROSEGUE, ALTRIMENTI VIENE BLOCCATO.

DIFEM: IL PROGRAMMATORI DEVE CONOSCERE IN ANTICIPO QUALE RISORSE RICHIESTA IL PROCESSO, IL PROCESSO DEVE AMMENDARE CHE NELLE RISORSE SONO LIBERE INVECE DI INIZIARE L'ESECUZIONE CON QUELLI GIÀ DISPONIBILI, BASSA UTILIZZAZIONE DELLE RISORSE

- ③ NO PREVENTION:

SI PUÒ IMPORRE CHE UN PROCESSO CHE POSSIEDE GIÀ DELLE RISORSE, SE NE RICHIESTA DI NUOVE E NON SONO DISPONIBILI, PRIMA DI BLOCCARSI RILASCI quelle che già possiede. IN ALTERNATIVA, IL SISTEMA OPERATIVO POTREBBE SOTTRAIRE RISORSE CHE SONO A UN PROCESSO CHE RISULTA BLOCCATO

DIFEM: LA SOTTRAZIONE DI RISORSE A UN PROCESSO È CORRUZIONE

- ④ ATTESA CIRCOLARE:

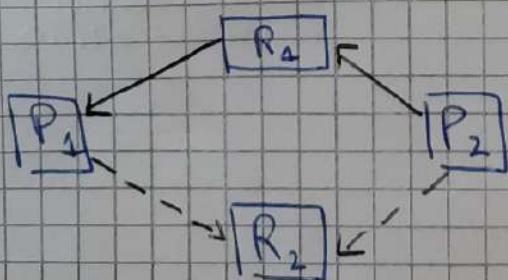
SI PUÒ IMPORRE CHE LE RISORSE VENGANO ACCEDUTE DA PROCESSI SEGUENDO UN ORDINE PREDEFINITO:

LE RISORSE VENGONO ORDINATE IN UNA GERARCHIA DI ZIVELLI:
SE P_i ALREADY HOLDS R_k, IT CANNOT REQUEST R_{k-i}

DIFEM: UN NUOVO INGRESSO DI NUOVE RISORSE

DEADLOCK AVOIDANCE (PREVENZIONE DINAMICA)

- ④ RESOURCE LOCATION GRAPH (1 ISTANZA PER RISORSE)



- P₁ E P₂ FARNO RICHIESTA PER R₁.
SE P₁ MUOCA R₁ ALLORA HUO UN DEADLOCK

② ALGORITMO DEL BANCHIERE (PIÙ Istanze per Risorsa)

- OGNI PROCESSO DEVE DICHIARARE A PRIORI QUANTE ISTANZE DI RISORSA USERA' AL MASSIMO
- QUANDO IL PROCESSO FA RICHIESTA PER UNA RISORSA POTREBBE DOVER ASSEGNARLE
- IL PROCESSO DEVE RITORNARE TUTTE LE RISORSE IN UN MODO FINITO

Let n = number of processes, and m = number of resources types.

- **Available:** Vector of length m . If Available [j] = k , there are k instances of resource type R_j available
- **Max:** $n \times m$ matrix. If Max [i,j] = k , then process P_i may request at most k instances of resource type R_j
- **Allocation:** $n \times m$ matrix. If Allocation [i,j] = k then P_i is currently allocated k instances of R_j
- **Need:** $n \times m$ matrix. If Need [i,j] = k , then P_i may need k more instances of R_j to complete its task

$$\text{Need } [i,j] = \text{Max}[i,j] - \text{Allocation}[i,j]$$

1. Let Work and Finish be vectors of length m and n , respectively. Initialize:

Work = Available
Finish [i] = false for $i = 0, 1, \dots, n-1$

2. Find an i such that both:

- (a) Finish [i] = false
 - (b) Need $_i \leq$ Work
- If no such i exists, go to step 4

3. Work = Work + Allocation $_i$

Finish [i] = true
go to step 2

4. If Finish [i] == true for all i , then the system is in a safe state

Request = request vector for process P_i . If Request $_i[j] = k$ then process P_i wants k instances of resource type R_j

1. If Request $_i \leq$ Need $_i$, go to step 2. Otherwise, raise error condition, since process has exceeded its maximum claim

2. If Request $_i \leq$ Available, go to step 3. Otherwise P_i must wait, since resources are not available

3. Pretend to allocate requested resources to P_i by modifying the state as follows:

Available = Available - Request $_i$
Allocation $_i$ = Allocation $_i$ + Request $_i$
Need $_i$ = Need $_i$ - Request $_i$

- If safe \Rightarrow the resources are allocated to P_i
- If unsafe $\Rightarrow P_i$ must wait, and the old resource-allocation state is restored

ESERCIZIO

CHE ATTIVITÀ SONO POSSIBILI

P₀, P₁, P₂, P₃, P₄

RISORSE: A (10 Istanze), B (5 Istanze), C (7 Istanze)

	<u>ALLOCATION</u>	<u>NEED</u>	<u>AVAILABLE</u>
P ₀	A B C 0 1 0	A B C 7 4 3	A B C 3 3 2
P ₁	2 0 0	1 2 2	
P ₂	3 0 2	6 0 0	
P ₃	2 1 1	0 1 1	
P ₄	0 0 2	4 3 1	

- TUTTI I PROCESSI SONO NON TERMINATI PERCIÒ VOGLIE RITORNO

E' SAFE QUESTO STATO? PROVIAMO A CREARE DELLE EVOLUZIONI

<u>PROCESSI</u>	<u>AVAILABLE</u>
P ₁	5 3 2
P ₃	7 4 3
P ₄	7 4 5
P ₀	7 5 5
P ₂	1 0 5 7

⇒ HO TROVATO UNA SEQUENZA DI ESECUZIONE

↓
STATO SAFE

- SUPPONIAMO CHE P₁ RICHIEDA 1,0,2 ⇒ AVAILABLE CAMBIA 2,3,0

NEED (P₁) CAMBIA
ALLOC (P₁) CAMBIA

STATO IPOTETICO

	<u>ALLOC</u>	<u>NEED</u>	<u>AV</u>
P ₀	A B C 0 1 0	A B C 7 4 3	A B C 2 3 0
P ₁	3 0 2	0 2 0	
P ₂	3 0 2	6 0 0	
P ₃	2 1 1	0 1 1	
P ₄	0 0 2	4 3 1	

SEQUENZA

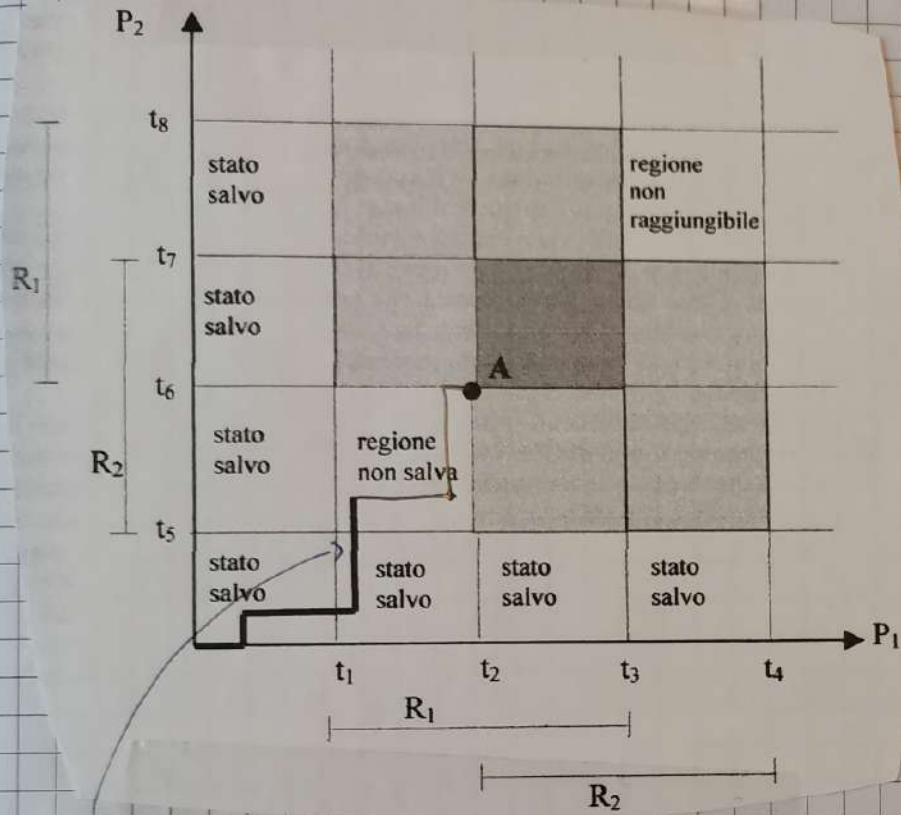
P ₁	5 3 2
P ₃	7 4 3
P ₀	7 5 3
P ₂	1 0 5 5
P ₄	1 0 5 7

⇒ STATO SAFE: P₁ PUÒ AVECRE 1,0,2

(3,3,0) PER P4? NO, NON HO ABBASTANZA DISPONIBILITÀ

(0,2,0) PER P0? POTREBBE, DEVO FARLO IL CALCOLO DELL'STATO FINITO

ALTRO ALGORITMO:



CAPOGLIO MA DOVRA' IN RISERVE
MODIFICANDO QUESTO SEGUENTI VERNUZIE.
INPEDIRA' LA WAIT DA PARTE DI P2, SU R2.

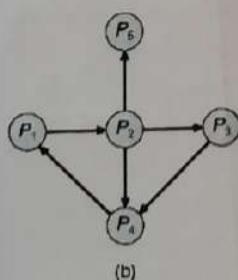
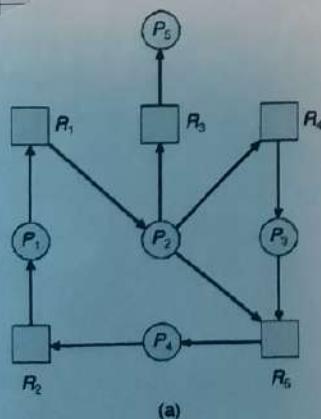
DEADLOCK DETECTION

PERMETTE CHE IL SISTEMA ENTRI IN DEADLOCK. IL SISTEMA OPERATIVO
PERIODICAMENTE NEGLI IN ESECUZIONE UN ALGORITMO PER
VERIFICARE SE ESISTONO PROCESSI COINVOLTI IN UN BLOCCO CICLO.
TRAMITE IL WAIT-FOR GRAPH

SOLO SE
C'E' UNA
SINGOLA ISTANZA
PER RISORSA

SE C'E' UN CICLO, MUORE
ESISTE UN DEADLOCK

↓
AZIONI DI
RECUPERO



Resource-Allocation Graph

Corresponding wait-for graph

SE LE RISORSE HANNO PIÙ DI UN'ISTANZA

- **Available:** A vector of length m indicates the number of available resources of each type.
 - **Allocation:** An $n \times m$ matrix defines the number of resources of each type currently allocated to each process.
 - **Request:** An $n \times m$ matrix indicates the current request of each process. If $\text{Request}[i][j] = k$, then process P_i is requesting k more instances of resource type R_j .
1. Let Work and Finish be vectors of length m and n , respectively initialize:
 - (a) $\text{Work} = \text{Available}$
 - (b) For $i = 1, 2, \dots, n$, if $\text{Allocation}_{i,i} > 0$, then $\text{Finish}[i] = \text{false}$; otherwise, $\text{Finish}[i] = \text{true}$
 2. Find an index i such that both:
 - (a) $\text{Finish}[i] == \text{false}$
 - (b) $\text{Request}_{i,i} \leq \text{Work}$If no such i exists, go to step 4
 3. $\text{Work} = \text{Work} + \text{Allocation}_{i,i}$
 $\text{Finish}[i] = \text{true}$
go to step 2
 4. If $\text{Finish}[i] == \text{false}$, for some i , $1 \leq i \leq n$, then the system is in deadlock state. Moreover, if $\text{Finish}[i] == \text{false}$, then P_i is deadlocked

Algorithm requires an order of $O(m \times n^2)$ operations to detect whether the system is in deadlocked state

- Five processes P_0 through P_4 ; three resource types A (7 instances), B (2 instances), and C (6 instances)

- Snapshot at time T_0 :

	<u>Allocation</u>	<u>Request</u>	<u>Available</u>
	ABC	ABC	ABC
P_0	010	000	000
P_1	200	202	
P_2	303	000	
P_3	211	100	
P_4	002	002	

- Sequence $\langle P_0, P_2, P_3, P_1, P_4 \rangle$ will result in $\text{Finish}[i] = \text{true}$ for all i

È PRATICAMENTE UGUALE
AL MIGLIORATO DEL BANCHIERE

DIFERENZE:

1. NEL BANCHIERE C'È AVAILABLE, MAX, ALLOCATION E NEED.

QUA C'È AVAILABLE, ALLOCATION E REQUEST

2. NEL BANCHIERE L'ALGORITMO SI SVOLGE NEL MONDO IN CUI UN PROCESSO EFFETTA UNA NUOVA RICHIESTA PER CAPIRE SE SA ACCETTABILE O NO.

QUESTO ALGORITMO VIENE ESEGUITO OGNI TANTO PER VERIFICARE SE CI SI TRONI IN UNA SITUAZIONE DI DEADLOCK O MENO.

- SE C'È DEADLOCK, COSA FARE?

1. ABORT DI TUTTI I PROCESSI

2. ABORT DI UN PROCESSO ALL'VOLTA FINO AD OTTENERE UNO STATO SAFE

SCEGLIERE IN BASE ALLA PRIORITÀ DEL PROCESSO, TEMPO DI ESECUZIONE, TEMPO RIMANENTE, RISORSE IN UTILIZZO RISORSE NEEDED,
SE IL PROCESSO È INTERATO O BATCH

MIGLIOR
INTERAZIONE
COLLEZIONE.

GESTIONE DELLA MEMORIA

PROCESSO → UNITÀ DI ELABORAZIONE (CPU)
→ MEMORIA

- ANALOGIE E DIFFERENZE CON LA GESTIONE DELLA CPU

ANALOGIE: PER ESISTERE UN PROCESSO HA INPLICATAMENTE BISOGNO DI UNA CPU E DI UN'AREA DI MEMORIA IN CUI IL CODICE RISIDE. A OGNI PROCESSO VIENE ASSOCIAUTO UN PROCESSORE VIRTUALE, E UNA MEMORIA VIRTUALE.

DIFFERENZE:

LA CPU VIRTUALE È COSTITUITA DA UNA STRUTTURA DANNA MOLTA IN MEMORIA, NEL DESCRIMARE AL PROCESSO. CIO' NON È VERO PER LA MEMORIA VIRTUALE.

LA MEMORIA PRINCIPALE A DIFFERENZA DELLA CPU PUÒ ESSERE UTILIZZATA PER FORNIRE SUPPORTO A DIVERSI PROCESSI CONTEMPORANEAMENTE (PROBLEMA DI PROTEZIONE E OPPORTUNITÀ DI SHARING DEL CODICE)

DEFINIZIONE DI MEMORIA VIRTUALE: È L'INSIEME DI TUTTE LE LOCALIZZAZIONI DI MEMORIA E DELLE INFORMAZIONI IN ENTE CONDENSE I CUI INDIRIZZI POSSONO ESSERE GENERATI DAL CPU DURANTE L'ESECUZIONE DEL PROCESSO.

IN SOSTANZA È LA MEMORIA NECESSARIA A CONTENERE IL CODICE DEL PROCESSO, I DATI SU CUI DEVE OPERARE, E LO STACK.

COME VIENE CREATO IL FILE ESEGUITIBILE?

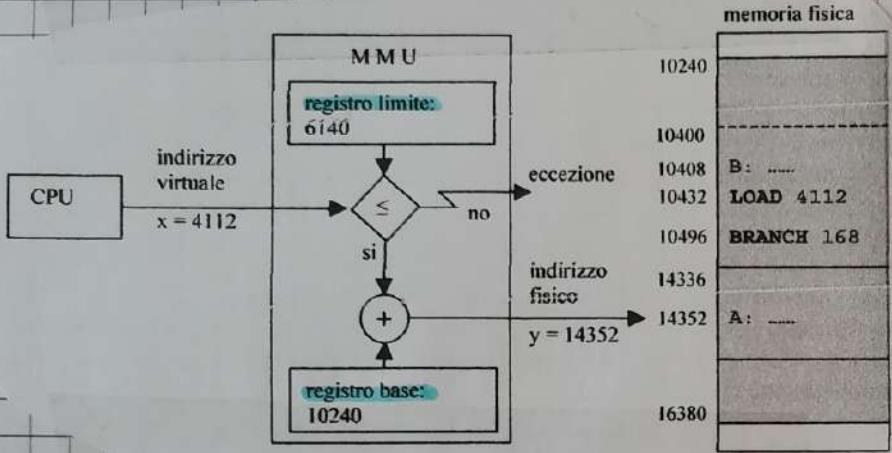
UN PROGRAMMA CONSISTE IN UN INSIEME DI MODULI CHE VENGONO COMPILATI SEPARATAMENTE GENERANDO DEI FILE .O. INFINE VENGONO TUTTI COLLEGATI TRA DISSOCIATI, LIBRERIE... DAL CARICATORE

COMPILAZIONE → LINKER → CARICATORE

① RICOLAZIONE STatica E DINAMICA

RICOLAZIONE STatica: IL CARICATORE È UN "CARICATORE RICOLANTE": NEL TRASFERIRE IN MEMORIA IL MODULO DI CARICAMENTO MODIFICA TUTTI GLI INDIRIZZI VIRTUALI A FISICI SONNANDOVI L'INDIRIZZO INIZIALE DI CARICAMENTO. SOLUZIONE SENZIALE, MA UNA VOLTA CARICATO IN MEMORIA UN PROGRAMMA, QUESTO NON PUÒ PIÙ ESSERE RICOLLOCATO IN UNA DIVERSA POSIZIONE DI MEMORIA. IL PROBLEMA DERIVA DAL FATO CHE SE AD ESEMPIO CHIAMO UNA FUNZIONE, NEGLIO STACK VIENE SCRISSO L'INDIRIZZO FISICO DI RITORNO. SE IN QUESTO MOMENTO VIENE FATTO SWAP OUT E SWAP IN IN UN PUNTO DIVERSO, COMUNQUE L'INDIRIZZO DI RITORNO NELLO STACK RIMANE TALE, E ALLA RETURN ANDREI A FINIRE A INDIRIZZI DIVERSI.

RILOCAZIONE DINAMICA: LA RILOCAZIONE VIENE EFFETTUATA IN FASE DI ESECUZIONE TRAMITE LA MMU DISPOSITIVO PIÙ VERSATILE TRA CPU E MEMORIA CENTRALE. LA CPU PRODUCE INDIRIZZI VIRTUALI, LA MMU SI OCCUPA QUINDI DI ELABORARE TUTTI INDIRIZZI PER RENDERLI FISICI ATTENUTO LA FUNZIONE DI RILOCAZIONE.



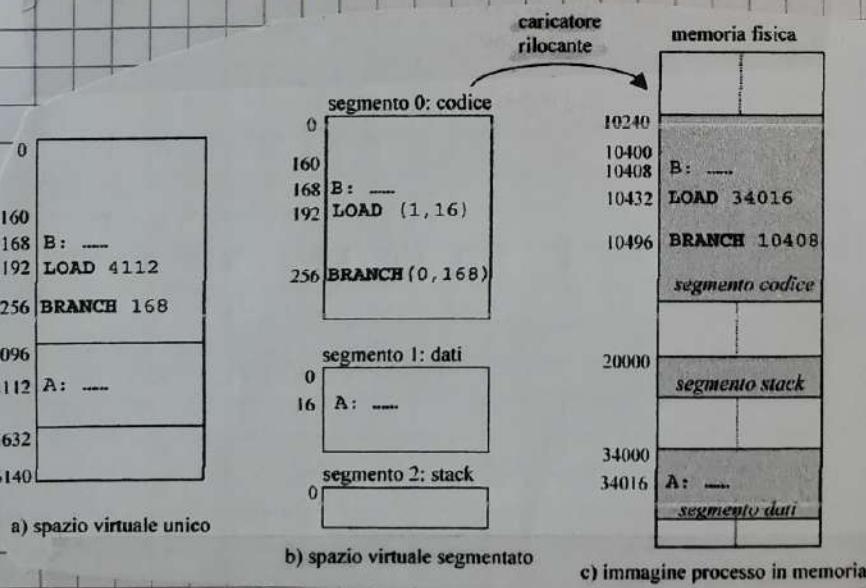
DEVO PERO'
SCRIVERE IN
REGISTRO LIMITE
E REGISTRO
BASE.

② ORGANIZZAZIONE DELLA MEMORIA VIRTUALE

FINO AD ADESSO E' STATA FATTA L'IPOTESI CHE IL LINKER ALLOCHE TUTTI I MODULI COMPOSTI IL PROGRAMMA A INDIRIZZI VIRTUALI CONSECUTIVI: POTREBBE NON ESSERE COSÌ, PUÒ DIVIDERE IL CODICE IN SEGMENTI.

SE LO SPAZIO VIRTUALE È SEGMENTATO, OGNIUNO DEI SEGMENTI PUÒ ESSERE RILOCATO IN MEMORIA FISICA INDEPENDENTEMENTE DAGLI ALTRI.

NEL CASO DI RILOCAZIONE STANCA E' IL CARICATORE RILOCANTE A DOVER MANTENERE UNA TABEUA CON GLI INDIRIZZI INIZIALI DI OGNI SEGMENTO.



- INVECE NEL CASO DI RILOCAZIONE DINAMICA BISOGNEREBBE AGGIUNGERE COPPIE DI REGISTRI NELLA MMU A SECONDA DEL NUMERO DI SEGMENTO

③ ALOCAZIONE DELLA MEMORIA FISICA

FINO AD ADESSO ALLA CONCETTA' DI INDIRIZZI VIRTUALI CORRISPONDE CONCETTA' DI INDIRIZZI FISICI. E' UN VINCOLO.

FRAGMENTAZIONE: MAGARI HANNO TANTI FRAMENTI IN MEMORIA, NELL'UNO ABBASTanza GRANDE DA CONTENERE IL SEGMENTO, MA LA SOGA SOMMA LO SAREBBERE.

CON LA RICLOCkATION DINAMICA POTREI RISOLVERE IL PROBLEMA
ATTRAVERSO IL COMPATTAMENTO: SE CI SONO DOTTI FERMAMENTI SPOR-
TIVI I PROGRAMMI IN MODO DA ELIMINARLI E OMENGERE UN'AREA
UNICA \rightarrow DISPENDIOSO.

RICLOCkATION STATIC: A INDIRIZZI VIRTUALI CONGHI DEVONO NECESSARIALMENTE CORRISPONDERE INDIRIZZI FISICI CONGHI

RICLOCkATION DINAMICA: DUE ISTRUZIONI, PUR ESSENDO CONGHI NELLO SPAZIO VIRTUALE, POTREBBERO ESSERE ALOCATE ATTRAVERSAMENTE IN TEORIA, IN LAVORAZIONI NON CONGHI DELLO SPAZIO FISICO.

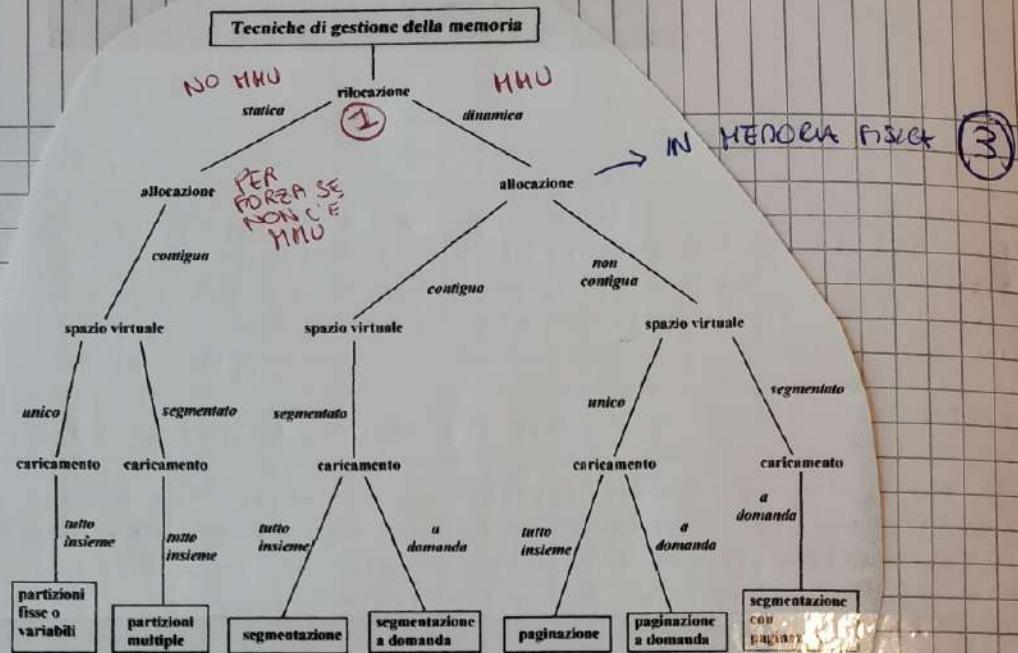
SI OMENGE CHE COMPLICANDO LA MMU, TRAMITE LA FUNZIONE DI RICLOCkATION $y = f(x)$

④ DIMENSIONAMENTO DELLA MEMORIA VIRUALE

SE LA DIMENSIONE DEL PROGRAMMA SUPERVA LA DIMENSIONE DELLA MEMORIA FISICA?

TECNICA DEI OVERLAY, CHE PREVEDE UN PARCOLORE CINGERE
IL PROGRAMMADORE DVE SPECIFICARE ONDRA PAER CARICARE INITIATIVAMENTE
IN MEMORIA E QUANDO E' RICHIESTA UNA COMPONENTE NON
ANCORA CARICATA SE NE OCUPA IL CARICATORE.
Nella RICLOCkATION DINAMICA E' POSSIBILE CONSUMARE LA MMU
E CREARE SPAZI VIRTUALI DI DIMENSIONI SUPERIORI A QUELLA DELLA
MEMORIA FISICA.

SE VIENE GENERATO UN INDIRIZZO VIRTUALE IL WI CORRISPONDENTE
FISICO ANCORA NON ESISTE, VIENE GENERATA UN'INFORMAZIONE AI
SISTEMA PER CARICARE IN MEMORIA L'INFORMAZIONE RICHIESTA
("CARICAMENTO A DOMANDA")



- PARTIZIONI FISSE E VARIABILI

LA MEMORIA VIRTUALE E' COSTITUITA DA UN UNICO SPAZIO VIRTUALE CONIGUO. E' NECESSARIO CERCARE UNA PORZIONE DI SPAZIO LIBERO DI DIMENSIONE \geq , TALE PARTIZIONE DEVE RIPARARE ASSOCIATA AL PROCESSO IN CASO DI SWAP IN-OUT.

PARTIZIONI FISSE: LA MEMORIA VIENE SVOLGIDA IN MOLTI PARZIONI FISSE, LA PRIMA CONFERMA IL SISTEMA OPERAIVO.

FRAGMENTAZIONE INTERNA: E' LO SPAZIO RIMANENTE LIBERO D'UNA PARZIONE NON OCUPATA DAL PROCESSO.

PER OGNI PARZIONE SI MANTIENE UNA LISTA DI PROCESSI CON PONTELLI ROUND-ROBIN E CONSEGUENTE SWAP OUT/IN.

PARTIZIONI VARIABILI: INIZIALMENTE CI E' SOLO UN'UNICA GRANDE PARZIONE E NAN NAN OGNI VOLTA CHE UN PROCESSO VIENE AVVOLTO LA MEMORIA DIVENTA SCARSA.

BISOGNA QUINDI MANTENERE UNA LISTA CON LE PARZIONI LIBERE (O CHE SONO STATE RIASCIATE) E CON LA LORO DIMENSIONE E SE CI SONO 2 PARZIONI LIBERE ADACQUA VANNO UNITE.

SCHEMA BEST-FIT

LISTA ORDINATA PER DIMENSIONE CRESCENTE DELLE PARZIONI.

2 INCONVENIENTI: OUVIMENTE SUCCEDERÀ LA PARZIONE PIÙ PICCOLA POSSIBILE A CONTENERE IL PROGRAMMA, SI CERCHERÀ UN PICCOLO FRAMMENTO

=> FRAGMENTAZIONE

SCANSIONE DALL'INTESA LISTA IN FASE DI RIASCUO PER VERIFICARE SE SIA CONIGUA AD UN'AUTRA PARZIONE

SCHEMA FIRST-FIT

LISTA ORDINATA PER INDICIZI CRESCENTI

PRO: BASTA GUARDARE L'ELEMENTO PRECEDENTE E SUCCESSIVO ALLA LISTA, NON SCOPPIERA NADA.

- PROTEZIONE RI SOLTA TRA I "REGISTRI DI FONTE/OGGETTO" AL LIVELLO DI ARCHITETTURA DELL'CPU

- PARTIZIONI MULTIPLE

LO SPAZIO VIRTUALE E' SEGMENTATO, AD ESEMPIO IN CODICE, DATI E STACK. L'ALLOCAZIONE DI UN NUOVO PROCESSO ORA NON IMPlica PIU' TROVARE UNA GRANDE PARTIZIONE, MA 3 ALCI' PICCOLE NON NECESSARIAMENTE CONSECUTIVE TRA LORE.

- MEMORIA SEGMENTATA

RICOLAZIONE: DINAMICA;
ALLOCAZIONE MEN FISICA: CONSECUTIVA
SPAZO VIRTUALE: SEGMENTATO
CARICAMENTO: NHO INSIEME

- LA MMU DEVE
POSSEDERE 3
COPPIE DI REGISTRI
UNA COPPIA PER OGNI
SEGMENTO DELLO SPAZIO
VIRTUALE

PER RIDURRE IL FENOMENO DELLA FRAGMENTAZIONE \Rightarrow MMU PER COMPATTARE I PROCESSI.

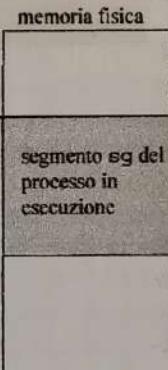
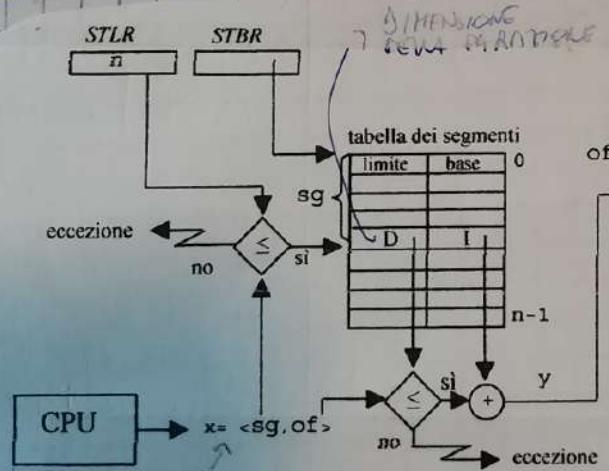
PER GARANTIRE LA CORRETTA RICOLAZIONE DEGLI INDIRIZZI, PER OGNI IND. VIRTUALE \times GENERATO DALLA CPU BISOGNA CAPIRE A CHE SEGMENTO APPARTENGA, MA E' IMPOSSIBILE.

(TUTTI GLI INDIRIZZI GENERATI IN RAME DI PIU' POSSONO APPARTENERE AL SEGMENTO CODICE. INDIRIZZI GENERATI IN RAME DI ESECUZIONE ~~APPARTENGONO~~ DI PUSH E POP \Rightarrow STACK).

RIDUZIONE DELLA FRAGMENTAZIONE ANCHE SE E' SEMPRE PRESENTE (LA DEN. FISICA VIENE CONSECUTIVAMENTE ALLOCATA PER PARTIZIONI). COMPATTARE E' POSSIBILE SENZA PUCITA' NHO SWAP.

INOLTRE, LO SPAZIO VIRTUALE PUO' ESSERE SEGMENTATO IN PIU' DI 3 PARTI, MA $X = \langle sg, of \rangle$

LA MMU NON MANTERRA' PIU' LE INFO NEI REGISTRI, MA IL SISTEMA OPERAVO MANTERRA' UNA "TABELLA DEI SEGMENTI". ORA IL DES-PROC CONTIENE ANCHE IL NUMERO DI SEGMENTI DELLA PROPRIA DEN. VIRTUALE E UN PUNTATORE ALLA TAB DEL SEGMENTI. PERO' PER OGNI INDIRIZZO GENERATO DALLA CPU ORA BISOGNA FARLE 2 ACCESI IN MEMORIA \Rightarrow TLB



NUOVA MMU
RIFANNOGLIO $n(f)$
REGISTRI ASSOCIAVII
DEGLI UNTAGLI $n(f)$
ACCESI IN MEMORIA.

SE LI BEGO LI
BENE, ALTRE NENA
MI SACCO 2
ACCESI IN MEMORIA

TLB SVUOTANO IN
CAMBIO DI PROCESSO

PROTEZIONE:

1. SG A CUI APPARTIENE X < STLR
2. OF S AUA DIN DEL SEGMENTO (NON NECESSARIAMENTE UN ERRORE, MA SENSO SE IL SEGMENTO MANTIENE STRUTTURE DATI)
3. AUA TAB DEL SEGMENTO VENGONO AGGIUNTI BIT DI CONTROLLO (R, W).

- SEGMENTAZIONE A DOMANDA

E' POSSIBILE CONOSCERE LA FUNZIONE DI RILOCAZIONE $y = f(x)$ IN MODO CHE DURANTE L'ESECUZIONE DI UN PROCESSO, IN OGNI Istante, sia possibile mantenere in memoria solo UNA PARTE DEL SUO SPAZIO VIRNALE E SE SI ACCERTE A PARTE NON ANCORA IN MEMORIA \Rightarrow SEGMENT-FAULT.

IL DESCRITTORE DI SEGMENTO DEVE ESSERE CORRUCCATO.
BIT P: PRESENZA O NANO DEL SEGMENTO IN MEM. PRINC.

OGNI TRADUZIONE DI $x = \langle sg, of \rangle$ CON $p=0 \Rightarrow$ SEG FAULT

E VIENE FATTO SWAP-IN.

PER FACILITARE LA CREAZIONE DI ALGORITMI DI RIMPIAZZAMENTO EFFICIENTI VENGONO AGGIUNTI AL DESCRITTORE DI SEGMENTO I BIT U (USED) E M (MODIFIED).

SERVE A ENTRARE A
PAGINA DI LAVORO

SE IL SEGMENTO NON E'
STATO MODIFICATO NON OCCORRE
CHE FACCIA SWAP-OUT, E' GA' IN
MEMORIA.

- MEMORIA PAGINATA

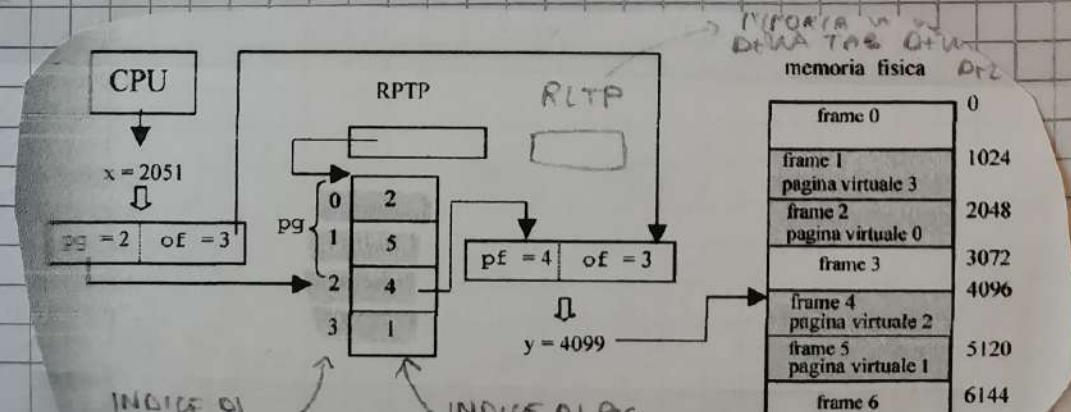
ALLOCAZIONE IN MEMORIA DI INFORMAZIONI I CUI INDIRIZZI VIRNALI SONO CONCERNI IN LOCAZIONI FISICHE NON NECESSARIAMENTE CONIGUE.

PER REALIZZARE LA FUNZIONE DI TRADUZIONE DEGLI INDIRIZZI E' SUFFICIENTE UNA TABELLA CON LE CORRISPONDENZE TRA PAGINE VIRNALI E FISICHE.

$$x / d_{\text{PAGINA}} = \text{INDICE DELLA PAG. VIRNIALE} \quad x \% d_{\text{PAGINA}} = \text{OFFSET}$$

CUI X APPARTIENE

PER OGNI PROCESSO VIENE MANTENUTA AGGIORNATA LA TAB. DELLE PAGINE CHE GLI APPARTENGONO.



RPTP E' IL REGISTRO DELLA CPU CHE MANTIENE L'INDIRIZZO DELLA TAB. DELLE PAGINE DEL PROCESSO IN ESECUZIONE, CHE DEVE ESSERE ALOCATA IN MEMORIA FISICA.
 PER RIDURRE IL NUM. DI ACCESSI \rightarrow TLB
 IL GESTORE DELLA MEMORIA FISICA MANTIENE AGGIORNATO L'ELENCO DELLE PAGINE FISICHE DISPONIBILI IN UNA PROPRIA STRUTTURA DATI ("TABELLA DELLE PAGINE FISICHE"). OGNI ELEMENTO DELLA TAB CONTIENE L'INDICAZIONE SE LA PAGINA FISICA E' LIBERA O OCCUPATA, E NEL SECONDO CASO L'INDICE DEL PROCESSO CUI E' ALOCATA.
 QUANDO UN PROCESSO DEVE ESSERE CARICATO IN MEMORIA GUARDA LA TAB DELLE PAGINE FISICHE. SE NON CI SONO ABBASTANZA LIBERE, VIENE FATTO SWAP-OUT DI UN ALTRO PROCESSO. (SONO QUINTO NECESSARI 1-2 STAM SWAPPED)
 IL DES-PROC CONTIENE L'INDIRIZZO IN MEMORIA DELLA TAB DELLE PAGINE E IL NUMERO DI PAGINE CHE POSSEDDE.

DIMENSIONE DELLE PAGINE: SE DIMINUISCO MOCHIO LA DIM DELLE PAGINE LE STRUTTURE DATI AUMENTANO MOCHIO DI DIMENSIONE MA CI DOPO IL PROBLEMA DELLA FRAGMENTAZIONE INTERNA (TIPLICAMENTE, IN MEDIA L'ULTIMA PAGINA E' PIENA A META')
 SE AUMENTO TROPPO LA DIM DELLA PAGINA LA DIM DELLE STRUTTURE DIMINUISCE MA AUMENTA LA FRAGMENTAZIONE INTERNA. (TIPLICAMENTE DA 512 byte A 4 Kbyte)

PROTEZIONE: 3 CONTROLLI

1. SE $Pg > n$ STO ACCEDENDO A UNA PAGINA NON MA
2. SE $OF > \text{dim_pag}$ STO ANDANDO FUORI PAGINA
3. CAMPO CONTROLLO NELLA TAB DELLE PAGINE: R, W.
 HA MENO SENSO CHE NELLA SEGMENTAZIONE PERCITE' COLA POTRA' AVERE AD ES. UNA PAG. META' DATI E META' CODICE.

- PAGINAZIONE A DOMANDA

LO SPAZIO VIRLENTE DI UN PROCESSO ALLA SUA CREAZIONE RISIENE COMPLETAMENTE NEGLI SPATI DI MASSA. VENGONO CARICATE IN RAM SOLO LE PARTI CHE SERVONO IN QUEL DETERMINATO ISTANTE.

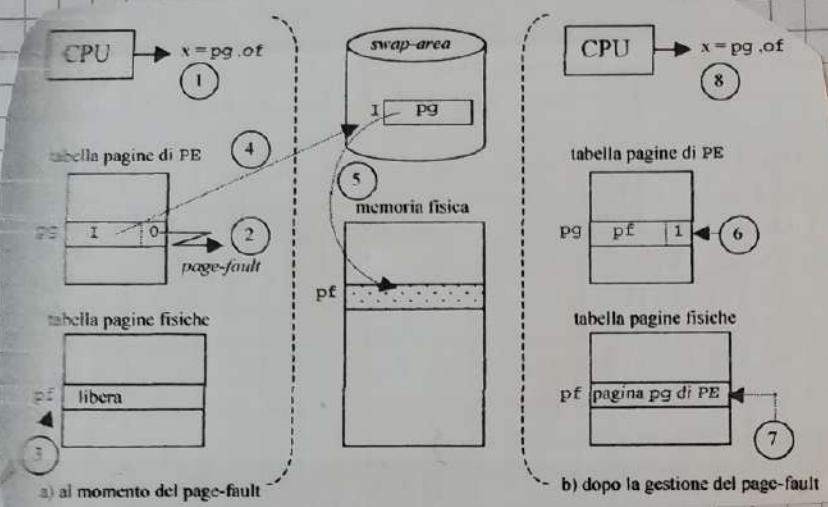
campo pagina fisica	campo controllo
indice della pagina fisica se $P=1$ indirizzo su disco se $P=0$	R W U M P

CADE L'ESIGENZA DI AVERE 1-2 STAM SWAPPED. A OGNI RIFERIMENTO RISPETTO A UNA PAGINA NON IN MEMORIA ($P=0$) VIENE LANCIATO UN PAGE-Fault RISOLTO IN DMA.
 2 CASI DI INTERESSE: SE CI SONO PAGINE LIBERE NELLA TAB. DELLE PAGINE FISICHE OPPURE NO.

+ L'ALGORITMO DMRU E' BASEATO SUL CONTO DI REFERENZE A PAGINE LIBERE.
 O UNA PAGINA NON VERRÀ MOLTIPLICATA PIÙ UN'UNIVOCITÀ, O CHE SARÀ QUESO AD ESSERE USATA TUTTO PIÙ TARDI.

1. C'È UNA PAG. LIBERA IN RAM

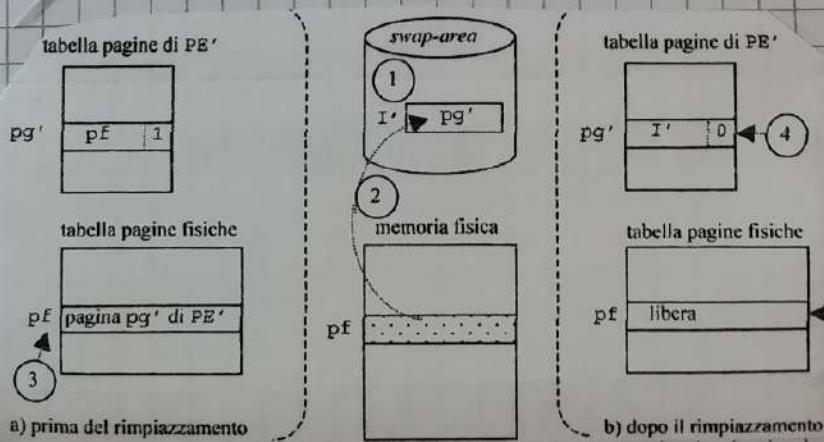
IREGISTRAZIONE



- ① LA CPU GENERA $X = \langle pg, of \rangle$
- ② LA MMU GUARDA LA TAB. DELLE PAGINE, VERDE $P=0 \Rightarrow$ PAGE-Fault
- ③ VADO A LIV-SISTEMA, RIAVO L'INDIRIZZO PF DI UNA PAGINA LIBERA DA LA TAB. DELLE PAGINE FISICHE
- ⑤ DMA PER PORTARE DATA MEN DI MASSA LA PAG IN RAM ALL'INDIRIZZO PF ④ RICAVANDO L'INDIRIZZO DELLA PAG. FISICA IN DEN DI MASSA DI I NUOVA TAB. DELLE PAGINE
- (6) CAMBIO DI CONTESTO, ENTRA UN NUOVO PROCESSO.
- AL TERMINE DEL DMA, VENGONO AGGIORNATE LA TAB. DELLE PAGINE SETTANDO $P=1$ E L'INDIRIZZO PF,
- ⑦ LA TAB. DELLE PAGINE FISICHE INDICANDO CHE PF E' OCCUPATA
- ⑧ VIENE RIAMMOSCATO P RACORDOGLI RIESGUIRE L'ISTRUZIONE CHE HA SCARICATO PAGE FAULT ($IP = IP + 1$)

E' POSSIBILE CHE ARRIVATO AL PUNTO DI CONFERIRE PROCESSO ABbia RI-RIMPIAZZATO LA PAGINA CHE HO APPENA CARICATO. IN QUESTO BISOGNA PRENDERE IL LOCK SULLA RISORSA DELLA PAGINA PER EVITARE CASINI.

2. NO PAG LIBERA IN RAM



BISOGNA EFFETTUARE IL RINPIAZZAMENTO DI CONCETTO INTERA PAGINA PER RICARCARNE UNA LIBERA.
 (PER ACCORGERSI CHE NON CI SONO PAGINE LIBERE DOVRE SCOPRIRE NELLA LISTA DEI VISTI DENTRO TAB DELLE PAGINE FISICHE, O(n)).

- ① CERCO NELLA SWAP AREA UNA PAGINA LIBERA
- ② VIENE TRASFERITO IL CONTENUTO DI PF NELLA PAGINA LIBERA DELLO SWAP
- ③ RIUOVO L'INDICE DEL PROCESSO P WIL STO TOGLIENDO LA PAGINA PF ACCEDENDO ALLA TABELLA DELLE PAGINE FISICHE.
- ④ METTO A 0 IL BIT P NELL'INDICE DELLA TABella DELLE PAGINE DI PF. E SCRIVO NELL'CAMPO INDIZIO L'INDIRIZZO DELLO SWAP IN QUI E' STATA DESA LA PAGINA.
- ⑤ SCRIVO NELL'INDICE DELLA TABella DELLE PAGINE FISICHE CHE ORA PF E' LIBERA

ALGORITMO DI RINPIAZZAMENTO DELLE PAGINE *

LRU = "LAST RECENTLY USED" EFFETTUATO DENTRO A SEGLIE CONE PAGINA DA RINPIAZZARE CON UNA NUOVA RECENTEMENTE UTILIZZATA. E' IRREALIZZABILE, OCCORREVEREBBE INSERIRE UN TIESTAMP A 32 b-t NELL'INDICE DELLA TABella DELLE PAGINE FISICHE A OGNI CAMPO.

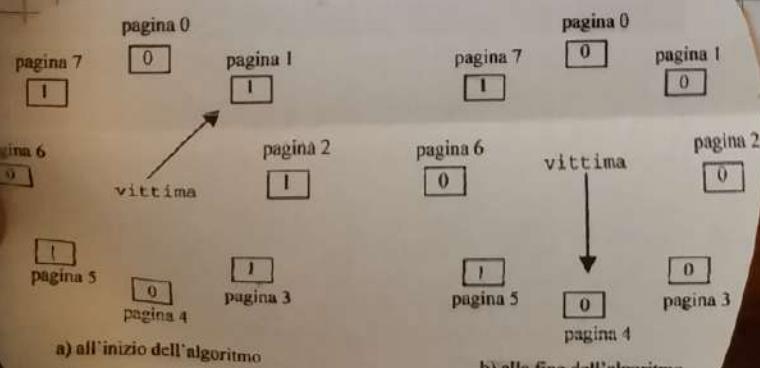
FIFO = ALGORITMO PIU' SENPUCE IN ARDORNO, MA POCO FUNZIONALE. VIENE RINPIAZZATA LA PAGINA CHE DA PIU' TEMPO E' IN MEMORIA ORGANIZZANDO LA TABella DELLE PAGINE FISICHE CONE VENORE CIRCOLARE.

"SECONDO CHANCE" O "CLOCK ALGORITHM" = SI SCEGLIE CON TECNICA FIFO LA PAGINA DA RINPIAZZARE CON BIT U=0. TUTTE quelle che incontrano con U=1 LE METTO CON U=0. DOPO UN GIRO INVERSO SICURAMENTE TROVERO' LA PAGINA DA RINPIAZZARE (SI COMPORTA CONE FIFO MA UN PEGGIORE). SPESO VIENE USATO ANCHE IL BIT DI MUODUTA M. PRIMA GUARDO U-M = 0,0, Poi U-N = 0,1, Poi SICURAMENTE TROVERO' LA PAGINA M PIU' GIO.

RINPIAZZAMENTO LOCALE

LA PAG DA RINPIAZZARE VIENE CERCATA TRA quelle che APPARTENGONO GAI AL PROCESSO

INOLTRE POSSANO PREVEDERE CHE OGNI PROCESSO PREARREDI UN CERTO NUMERO DI PAGINE IN MEMORIA PRIMA DI PARTIRE.



SEGMENTAZIONE CON PAGINAZIONE

- RICARICAZIONE: DINAMICA
- ALLOCAZIONE: NON CONTIGUA
- SPAZIO VIRNNALE: SEGMENTATO
- CARICAZIONE: A DOMANDA

$$X = \langle sg, sc \rangle$$

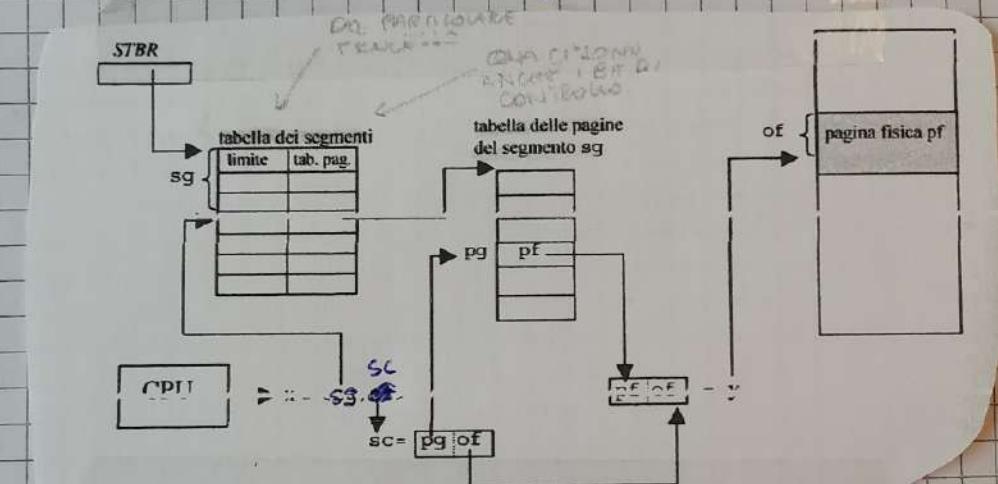
↓

$$\langle pg, of \rangle$$

- LA CPU GENERA
 $X = \langle sg, of \rangle$. IL PUNTO E' CHE
 ORA NON POSSO PIU' FARRE
 COSE PRIMA SG = X / DIM PAGINA
 E ACCEDERE A QVELLA PAGINA
 SPECIFICA, PERCHETE ADesso
 LO SPAZIO VIRNNALE E'
 SEGMENTATO.

PER OGNI SEGMENTO DEVO ANDARE
 UNA TAB PEGHE PAGINE DI QVEL
 SEGMENTO.

- OGNI INDIRIZZO VIRNNALE PUO'
 QUINDI GENERARE PIU'
 INTERRUZIONI: LA TABELLA DELLE
 PAGINE DEL SEGMENTO PUO'
 NON ESSERE IN MEMORIA, E'
 DOPO LA PAGINA STESSA PUO'
 NON ESSERE IN MEMORIA
 (SEGMENT-FAULT E PAGE-FAULT).



- GESTIONE DEGLI SPAZI VIRNNALE.

CON LA RICARICAZIONE DINAMICA (MMU) ABBIANO 2 PROBLEMI:

1. SE LO SPAZIO VIRNNALE CONTIENE DEGLI INDIRIZZI, QUESTI
 NON VENGONO RILOCATI.

2. A QUALE SPAZIO VIRNNALE APPARTENGONO LE SYSTEM CPU?
 QUANDO ENTRA IN ESECUZIONE UNA PRIMAVERA DI SISTEMA, LA
 CPU INIZIA A GENERARE INDIRIZZI RIVERNANTI A ISTRUZIONI E DATI
 DEL SISTEMA OPERATIVO.

1° SOLUZIONE:

IL SISTEMA OPERATIVO HA UN PROPRIO SPAZIO VIRNNALE. 2 INCONVENIENTI:

1. Oltre a cambiare lo stato di esecuzione da utente a
 sistema, ora e' anche necessario cambiare la funzione
 di rilocazione

2. Ad un eventuale chiamata del tipo `n-read` (dispositivo, buffer, nbytes)
 l'indirizzo del buffer e' un indirizzo virnnale del processo.

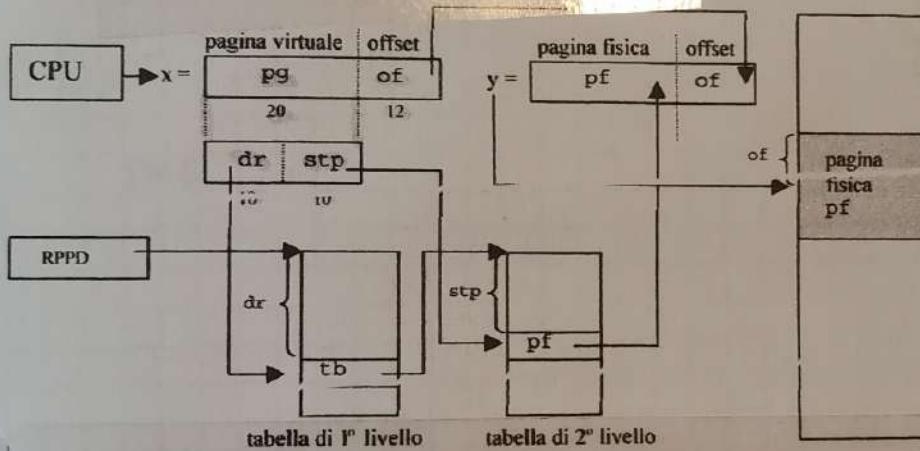
IL SISTEMA NON PUO' TRADURRE CON LA PROPERA FUNZIONE DI TRADUZIONE,
MA CON CERTELLA DEL PROCESSO.

2° SOLUZIONE:

IL SISTEMA OPERA CON UNA PARTE DELLO SPAZIO DI INDIRIZZAMENTO DI OGNI PROCESSO (SE HO 32 bit, SPAZI VIRTUALI DI 4 Gbyte, CHE QUINDI PERME MONO LO STORAGE DEL PROCESSO E DEL SO). IL SISTEMA OPERA CON UNA PARTE CONDIVISA TRA TUTTI I PROCESSI.
=> LA FUNZIONE DI TRADUZIONE NON DEVE ESSERE CAMBIATA ALLA MIGRAZIONE DI UNA CHIAMATA DI SISTEMA. POICHE' GLI INDIRIZZI VIRTUALI DEL SO SONO PARTE DELLO SPAZIO VIRTUALI DEL PROCESSO.

NUOVO PROBLEMA: LA TABEWA DEI PAGINE DIVENTA TROPPO GRANDE (SUPponendo 4 byte A ENTRATA, PAGINE DA 4 kbyte, HO 4 MEGabyte PER TABEWA)

=> PAGINAZIONE A PIÙ LIVELLI



NEI SISTEMI SEGMENTATI LA TABEWA DEI SEGMENTI VIENE DIVISI IN 2 SOTTOTABEWI: SISTEMA E PROCESSO

NON VIENE
COMUNITATA OVANDO
SI PASSA DA UN
PROCESSO ALL'ALTRO

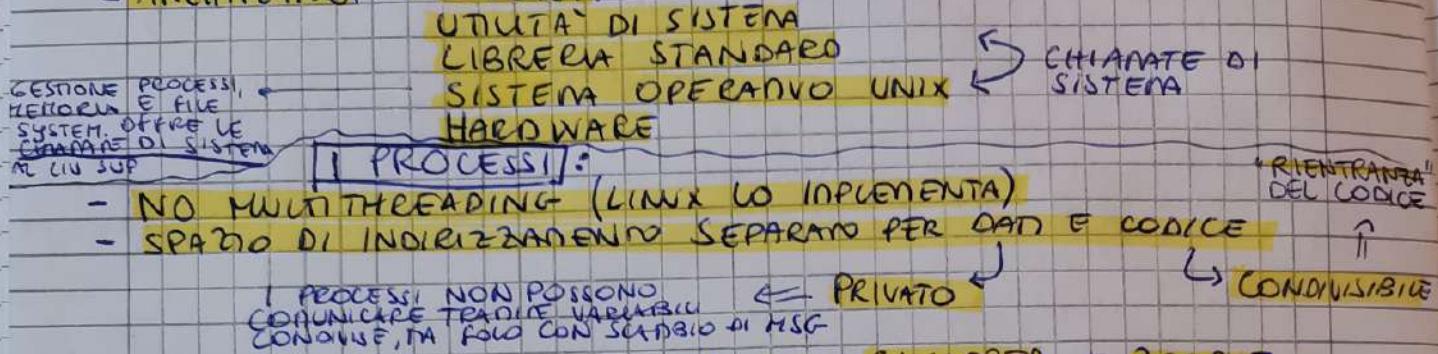
ALTRA PROBLEMA D'UNA RIDAZIONE DINAMICA: LA SCHEDUOLAZIONE DI UN PROCESSO A UN ALTRO E' COSTOSA, PERCIOME' E' NECESSARIO CAMBIARE LA FUNZIONE DI TRADUZIONE DELLA MMU E INVALIDARE QUINDI IL TLB.

ARCHITETTURA DI UNIX

UNIX E' UN SISTEMA MULTIUTENTE E MULTIPROGRAMMATO.

- GESTIONE DELLA MEMORIA: PAGINAZIONE E SEGMENTAZIONE (A DOMANDA)

- ARCHITETTURA:



- DIAGRAMMA DEGLI STATI CON ANCHE SWAPPED E ZOMBIE

PREFERIBILITÀ
SWAP-IN / OUT DEI
PROCESSI DI DIMENSIONE
MAGGIORE.

NONOSTANTE IL
CARICAVENTO SIA A
DOMANDA

SE UN FIGLIO
TERMINA PRIMA
DEL PADRE NA
CESSA NON HA ANCORA
LEMO IL RISULTATO.

- IMMAGINE DI UN PROCESSO UNIX:

PROCESS STRUCTURE

- PID
- STATO DEL PROCESSO
- RIFERIMENTO AREE DATI/STACK
- RIFERIMENTO AL CODICE
- PID DEL PADRE
- PRIORITÀ
- RIFERIMENTO AL PROPS
PROCESSO IN CODA
- PUNTATORE ALLA USER STRUCT

USER STRUCTURE

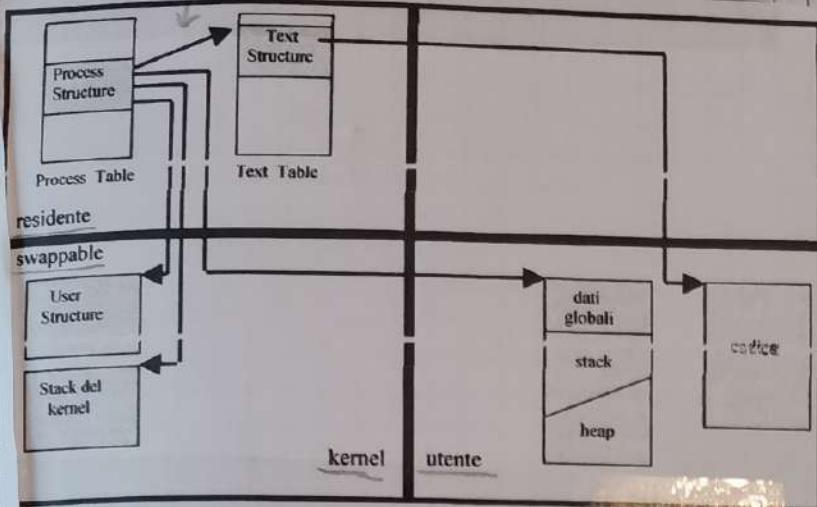
- COPIA REG CPU
- INFO RISORSE ATTUALI
(ES. FILE APERTI)
- INFO SULLA GESTIONE
DI EVENTI MINICRONI
- DIRETTOREIO CORRENTE
- UTENTE PROPRIETARIO
- GRUPPO

2 STRUTTURE DATI
DISTINTE: PROCESS
STRUCTURE E USER
STRUCTURE.

NAME OF PROCESS
STRUCTURE SONO CONTENUTE
NELL'PROCESS TABLE
CHE E' UNA
STRUTTURA DATI DEL
KERNEL.

PER PERMETTERE A PIU' PROCESSI DI USARE LO STESSO CODICE
=> TEXT TABLE. OGNI ELEMENTO DI UNA TEXT TABLE CONTIENE UN
PUNTATORE ALL'AREA DI MEMORIA IN CUI E' AVVOLTO IL CODICE

NEL DESCRIZIONE DEL PROCESSO C'E' UN RIFERIMENTO ALL'ELEMENTO
DELLA TEXT TABLE DEL CODICE DA ESEGUIRE.



NON TUTTE LE PARTI DEL MIGRANTE DI UN PROCESSO POSSONO ESSERE SOTTOPOSTE A SWAPPING.

PROCESS STRUCTURE E TEXT STRUCTURE DEVONO ESSERE SEMPRE RESIDENTI IN MEMORIA.

- SCHEDULING: BASATO SU ROUND-ROBIN CON PRIORITA'

PROCESSI CON LA STESSA PRIORITA' SONO COLLOCATI NELLE STESE CODE, ESEGUITI CON TECNICA ROUND-ROBIN

VALORI NEGANVI: SISTEMA
VALORI POSITIVI: USER

SONO DINAMICI E RICALCOLATI DI TANNO IN TANNO.

- GESTIONE DELLA MEMORIA

3 SEGMENTI PER OGNI PROCESSO:

CODE SEGMENT
STACK SEGMENT
DATA SEGMENT

- ALLOCAZIONE DEI SEGMENTI GESTITA CON PAGINAZIONE A DOMANDA

→ PROCESSO "PAGE DAEMON" ESEGUITE PERIODICAMENTE SOSPIRA LE PAGINE.

VALORE SECONDO CHANCE

ESEGUE SE IL NUMERO DI PAGINE LIBERE < LOTS FREE

⇒ SE CONUNQUE PAGE DAEMON ESEGUE TROPPO FREQUENTEMENTE E IL NUMERO DI PAGINE RIMANE CONUNQUE INTERO A LOTS FREE → SWAPPER

LOTS FREE > DESFREE > MINFREE

VALORE DI DESFREE DI PAGE DAEMON

NUMERO MEDIO DI PAGINE LIBERE

MINIMO PER EVITARE LO SWAPPER.

TRASFERIRE PROCESSI INTERI DA RETI PRINCIPALI A RETI SECONDARIE.

SWAPPER CHIAMATO SOLO SE:

- 1- NUM PAGINE LIBERE < MIN FREE

2- NUMERO MEDIO DI PAGINE LIBERE NEW UNITA' DI TEMPO < DESFREE.

dal punto di vista macroscopico, le attività svolte da un calcolatore possono essere suddivise in attività di ingresso/uscita (I/O) e attività di elaborazione svolte dalla CPU.

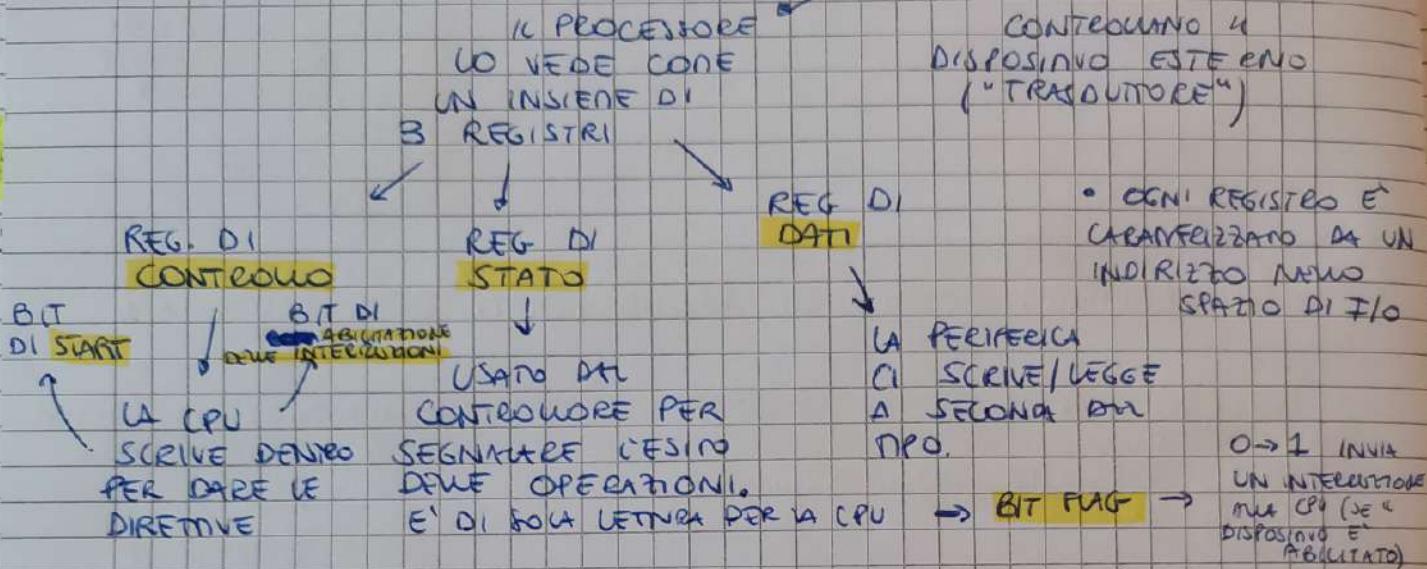
PROCESSI
CPU BOUND

PROCESSI I/O BOUND

GESTIONE DELLE PERIFERICHE (I/O)

dal punto di vista
ARCHITETTURALE...

- LE PERIFERICHE SONO COLLEGATE AL BUS TRAMITE CONTEOLATORI



- L'INSIENE DI NOME LE FUNZIONI DI ACCESSO ALLE PERIFERICHE COSTRUISCE LE API (APPLICATION PROGRAMMING INTERFACE)

- LE VELOCITÀ CON CUI I DISPOSITIVI TRASMETTONO O RICEVONO DATI SONO NUOVE DIVERSE TRA DI LORO.

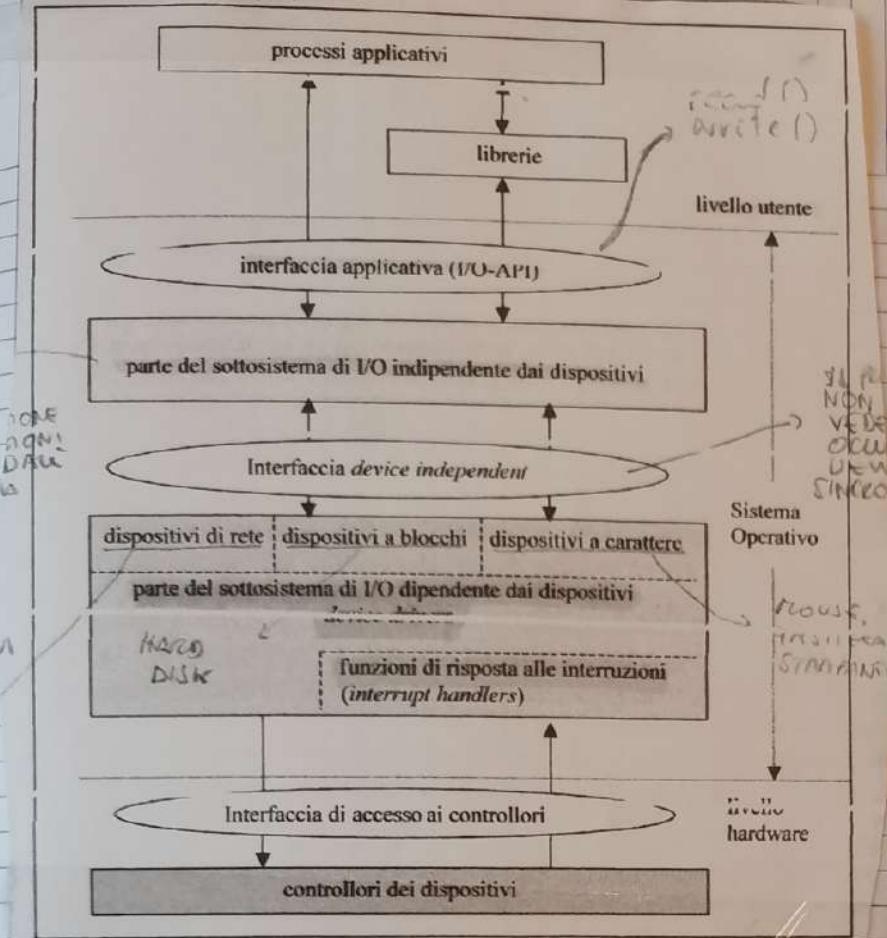
- CONCETTI DEL SOTOSISTEMA DI I/O

① DEFINIZIONE DI UNO SPAZIO DI NONI TRAMITE I quali i DISPOSITIVI VENGONO IDENTIFICATI DAI PROCESSI. ("NAMING") (I PROCESSI IDENTIFICANO LE PERIFERICHE TRAMITE NOMI, DENTRO A LIVELLO SISTEMA TRAMITE INTERI)

② MALFUNZIONAMENTI: GESTIONE DEI MALFUNZIONAMENTI, IN ALCUNI USI RIDONABILI AL SISTEMA, IN ATTEI NO (ES. CARTA PINTA) (INTERRUZIONE DEL PROCESSO E MSG IN CONTROL)

③ SINCRONIZZAZIONE: I DISPOSITIVI I/O NORMAMENTE SONO PIÙ LENTI DELLA CPU. IL PROCESSO VA BLOCCATO E RISVEGLIA ALLA FINE. FANNO PARTE DEL SISTEMA ANCHE NUOVE FUNZIONI DI RISPOSTA ALLE INTERRUZIONI GENERATE DAI DISPOSITIVI.

④ BUFFERIZZAZIONE: PER I DISPOSITIVI A CARICA (TASTIERA) SERIE PERCHÉ LA CPU È NUOVO PIÙ VELOCE. PER I DISPOSITIVI A BLOCCI SERIE PERCHÉ POSSONO AVERE BLOCCO SOLO DI UNA PICCOLA PARTE DEL BLOCCO, MA TUTTI I DISPOSITIVI TRASFERISCONO SOLO BLOCCI INTEI.



* IL COPPIA PER SOTTOSISTEMA DI I/O DEL SISTEMA OPERAVANO E' CALCOLO DI FORNIRE UN INTERFACCIA UNIFORME CHE MIGLIORI LE SINGOLE PECULIARITA' DEI DISPOSITIVI E NE GARantisca L'EFFICIENZA DI ACCESSO

L'OBBIENO E' QUINDI DI FORNIRE UN INSIENE DI FUNZIONI GENERICHE

read, write
open, close

• SOTTOSISTEMA DEVICE INDEPENDENT

- FORNISCE L'INSIENE DI NIVELE CHIAMATE DI SISTEMA DEDICATE AL TRASFERIMENTO DI I/O E DA UN'AUTRA PARTE SI INTERFACCIA CON LA COMPONENTE DEVICE DEPENDENT.

- RENDE PIU' SENZIALE, SICURO ED EFFICIENTE L'USO DEL DISPOSITIVO.

- SI OCCUPA DI:

- 1- NAMING DI FILE E DI DISPOSITIVI
- 2- PROTEZIONE
- 3- BUFFERING
- 4- ALLOCAZIONE DINAMICA DEI DISPOSITIVI E SPOOLING
- 5- GESTIONE DEGLI ERRORI

SE UN PROCESSO CHIAMA $n = \text{read}(Fd, ubuf, nbytes)$

• VIENE USATO UN BUFFER DI SISTEMA COME TRAMITE

E PERCHÉ POTREI LEGGERE E ELABORARE UNA SEQUENZA DI BLOCCI INVECE CHE UNO UNICO

INDIRIZZO DEL BUFFER IN MEMORIA VIRTUALE DEL PROCESSO

⇒ L'ELABORAZIONE DI UN BLOCCO DI DATI DA PARTE DEL PROCESSO VIENE ESEGUITA IN PARALLELO CON IL TRASFERIMENTO DEL BLOCCO SUCCESSIVO NEL BUFFER DI SISTEMA.

- PER LA GESTIONE DEGLI ERECOLI, IN GENERALE SI CERCA DI RITROVARE AL LIVELLO IN CUI SONO STAN RICHIESTI O PROPAGARLI AL LIVELLO SUPERIORE.
AL LIVELLO DEVICE INDEPENDENT POSSONO ESSERE PRESE DECISIONI COME RENDIMENTARE UNA STAMPA.
- ALLOCAZIONE DINAMICA: ALCUNI DISPOSITIVI POSSONO ESSERE ALLOCATI SOLO A UN PROCESSO ALLA VOLTA \Rightarrow IL SISTEMA DEVICE INDEPENDENT PROVVEDE ALLA DURATA ESISTENZA.

"SPREADING": I PROCESSI OPERANO SU DUE COPIE VIRTUALI DEI DISPOSITIVI. VENGONO CREATI DEI FILE SU DISCO CHE Poi VERRANNO RECUPERATI DA UN PROCESSO "DAEMON" CHE E' L'UNICO AD AVERE ACCESSO ALLA STAMPANTE REALE

• SOTTOSISTEMA DEVICE DEPENDENT

- NASCONDE AL SUO INTERNO I DETTAGLI RELATIVI AI SINGOLI DISPOSITIVI DEFINENDO L'INTERFACCIA "DEVICE INDEPENDENT" CHE PERMETTE AL LIVELLI SUPERIORI DI ASTRAERSI DA TUTTI I DETTAGLI.
- OGNI DISPOSITIVO FISICO VIENE RAPPRESENTATO TRAMITE UNA STRUTTURA DATI. LE FUNZIONI OFFERTE SONO PRATICAMENTE LE STESE OFFERTE AL LIVELLO DI INTERFACCIA APPLICATIVA:

n-read (dispositivo, buffer, nbytes)
n-write (dispositivo, buffer, nbytes)

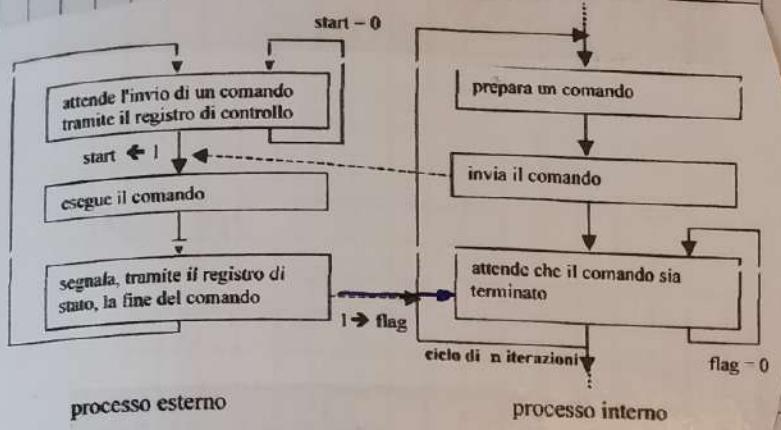
VENGONO REALIZZATE ANDANDO A OPERARE DIRETTAMENTE SUI REGISTRI DEL CORRISPONDENTE CONTROLLORE TRAMITE L'INTERFACCIA DI BASELIVELLO.

E' IL BUFFER DI SISTEMA, NON L'AREA USBUF DEL PROCESSO APPLICATIVO.

- NOTA: LE OPERAZIONI SVOLTE DAL DISPOSITIVO SONO CONCORSUALE E ASINCRONE CON LE OPERAZIONI SVOLTE DAL PROCESSORE. L'INSIEME DEL DISPOSITIVO COL SUO CONTROLLORE PUO' ESSERE VISTO COME UN PROCESSORE SPECIALE PURPOSE.

PROCESSO ESTERNO

OGNI DISPOSITIVO ESEGUE UNA SEQUENZA DI AZIONI CHE POSSANO CONSIDERARE COME L'ESECUZIONE DI UN PROCESSO



3 METODI DI GESTIONE DEL DISPOSITIVO:

TRANSFERIMENTO DI DATI CONSECUTIVAMENTE:
CON LA GESTIONE A INTERRUZIONE,
SE IL PROCESSO APPlicativo
DEVE TRASFERIRE N DATI, AVRA' N CAMBI
DI CONTESTO.

CONTROLLO DI PROGRAMMA: LA CPU
CICLA SUL FLAG: CICLO DI ATTESA A MVA.
E' IL PROCESSO APPlicativo A
CONTROLLARE, IN OGNI CICLO, CIF CI
SARANNO DATI DISPONIBILI NEL REG. DATI.

GESTIONE A INTERRUZIONE: IL PROCESSO
APPlicativo SI SOSPENDE SU UN SEMAFORO
FINO A CHE IL CONTROLLORE METTE
IL BIT FLAG A 1, INVIANDO UN'INTERRUZIONE
CHE RIUSVEGLIA IL PROCESSO.
LA FUNZIONE CHE C'È L'INTERRUZIONE ANDA IN
ESECUZIONE E' LA "FUNZIONE DI RISPOSTA ALL'
INTERRUZIONI DEL DISPOSITIVO" ED ESEGUE LA
SIGNIFICA SUL PROCESSO.

SAREBBE CONORDO CHE IL PROCESSO SI BLOCCHE UNA VOLTA FINO
A CHE IL CONTROLLORE C'È IL TRASFERIMENTO NON E' COMPLETATO.

DESCRITTORE DI UN DISPOSITIVO

- INDIRIZZO REGISTRO DI CONTROLLO
 - INDIRIZZO REG STATO
 - INDIRIZZO REG DATI
 - SEMAFORO DI SINCHRONIZZAZIONE: dato disponibile
 - CONTATORE DATI DA TRASFERIRE: Contatore
 - INDIRIZZO DEL BUFFER (DI SISTEMA) buffer
 - RISULTATO DEL TRASFERIMENTO esito.
- 1- NASCONDE AL PROPRIO INTERNO TUTTE LE INFO DEL DISPOSITIVO
 - 2- CONSENTE LA COMUNICAZIONE TRA PROCESSO APPlicativo E DISPOSITIVO.

DEVICE-DRIVER

```
int read (int disp, char * pbuf, int cont)
{
    descrittore[disp].contatore = cont;
    descrittore[disp].puntatore = pbuf;
    <attivazione del dispositivo>; // bit di start=1
    // sospensione del processo
    descrittore[disp].dato_disponibile.wait();
    // in caso di errore restituisce -1
    if (descrittore[disp].esito == <codice di errore>)
        return(-1);
    // altrimenti restituisce il numero di dati letti
    return(cont - descrittore[disp].contatore);
}
```

BUFFER
DI SISTEMA

+ attivazione
PE

READ
DEVICE
DEPENDENT

VENNE
PRODOTTO

E' IL NUMERO
DI BYTE LETTI

- void inth()

"FUNZIONE DI RISPOSTA ALLE
INTERRUZIONI DEL DISPOSITIVO"

(char b)

<legge il reg. di stato del controllore>

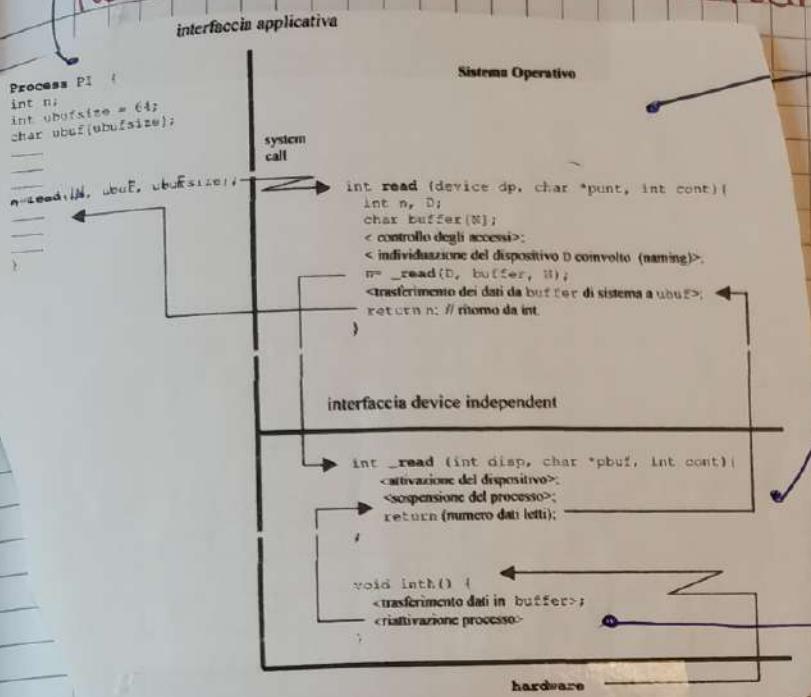
if (bit_ERRORE == 0) { //nessuna di errori

```
b = <lettura del registro dati>
<assegnamento alla variabile locale b>;
*descrittore[disp].puntatore = b;
descrittore[disp].puntatore++;
descrittore[disp].contatore--;
if (descrittore[disp].contatore != 0)
    <riattivazione dispositivo>;
else {
    descrittore[disp].esito = <terminazione corretta>;
    <disattivazione del dispositivo>;
    // riattivazione processo
    descrittore[disp].dato_disponibile.signal();
}
else { // presenza di errori
    <routine di gestione errore>;
    if (<errore non recuperabile>)
        descrittore[disp].esito = <codice errore>;
    // riattivazione processo
    descrittore[disp].dato_disponibile.signal();
}
}
return; // ritorno da interruzione
}
```

IN
OP
C'E'
SIGN
PRO

È IL PROCESSO APPUTRIVO CHE CHIAMA LA READ CON IL NOME SIMBOLICO DEL DISPOSITIVO, IL BUFFER UTENTE E IL NUM DI BYTE DA LEGGERE

FUSSO DI CONTROLLO DURANTE UN TRASFERIMENTO



CONTROLLO: SISTEMA OPERATIVO
MA LO SPAZIO DI INDIRIZZI
E' SEMPRE CELVIO PER
PROCESSO.

LIVELLO INDIPENDENTE DAL
DISPOSITIVO

E' LA READ DI
PIÙ BASSO LIVELLO, DEL
DRIVER, ATTRAVERSO IL
DISPOSITIVO E BLOCCO IL
PROCESSO APPUTRIVO
(DEV DIPENDENT)

LE INTERRUZIONI SONO
GESATE DA inth.
FINO A CHE TUONO NON
E' STATO TRAFFERITO NELL'
BUFFER NON SI TORNA
MA -read

TIMER

- UTILITA': ALGORITMI DI SCHEDULAZIONE COME RR, OPPURE MANUTENERE AGGIORNATA LA DATA DELLA MACCHINA.

• DESCRITTORE:

- INDIRIZZO REG CONTROLLO
- INDIRIZZO REG STATO
- INDIRIZZO REG CONTATORE
- Array DI SENATORI PER UN:
fine_attesa [N];
- Array DI INTERI:
ritardo [N];

ROUTINE DI RISPOSTA AGLI
INTERVENTI:

void inth () {

```

for (int i=0; i<N; i++) {
    if (descrittore.ritardo[i] != 0) {
        descrittore.ritardo[i] -= 1;
        if (descrittore.ritardo[i] == 0)
            descrittore.fine_attesa[i].signal();
    }
}

```

- SI SUPONE N = NUM PROCESI, TOH,
IN REALTA' NON E' COSÌ,
SAREBBE TROPPO GRANDE

PRIMAVERA delay:

void delay (int n){

```

int proc;
proc = <INDICE PROC IN EXEC>;
descrittore.ritardo[proc] = n;
descrittore.fine_attesa[proc].wait();
}

```

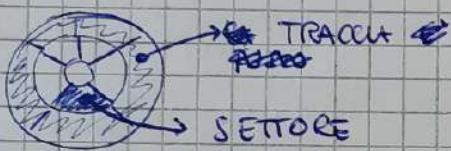
DISCO → PERIFERICA DI I/O

IMPORTANZA: PUO CHIUDERE IN SWAP IN/OUT DEI PROCESSI

- ORGANIZZAZIONE FISICA DEI DISCHI

TESTINE FISSE

TANTE TESTINE
QUANTE SONO
LE TRACCE



SETTORE: (F, T, S)

NUM SETTORE
NUM TRACCIA
NUM RACCA

- TEMPO MEDIO DI TRASFERIMENTO (TF)

$$TF = TA + TT \rightarrow \text{TEMPO PER VOLTEGGIARE} \quad \text{TEMPO DI ROTAZIONE}$$

TEMPO PER
SPOSTARE LA
TESTINA SUL
SEMORDE

SEEK TIME:
TEMPO PER SPOSTARE
LA TESTINA SULLA
TRACCIA

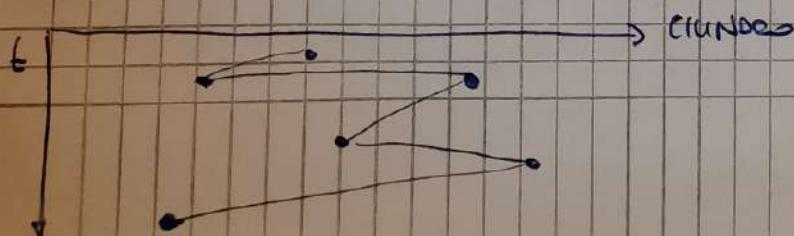
SENON POSSO
IN PIU:

$$620(5.2 + 3 + 0.019) = 5260$$

TEMPO PER
LEGGERE UN
SOLO SERVIZIO

METODI DI
SCHEDULING PER L'ACCESSO
AL DISCO:

- FCFS



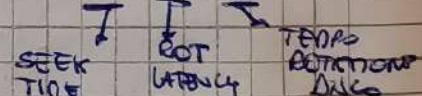
- IL NUMERO DI BIT MENDICITAN SO
UNA TRACCIA E' LO STESSO PER
TUTTE LE TRACCE
- IL TRASFERIMENTO DI DATI AVVIENE
PER SETTORI.

Ese-

FILE 320 kb
SETTORE: 512 b
⇒ 640 SETTORI
(IPOTESI: 2 TRACCIE
CONSECUTIVE)

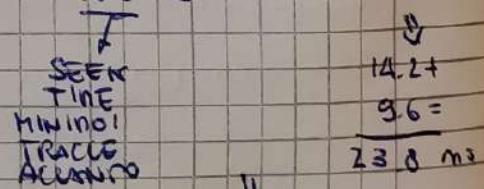
1° TRACCIA:

$$5.2 + 3 + 6 = 14.2$$



2° TRACCIA:

$$0.6 + 3 + 6 = 9.6$$



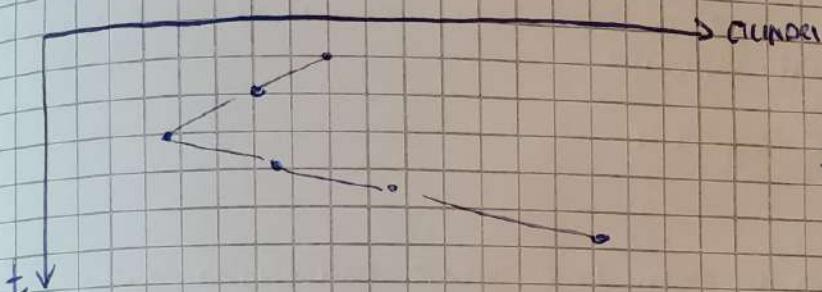
2 COSE IMPORTANTI:

- ① CRITERI CON CUI I DATI VENGONO RENDICIENI SU DISCO
- ② COME SCHEDULARE LE RICHIESTE DI LETTURA/SCRITTURA

- SSTF (SHORTEST SEEK TIME FIRST)



- SCAN



FILE SYSTEM

- UTILITÀ: È LA PARTE DI SISTEMA OPERATIVO CHE FORNISCE I MECANISMI NECESSARI PER L'ACCESO E L'ARCHIVIAZIONE DELLE INFORMAZIONI IN MEMORIA SECONDAUTA.

- STRUTTURA:

APPLICATIONS

- STRUTTURA LOGICA ①
(FILE E DIRECTORY)
- ACCESSO ②
(SEQUENZIALE, DIREMO, PROTEZIONE)
- ORGANIZZAZIONE FISICA ③
(ALLOCAZIONE DEI FILE NEI BLOCCHI FISICI)
- DISPOSITIVO VIRTUALE ④
(VETTORE LINEARE DEI BLOCCHI FISICI)

→ 4 COMPONENTI

HARDWARE

- ① STRUTTURA LOGICA

- IL FILE

- NOME (SIMBOLO, MEDIANTE IL QUALE CI SI PUÒ RIFERIRE AL FILE)
- TIPO (STABILISCE L'APPARTENENZA DEL FILE A UNA CLASSE)
- INDIRIZZI (PUNTATORI AI BLOCCHI PER L'ALLOCAZIONE IN MEMORIA)
- DIMENSIONE
- DATA E ORA DI CREAZIONE
- DATA E ORA DI ULTIMA MODIFICA
- PROPRIETARIO
- PROTEZIONE

SONO GLI ATTRIBUTI PIÙ RICORRENTI DEL FILE NEL S.O.

→ SOLO NEI S.O. MULTUTENTE

MAGGIORE
⇒ OVERHEAD:
OGNI VOLTA DEVO
SCORRERE LA
LISTA DI NUOVE
RICHIESTE PENDENTI PER
CAPIRE QUALE È LA PIÙ
VICINA A ME. POSSIBILE
STARVATION

LA TESTINA PARTE DAL
BORDO DEL DISCO, E SERVE
TUTTE LE RICHIESTE PENDENTI
IN ORDINE CRESCENTE DI
CILINDRO. ARRIVATO AL UNO, ATTRAVERSO SUBITO, OPPURE
"RIBATTEZZA" SU UN'ULTRA TRACIA
E Torna Indietro.
C'È CONVULSI UN PO' DI
OVERHEAD

FILE
SYS

PROTEZIONE

- LA DIRECTORY

E' UN'ABSTRAZIONE CHE PERMETTE DI RAGGRUPPARE PIU' FILE.
STRUTTURAZIONE AD ALBERO: ESISTE UNA RADICE E
NELL'ALBERO FILE/DIRECTORY SONO PARTE DELLA RADICE.
SE PIU' DIRECTORY CONDIVIDONO UNO STESSO FILE \rightarrow LINKING
 \Rightarrow GRAFO DIRETTO ACICULICO.

- COMANDI DI GESTIONE DELLA STRUTTURA LOGICA

CREAZIONE E CANCELLAZIONE DI DIRECTORY, AGGIUNTA/CANCELLAZIONE
DI FILE, LISTING, ATTRAVERSAMENTO

(2) ACCESSO DEL FILE SYSTEM

- STRUTTURE DATI

L'INSIEME DEGLI ATTRIBUTI DI OGNI FILE VIENE MANTENUTO ALL'INTERNO DEL "DESCRITTORE DEL FILE". OGNI DIRECTORY NECESSITA
DEL COLEGAMENTO CON I DESCRITTORI DEL FILE CHE CI APPARTENGONO.
DIVERSI APPROCCI TRA WINDOWS E LINUX
IL SISTEMA INOLTRE MANTIENE IN MEMORIA CENTRALE UNA STRUTTURA
D'ANNALETTA "TABELLA DEI FILE APERTI"

- UNA COPIA DEL DESCRITTORE DEL FILE
- PUNTATORE AL PROSSIMO RECORD LOGICO DA LEGGERE/SCRIVERE
- INFO RELATIVE AL PROCESSO CHE ACCADE AL FILE.

"MEMORY MAPPING DEL FILE": IL FILE APERTO (O UNA SUA PARTE)
VIENE TEMPORANEAMENTE PORTATO IN MEMORIA CENTRALE.

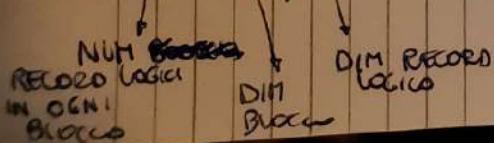
- METODI DI ACCESSO

- ACCESSO SEQUENZIALE: SI ASSUME CHE OGNI FILE SIA ORGANIZZATO
COME UNA SEQUENZA DI RECORD LOGICI. VIENE MANTENUTO
PUNTATORE AL PROSSIMO RECORD LOGICO DA LEGGERE/SCRIVERE
- ACCESSO DIREMO: TRAMITE UN INDICE; SI ACCODE DIRETTAMENTE
AL RECORD LOGICO CERCATO
- ACCESSO A INDICE: PREVEDE CHE OGNI RECORD LOGICO ABBAIA
AL SUO INTERNO UNA "CHIAVE" UNIVOCHE ALL'INTERNO DEL FILE.
A OGNI FILE E' INOLTRE ASSOCIA UNA STRUTTURA TABELLARE DENOMINATA
INDICE, CHE RECONISCE LA CHIAVE.

(3) ORGANIZZAZIONE FISICA

DEFINISCE LA CORRISPONDENZA TRA I RECORD LOGICI CONTENUTI
NEL FILE E L'INSIEME DEI BLOCCHI FISICI IN LIUI SONO MEMORIZZATI.

$$Nb = Db / Dr$$



"RECORD LOGICO" = UNITÀ DI ACCESSO AL FILE
"RECORD FISICO" = UNITÀ DI ALLOCATRICE DEVE
INFORMAZIONI SU DISCO (O BLOCCO)

LO SPAZIO DISPONIBILE PER L'ALLOCAZIONE DI UN VETTORE LINEARE DI BLOCCHI. (4) UN VETTORE LINEARE DI BLOCCHI E' VISTO COME FISICI

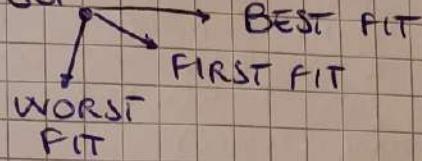
- ALLOCAZIONE CONTIGUA

- OGNI FILE DEVE OCUPARE UN INSIENE DI BLOCCHI FISICAMENTE CONGU
- ACCESSO SEQUENZIALE \Rightarrow SEPARATE
- ACCESSO DIRETTO \Rightarrow FACILE

(SIMILI ALLE PARTIZIONI)

SVANTAGGI:

- PER OGNI FILE E' NECESSARIO TROVARE SU DISCO UN INSIENE DI BLOCCHI LIBERI CONGU
- FRAMMENTAZIONE



MECCANISMO DI DEFRAFFMENTAZIONE PER RENDERE NAGE.

- ALLOCAZIONE A LISTA CONCATENATA

I BLOCCHI NON SONO PER FORZA CONTIGUI. OGNI BLOCCO CONTIENE L'INDIRIZZO DEL PROSSIMO.

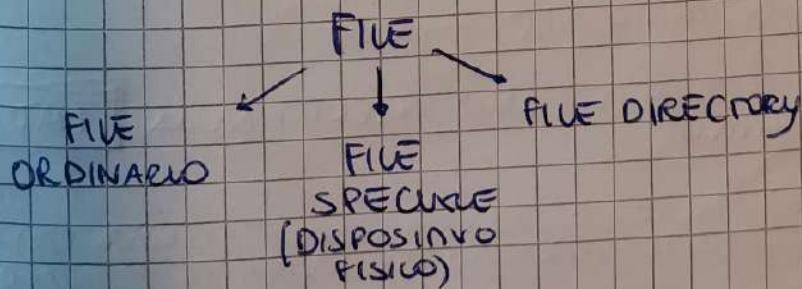
- ACCESSO SEQUENZIALE \Rightarrow POCO COSTOSO
- ACCESSO DIRETTO \Rightarrow COSTOSO
- ELIMINAZIONE DELLA FRAMMENTAZIONE (ESTERNA, CELLULI INTERNA RITRAME)
- POCA ROBUSTESZA:
 - SE UN PUNTATORE SI CORROMPE PERDIO NUO DA QUESTO PUNTO IN POI \Rightarrow LISTA A DOPPIO COLLEGAMENTO
 - \Rightarrow TABELLA DI ALLOCAZIONE DI FILE CON LA MAPPA DI TUTTI I BLOCCHI.

- ALLOCAZIONE A INDICE

- UN BLOCCO VIENE UTILIZZATO COME INDICE PER TUTTI I BLOCCHI UTILIZZATI PER ALLOCARE UN FILE

- NO FRAMMENTAZIONE
- ACCESSO SEQUENZIALE E DIRETTO POCO COSTOSI.
- LA DIMENSIONE DEL BLOCCO INDICE SPESO E' SPRECATO.

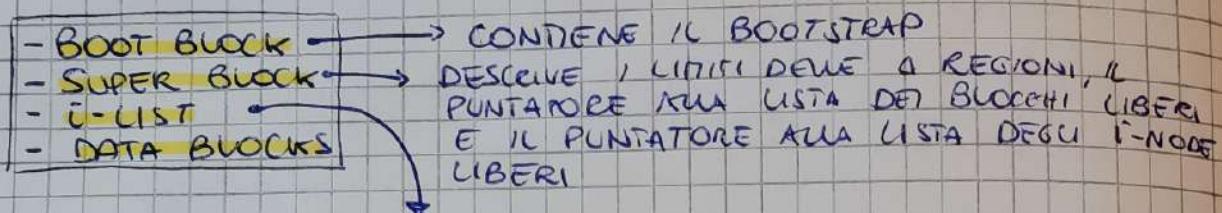
FILE SYSTEM IN UNIX



- STRUTTURA GERARCHICA RAPPRESENTABILE CON E GRAFO ACICLICO DIRETTO (SI LINKING)

ORGANIZZAZIONE PISICA

SUPERFICIE DEL DISCO SUDDIVISA IN 4 REGIONI:



- TIPO (FILE, DIRECTORY, SPECIALE)
- PROPRIETÀ DEL GRUPPO
- DIMENSIONE
- DATA
- LINK (NUMERO DEI NODI CHE RIFERISCONO AL FILE)
- BIT DI PROTEZIONE
- VETTORE DI INDIRIZZAMENTO

13 PUNTATORI

10: RIFERISCONO BLOCCI DATI

11°: BLOCCO DI LIVELLO 1
12°: BLOCCO DI LIVELLO 2
13°: BLOCCO DI LIVELLO 3

SE:

$$\begin{aligned} \text{DIM-BLOCCO} &= 512 \text{ byte} \\ \text{INDIRIZZO} &= 32 \text{ bit} \\ \Rightarrow \text{OGNI BLOCCO} &\text{ CONTIENE } 128 \\ &\text{INDIRIZZI} \end{aligned}$$

ACCESO A UN FILE

- IL RECORD LOGICO E' IL byte.
- METODO DI ACCESSO SEQUENZIALE
- A OGNI FILE APERTO E' ASSOCIATO UN I/O POINTER.

$$\begin{aligned} \Rightarrow 10 \cdot 512 + 128 \cdot 512 + \\ 128 \cdot 128 \cdot 512 + \\ 128 \cdot 128 \cdot 128 \cdot 512 = \\ \pm 4 \text{ GB} \end{aligned}$$

- TABELLA DEI FILE APERTI DI SISTEMA

(TRAS)

CONTIENE UN ELEMENTO PER OGNI FILE APERTO NEL SISTEMA. OGNI VOLTA CHE UN FILE VIENE APERTO VIENE AGGIUNTO UN NUOVO RECORD (2 ENTRATE DISTINTE ANCHE SE IL FILE E' LO STESSO) NELLA TAB. C'E' ANCHE L'I/O POINTER.
C'E' ANCHE UN RIFERIMENTO ALLO i-NODE

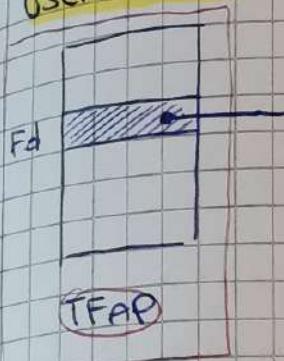
GLI i-NODE DEI FILE APERTI SONO INSERITI NELLA TAB. DEI FILE APERTI (TAB. DEI FILE APERTI)

(TRAT)

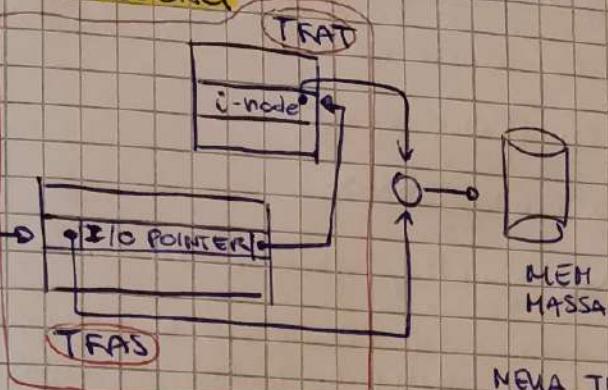
(TRAP)

A OGNI PROCESSO E' ANCHE ASSOCIASTA UNA TAB. DEI FILE APERTI (DI DIM. LIMITATA). IL SISTEMA APRE AUTOMATICAMENTE STANDARD OUTPUT, INPUT E ERROR.

USER STRUCTURE



STRUCTURE DATI GLOBALI



SYSTEM CALL

int open(char namefile, int mode, int prot)

int close (int Fd)

int read (int Fd, char *buf, int n) → UMANO ANCORA PIÙ KNO
DEL UMANO DEVICE INDEPENDENT

int write (int Fd, char *buf, int n)

NUOVE TRAS CI
SONO RECORD DIVERSI,
PER LO STESSO FILE APERTO,
MA PUNTANO SUO STESSO
I-NODE NELLA TRAT.

PROTEZIONE E SICUREZZA

INSIEME DEI MECHANISMI
PER ACCEDERE ALLE RISORSE
CON DETERMINATE POLITICHE

REGOLAMENTARE L'ACCESSO
DA PARTE DI UTENTI ESTERNI

- UN MODELLO DI PROTEZIONE DEFINISCE: SOGGETTO, GLI OGGETTI
ACCEDONO I SOGGETTI E I DIRITTI DI ACCESSO

SOGGETTO = (PROCESSO, DOMINIO)

POLITICHE DI PROTEZIONE:

- ① **DAC** (DISCRETIONARY ACCESS CONTROL): LE POLITICHE DI ACCESSO SONO DEFINITE DAGLI UTENTI STESSI CHE VENGONO DEFINITE PROPRIETARI
- ② **MAC**: OGGETTI CLASSIFICATI IN BASE ALLA LORO SENSIBILITÀ E I SOGGETTI POSSONO ACCEDERE AD ESSI TRAMITE UN'AUTORIZZAZIONE FORMALE (EAD GOVERNANTI)
- ③ **RBAC**: LE DECISIONI SUGLI ACCESSI DA CONCEDERE SONO DECISI IN BASE ALLE FUNZIONI CHE I SOGGETTI SVOLGONO MIGLIORE ALL'ORGANIZZAZIONE.

PRINCIPIO DEL MINIMO PRIVILEGIO: A UN SOGGETTO SONO GARANTITI I DIRITTI DI ACCESSO SOLO PER GLI OGGETTI STRETTAMENTE NECESSARI AL COMPLETAMENTO DELLA FUNZIONE CHE STA SVOLGENDO

- DOMINI DI PROTEZIONE

~~OGGETTO, INSIEME DI DIRITTI DI ACCESSO~~

INSIEME DI

D = ~~OGGETTO~~ oggetto, insieme di diritti di accesso

IN OGNI ISTANTE OGNI PROCESSO E' ESEGUITO IN QUATRO DOMINIO:
PUO' ACCEDERE A CONCETTI INIENI DI OGGETTI E PER CUSCINO
MA PARTECIPARE DIRITTI DI ACCESSO.

ASSOCIAZIONE: STATICA o DINAMICA

L'ASSOCIAZIONE PROCESSO
DOMINIO NON PUO' ESSERE
MODIFICATA DURANTE LA
SUA ESECUZIONE

PUO' ESSERE MODIFICATA.
ES. DI DOMINIO: SYSTEM
E USER.

IN UNIX: VA COPPIA UID E
GID DEFINISCE IL DOMINIO
SUL QUALE OPERA IL PROCESSO.

- UN PROCESSO PUO' CAMBIARE DOMINIO
SE SUO = 1, L'ESECUZIONE DEL FILE
AVVIENE NEL DOMINIO ASSOCIATO ALL'UTENTE
PROPRIETARIO.

- MODELLO A MATRICE DEGLI ACCESSI

- RAPPRESENTA LO STATO DI PROTEZIONE
- GARantisce il rispetto dei vincoli
- PERMETTE LA MODIFICA CONSEQUENTE allo STATO DI PROTEZIONE

RAPPRESENTAZIONE

	X ₁	X ₂	X ₃	S ₁	S ₂	S ₃
S ₁	read		exec		receive	terminate
S ₂		read, write			send	receive
S ₃	write, exec		read	send	terminate	

1. UN SOGGETTO S TENTA DI ESEGUIRE X SU X
2. VIENE GENERATA LA TRIPLOA (S, X, X) E PASSATA AL MONITOR DI PROTEZIONE
3. SE X ∈ A [S, X] OK, ALTRIMENTE NO.

COME SI MODIFICA LO STATO DI PROTEZIONE?

GRAHAM E DENNING

- ① Si può trasferire un diritto α a S_j ^{PER X} solo se S_i possiede α^* (COPY FLAG). Può essere copiato solo α oppure α^* .
- ② Si può assegnare un diritto α per X a S_j solo se "OWNER" $\in A[S_i, X]$
- ③ Si può eliminare un diritto di accesso da S_j solo se "control" $\in A[S_i, S_j]$ oppure se "OWNER" $\in A[S_i, X]$.

- PROBLEMA: MATRICE GRANDE E SPARSA.

\Rightarrow ACL ("ACCESS CONTROL LIST")

PER OGNI FILE TRAMITE I
9 BIT IO CONOSCO QUALI
SOGGETTI POSSONO
ACCEDERE

LA MATRICE VIENE SVODINISCA PER COLONNE IN UNA LISTA

\Rightarrow < SOGGETTO, INSieme dei diritti > ~~LIMITATAMENTE AI SOGGETTI CON UN INSieme NON Vuoto DI DIRITTI PER SOGGETTO~~. \Rightarrow UNIX USA LE ACL SEMPLICI.

• VIENE USATA PREVENDIVAMENTE UNA LISTA DI DEFAULT CONTENENTE I DIRITTI DI ACCESSO CHE PER LE LORO GENERALITA' SONO APPLICABILI A tutti GLI OGGETTI.

SE S_i TENTA DI ESEGUIRE M su O_j , VADO A VEDERE SE ALLA LISTA RIFERITA O_j E' PRESENTE S_i , E CON CHIALE DIRITTO.

• A SECONDA DEL GRUPPO DEL SOGGETTO APPARTENNE (RUOLO) SCRIVO

$UID_1, GID_1 : <-->$

$UID_2, GID_2 : <-->$

UNIX: 3 TIPI
DI SOGGETTO:
PROPRIETARIO
GRUPPO PROPRIET
ALTRI

$UID_1, + : <-->$ \Rightarrow INDIPENDENTI DAL GRUPPO (RUOLO)

$UID_2, + : < \text{vuoto} >$ \Rightarrow UID_2 , INDIP dal GRUPPO, Non Accese.

$+ , + : <-->$

ACL \Rightarrow INEFFICIENTI

CL "CAPABILITY LIST"

⇒ SUDDIVISIONE DELLA MATRICE PER RIGHE. E' QUINDI UNA LISTA DEGLI OGGETTI, E PER OGNI SOGGETTO ESISTE TALE LISTA.

ELEMENTO = "CAPABILITY"

↳ I SOGGETTI PERÒ NON POSSONO E NON DEVONO ACCEDERVI DIRETTAMENTE

- DI SOLITO SI COMPOSTO DI:
- ID OGGETTO (IN UNIX: C-NUMBER)
 - MAPPA DI BIT PER I DIRITTI

STRUTTURA PROVVISORIA DEL S.O.

QUANDO UN PROCESSO FA UN OPEN, IL 2° PARAMETRO, INDICA LA MODALITÀ CON CUI VOGLIO APPIRE IL FILE. L'ATO CONSEGUE CHE QUESTA SIA COMPATIBILE CON THI BIT

CONCLUSIONE: NEI S.O. VENGONO USATE LE ACL OTE CL

* QUANDO UN SOGGETTO TENTA DI ACCEDERE PER LA 1° VOLTA AD UN OGGETTO, VIENE FATTA UNA RICERCA NELLA ACL ASSOCIA A TALE OGGETTO. SE L'ACCESSO È CONSENTITO, VIENE CREATA UNA CAPABILITY LIST CHE SI ASSOCIA AL SOGGETTO E PER GLI ACCESSI SUCCESSIVI SI VERIFICA QUESTA.

* IN UNIX: TRAP = CL

- REVOCÀ / CONCESSIONE DEI DIRITTI DI ACCESSO.

PER LA REVOCÀ NELL'EACL SI FA PRESTO: VADO A COLLEGARE E CAMBIO I BIT. * LE CL INVECE VANNNO SCANNERIZZARE TUTTE.

MODELLO DI Bell-LAPADULA ⇒ GESTISCE LA SICUREZZA (USATO IN AMBIENTE MILITARE)

MAC

① "THE SIMPLE SECURITY POLICY"

LETTRA: $SC(s) \geq SC(x)$: UN SOGGETTO PUÒ LEGGERE LE INFORMAZIONI CONTENUTE IN OGGETTO CON UN LIVELLO DI SICUREZZA \leq M SUO.

② "STAR PROPERTY"

SCRITTA: $SC(s) \leq SC(x)$: UN SOGGETTO PUÒ SCRIVERE IN OGGETTO CHE APPARTENGONO A LIVELLI DI SICUREZZA \geq M SUO

⇒ UN SOGGETTO CHE STA A UN DETERMINATO LIVELLO NON PUÒ MAI FORNIRE INFO A SOGGETTI CHE SONO A UN LIVELLO PIÙ BASSO.

SUPPONIAMO S₁ SIA A LIVELLO RISERVATO E ENTRI UN UTENTE
OS DUE S₂, AL LIVELLO PUBBLICO.

~~S₁~~ S₁ PUO' ESEGUIRE IL PROGRAMMA O₂ PERCHÉ PER LA ②
ESSENDO S₁ = RISERVATO E O₂ PUBBLICO.
NON PUO' PERO' SCRIVERE LE INFO LUNGHE IN O₃ PERCHÉ
APPARTIENE A UN LIVELLO INFERIORE.

=> NON SI GARANTISCE PERO' L'INTEGRITÀ DEI DATI.

MODELLO BIBA

① E ② SONO INVERTITE. SI PUO' DEMOSTRARE CHE COSÌ
SI MANTIENE L'INTEGRITÀ DEI DATI MA NON LA SICUREZZA

- BEL-LAPADUA E BIBA SONO OVVIAMENTE IN CONFLITTO,
NON POSSONO ESSERE USATI ENTRAMBI.
POSSONO PERO' ESSERE AFFIANCATI A UNA MATRICE DEGLI
ACCESI.