

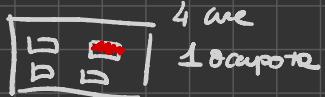
25/09/2023

## SISTEMI OPERATIVI

### ↳ PROCESSI

main

{ monoflano



attivazione o non  
comparte i file  
sistema di controllo



Codice multithreaded      programmazione concorrente

26/09/2023

## SISTEMA OPERATIVO → SOFTWARE

- ↳ ALLOCAZIONE DI RISORSE
- ↳ COME GESTIRE I CONFLITTI
- ↳ CONTROLLO DEI PROGRAMMI

CONTROLLA E PROTEGGE I PROCESSI  
DA ALTRI PROCESSI → SEGMENTATION FAULT

ESEMPIO:

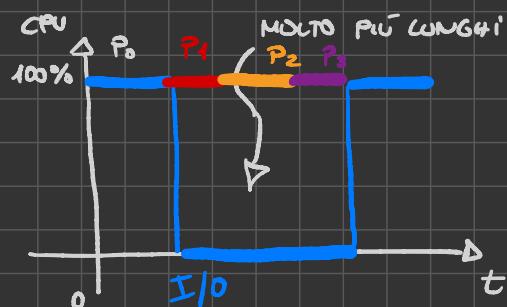
DEF.

QUEL PROGRAMMA CHE IN OGNI Istanze È ESEGUITO NEL COMPUTER → KERNEL

### EVOLUZIONE:

0 INSIEME DI LIBRERIE VENIVANO UTILIZZATE PER RENDERE PIÙ VELOCI alcune ATTIVITÀ  
↳ ROUTINE, FUNZIONI

1 SI POTEVA USARE UN PROGRAMMA ALLA VOLTA



≡

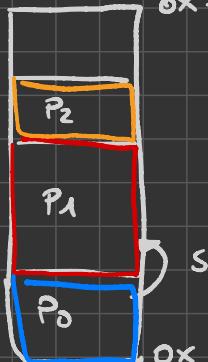
while (1);

≡

STALLO DEL COMPUTER

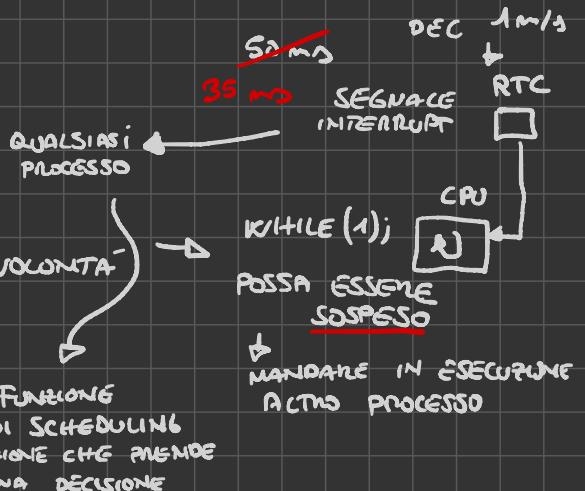
SISTEMA TIME SHARING DIPENDE DA RISORSE CONDIVISE, SFUTTANO TUTTO L'HARDWARE

2 SISTEMA MULTI PROGRAMMATO (MOLTO PIÙ COMPLESSO)



## GESTIONE MEMORIA E CONTROLLO DEI PROGRAMMI

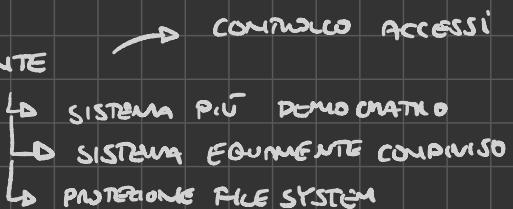
PREDAZIONE → SOSPENSIONE UN PROCESSO CONTRO LA SUA VOLONTÀ



PROTEZIONE → DEI PROCESSI DA ALTRI PROCESSI

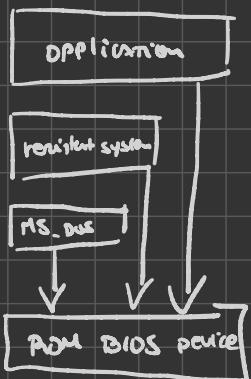
MMU → MEMORY MANAGEMENT UNIT

3  
MULTI PROCESSO E MULTIPLO UTENTE



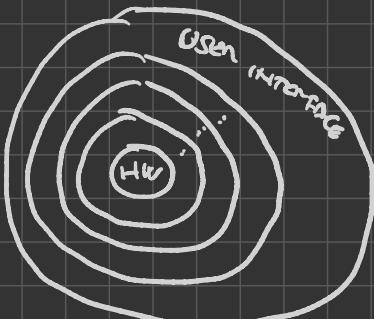
## STRUTTURA SISTEMA OPERATIVO

### MS-DOS



NON LASCIARE HARDWARE SCOPERTO

NASCITA SISTEMI OPERATIVI A STRATI



PROCESSO PARSA SU TUTTI GLI STRATI

HW ↗ LAYER 1 ↘ LAYER 2 ↗ ...

PROTEGGERE L'HW, LAVORO IN TERM

PORTABILITÀ SU ALTRI HW modifica del layer 1

LATI NEGATIVI → PESSIME PERFORMANCE con 15/10 Layer

10:18



2 livelli

PERFORMANCE , ALL'AVANGUARDIA

SISTEMA MODULARE → KERNEL → HW

tutto il resto viene spinto fuori che interagisce con il kernel

possibile bontà combinare il kernel, tenere nascoste anche con programmazione ad oggetti  
monolitico → kernel come → poco portatile  
→ performance buone

UNIX → LINUX → micro Kernel → tutti i moduli non necessari stanno fuori  
MAC

MONOLITICO

micro KERNEL → moduli scritti con linguaggio di programmazione ad oggetti

## FUNZIONI SISTEMA OPERATIVO

- Command line interface (CLI)
- Graphical User Interface (GUI)
- Esecuzione programmi
- controllo memoria
- Comunicazione tra processi
- Controllo e identificazione errori

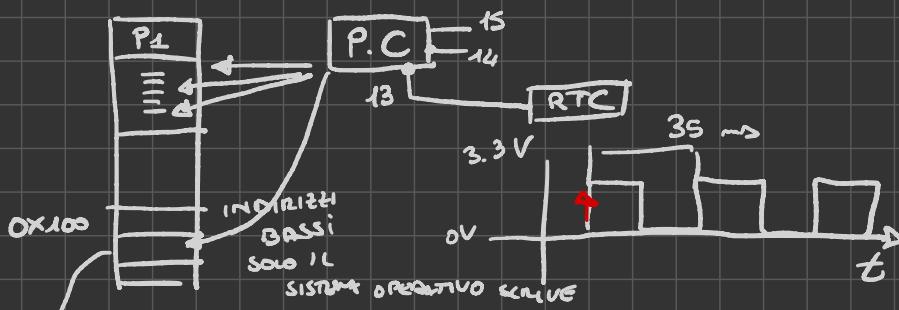
## PROTEZIONE

- Programmi
- RISORSE → CPU, periferiche, memoria, SPECIAL REGISTER
- UTENTI
- DATI
- E SE STESSO

PROGRAMMA → WHILE (1); → INTERRUPT REAL TIME CLOCK



ogni tot alleo  
il valore  
de 0 e 1



## FUNZIONI SISTEMA OPERATIVO, PUNTATORI A TABELLA DI SCHEDULING

LD DECIDE COSA FARE CHE PROGRAMMA LANCIARE

TIME SHARING → DEMOCRATICITÀ DEI PROCESSI

2/10/2023

### PROTECTION

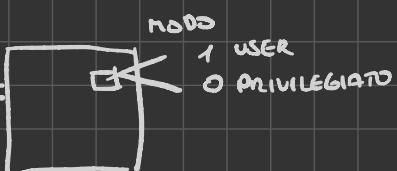
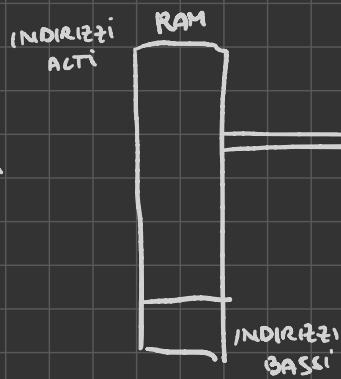
LD PREEMPTION

CPU execution Modes: I/O protection

Gentine I/O → modalità di esecuzione duplice → user mode

→ Supervisor mode (Kernel mode)

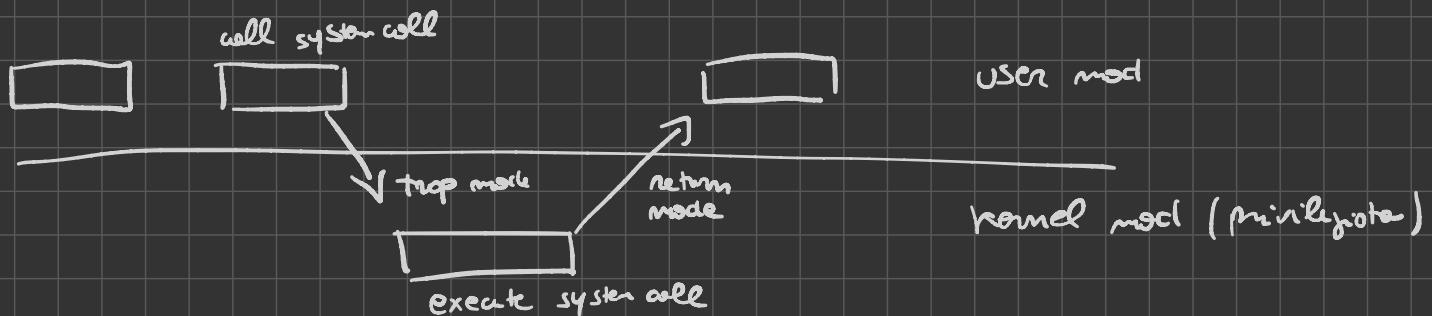
AL BOOT PARTE CON LA MODALITÀ  
privilegiata per switchare in  
modalità non privilegiata



Lavorare e scrivere in indirizzi  
bene bisogna essere in privilegiato

ISTRUZIONE MACCHINA per poterlo  
switchare da user mode a privilegiato

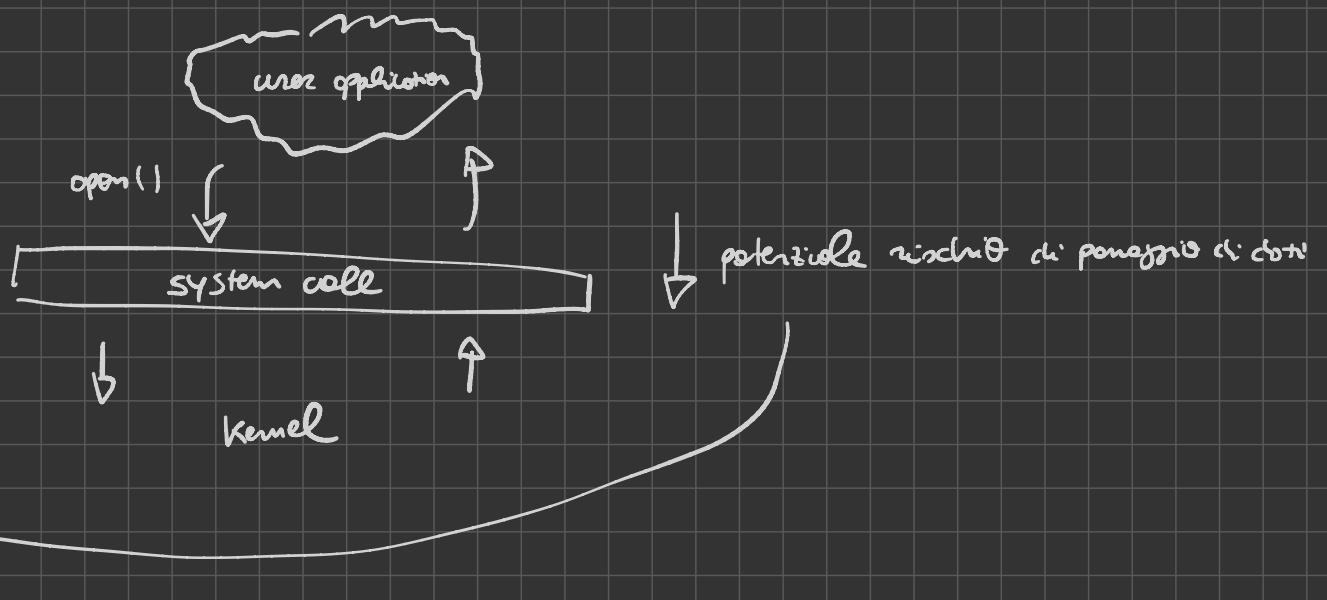
### SYSTEM CALL (chiamate chi invoca)



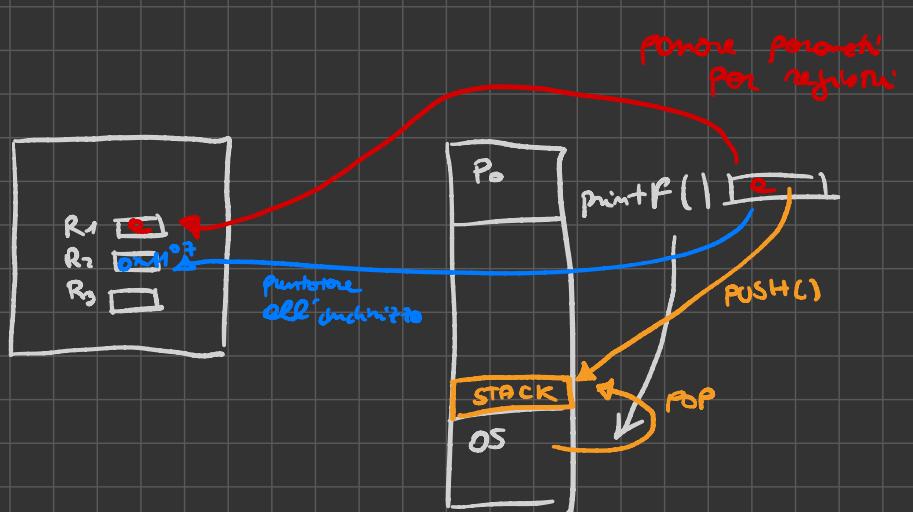
tutte le funzioni hanno una trap al loro interno (ogni I/O process)

System call API sono delle funzioni scritte ad alto livello

stdio.h



- REGISTERS 1
- BLOCK, OR TABLE 2
- STACK 3



piccolo over head perché  
c'è una copia

- piccole dimensioni
- registrati mem infiniti

Quali tipologie di system call esistono?

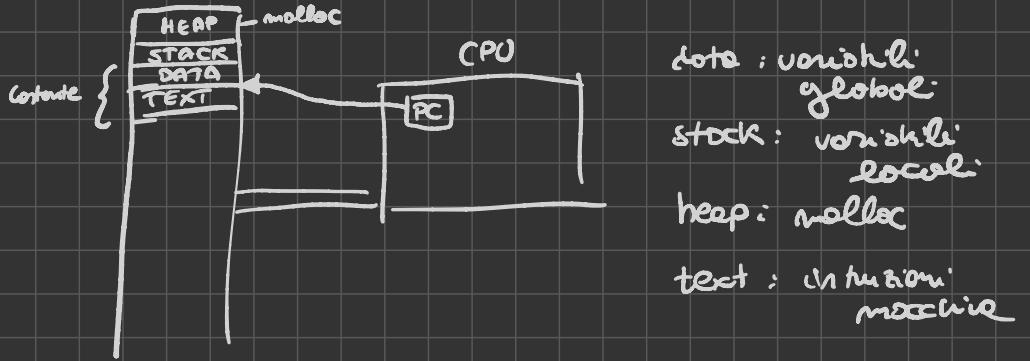
- Processi
- File manager
- Device manager
- Information maintenance
- Comunicazione tra processi

## GESTIONE PROCESSI :

Multi programmazione

processo → sezione codice → code text (tutte istruzioni macchina)

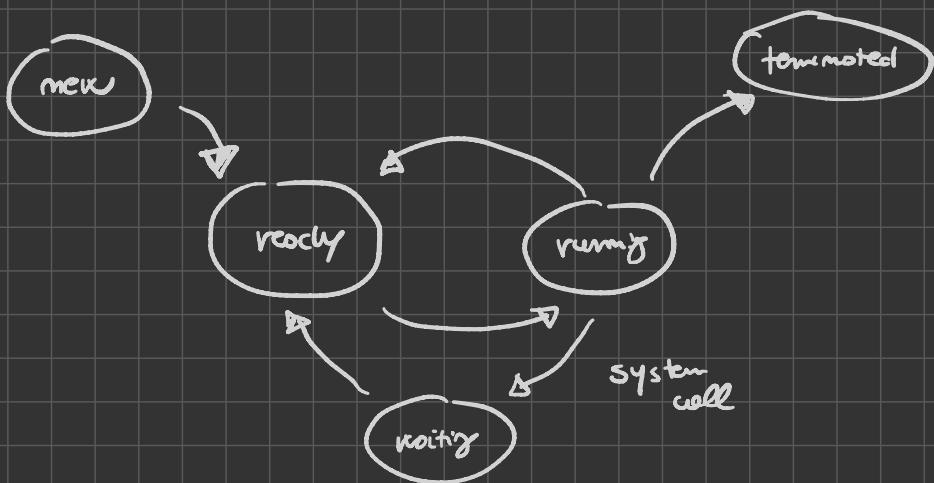




Fasi della vita di un processo

- NEW (appena creato)
- RUNNING (in esecuzione => PC)
- WAITING (in attesa per I/O o un altro processo)
- READY (pronto però non c'è disponibilità CPU)
- TERMINATED (finita esecuzione)

ogni processo può avere  
 eseguito con un solo  
 core  
 8 core 8 processi



PCB process control Block struct

Process state

Program counter

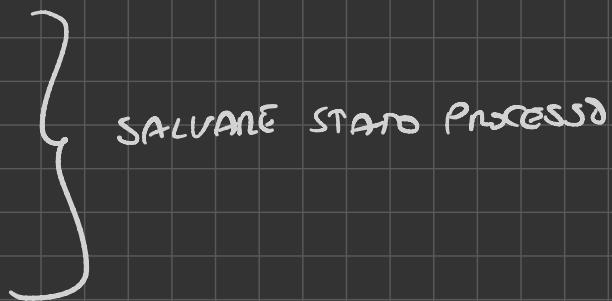
CPU register

CPU scheduling information

Memory management information

Acording information (impostante)

I/O status information



Vengono allocate nelle zone base del sistema operativo

CREAZIONE PROCESSO

↳ creazione di figli

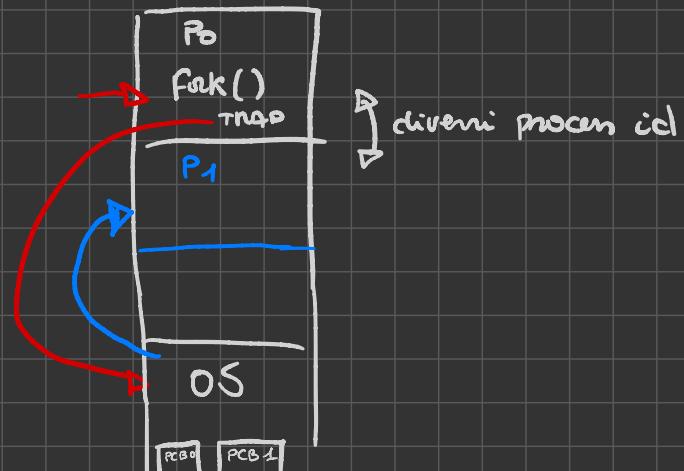
Creazione altrove

03/10/2023

Processo figlio si crea solo se vi è un processo padre che lo crea non il contrario, formando un albero della gerarchia dei processi.

## CREAZIONE PROCESSI in (UNIX)

↳ SYSTEM CALL (FORK) → ESEGUE UNA COPIA ESATTA DEL PADRE (stesso codice)



System call fork → ritorna il valore pid

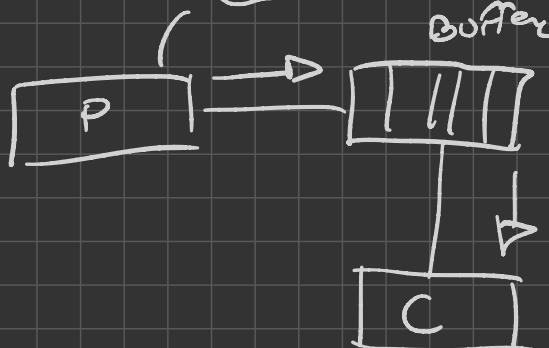
(fork - exec in C)

pg. 16

Esempi

ciao

problema compositivo



producer-consumer

producer scrive velocità  
monica nel buffer

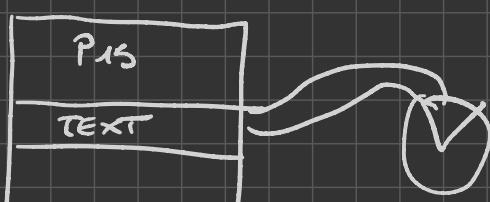
consumer

scrive o velocità  
attuale

while (1);  
→ fork();

## SYSTEM CALL EXEC

EXEC ("CHROME");



## TERMINAZIONE PROCESSI :

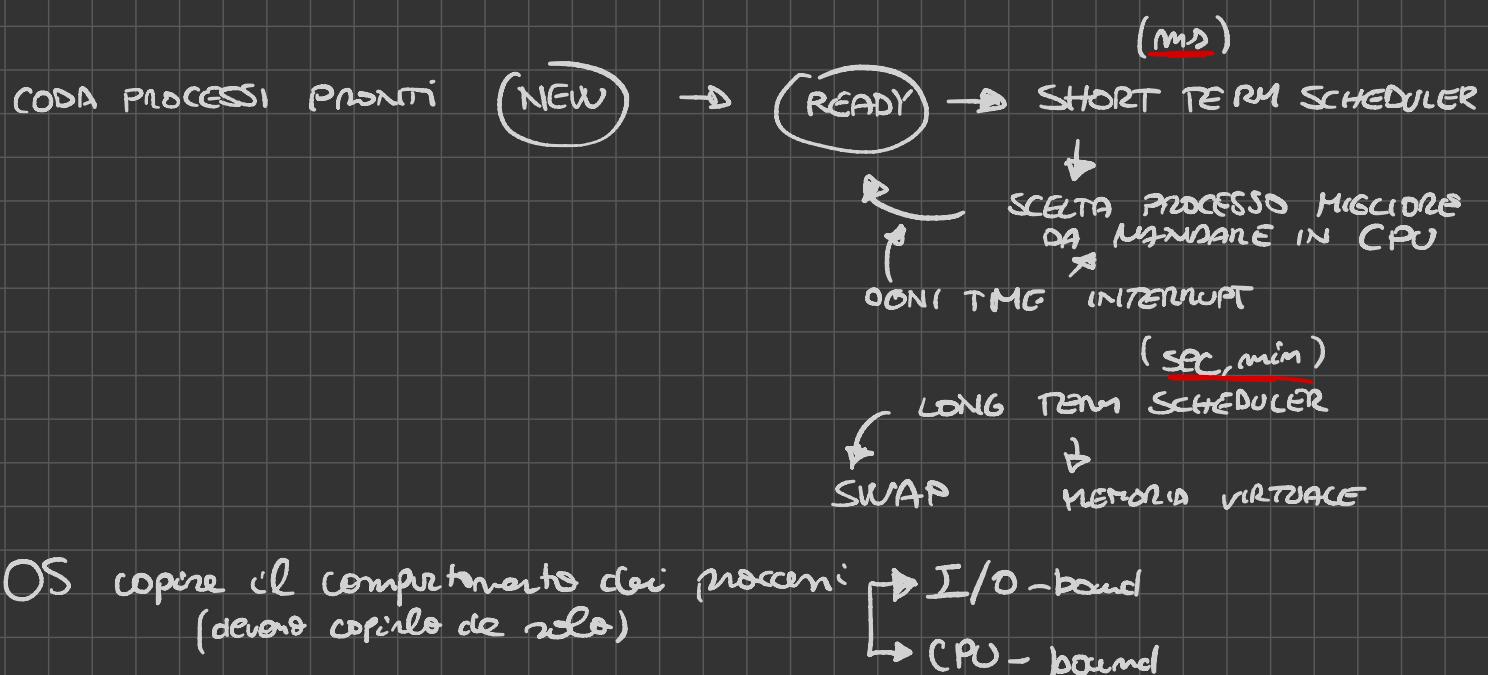
programm. concorrente  $\rightarrow$  il padre deve aspettare che finisca il figlio per controllarlo

## CAMBIO DI CONTESTO

$\hookrightarrow$  GESTIRE CAMBIO DI CONTESTO  $\rightarrow$ 

- READY CODE (PROCESSI PRONTI)
- I/O CODE (INPUT/OUTPUT)
- SYNC CODE (SINCRONIZZAZIONE)

SLIDE 4 PG 4

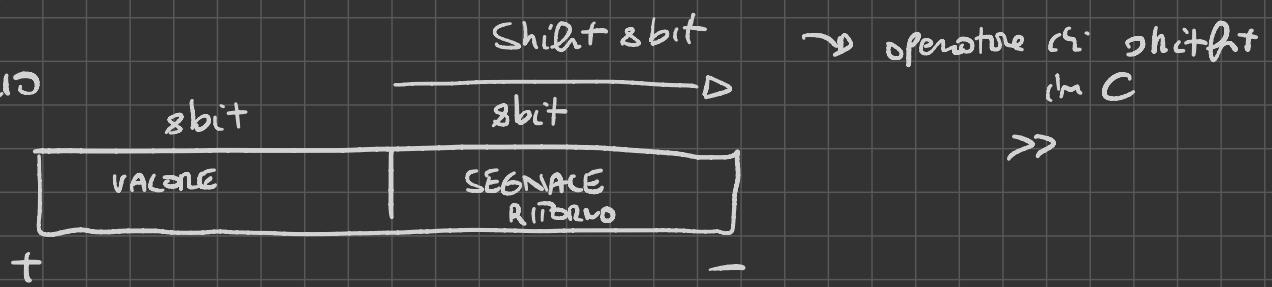


## SCHEDULING POLICIES :

- $\rightarrow$  dare risorse a tutti
  - $\rightarrow$  Priorizzare il più importante
  - $\rightarrow$  Garantire un corretto sincronismo nei processi paralleli
  - $\rightarrow$  ottimizzazione qualcosa
- } Schedulazione

09/10/2023

## LABORATORIO



10/10/2023

## Context switch

SCHEDULING → SALVARE PROGRAMMA PCB (process control block)

↳ MOLTO ONEROVO

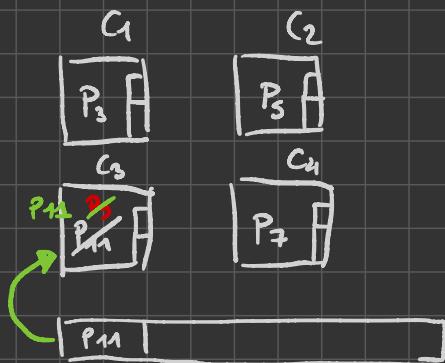
OVERHEAD OF A CONTEXT SWITCH

↳ DIRECT = SAVE / STORE → ISTRUZIONI MACCHINA APPSTA

↳ INDIRECT = CACHES AND TLB → MOLTO DIFFICILE DA MITIGARE



IL PROGRAMMA  
CHE ENTRA AL POSTO  
DEL PRECEDENTE PERDE UN SACCO DI TEMPO



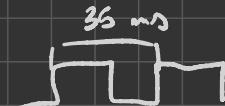
AFFINITÀ per un processore

P11 una volta uscito dal processo conviene  
che ricorra nel processo 3 (C3)

## KERNEL PREEMPTION

↳ SI PUÒ? SI MA SOLO UN PROGRAMMA DEL SISTEMA OPERATIVO, NON UTENTE

TIMER INTERRUPT



PUÒ ANDARE A SOSPENDERE IL FLUSSO KERNEL  
SOLO NEI SISTEMI PIÙ RECENTI

↳ KERNEL PREEMPTIVE

↳ DEVE ESSERE RIENTRANTE

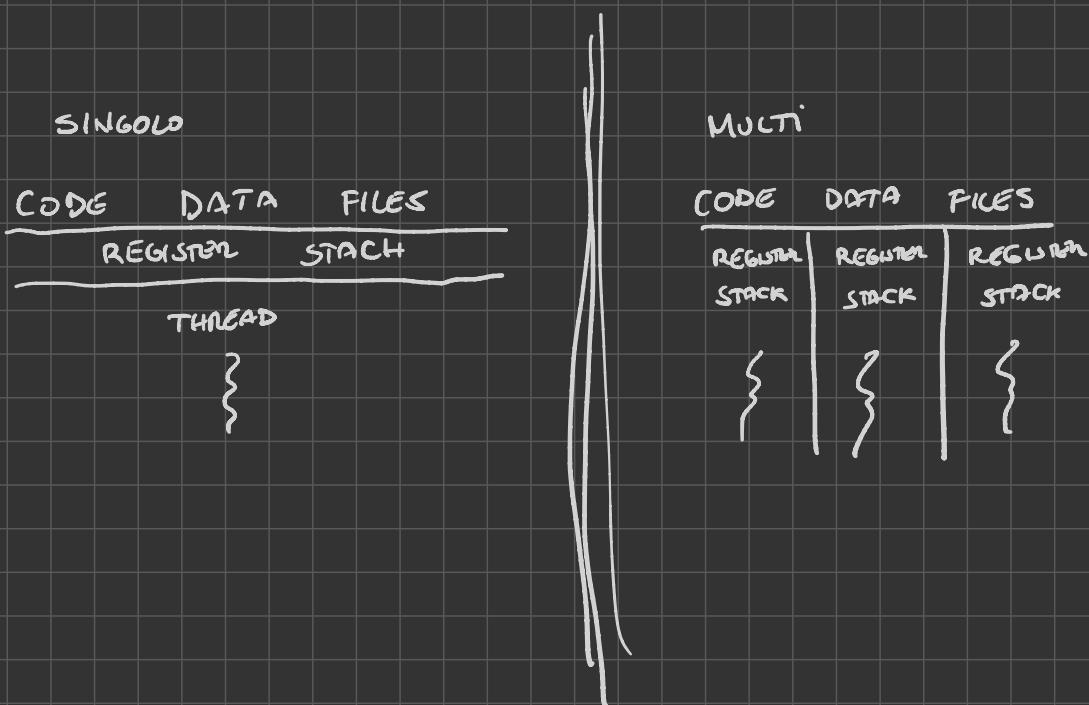
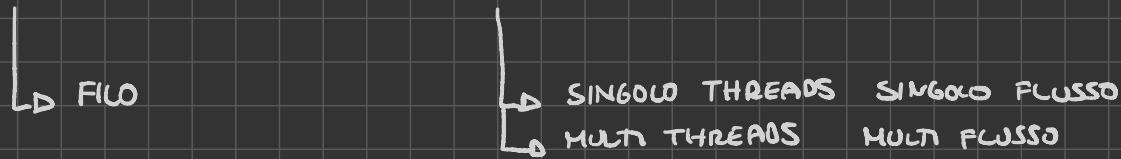
↳ NO VARIABILI STATICHE GLOBALI

↳ NO CODICE CHE MODIFICA SE STESSO

↳ NON DEVE INVOCARE FUNZIONI NON RIENTRANTI

POSSENGO ANCHE ESEGUIRE IN PARALLELISMO COME UN PROGRAMMA UTENTE

# THREADS VS PROCESSO



## MULTIPLO THREADS

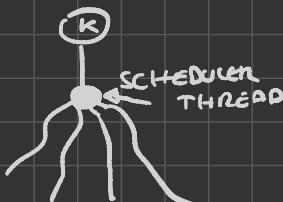
- ↳ LWP (Light-weight process) → occupano meno memoria → più veloci
- ↳ NESSUN INTERRUPT
- ↳ PROGRAMMAZIONE CONCURRENTE → NESSUN CONFLITTO ALLE RISORSE CONDIVISE  
↳ LOCK, SEMAFORI

LIBRERIE → POSIX Pthreads.h  
 ↳ WIN32 threads.h  
 ↳ JAVA threads.h

### MAPPATURA MODELLI MULTITHREADS



↳ MANY TO ONE → MAPPATI SU UNA SOLA CPU  
NON SFUSSI IL PARALLELOISMO

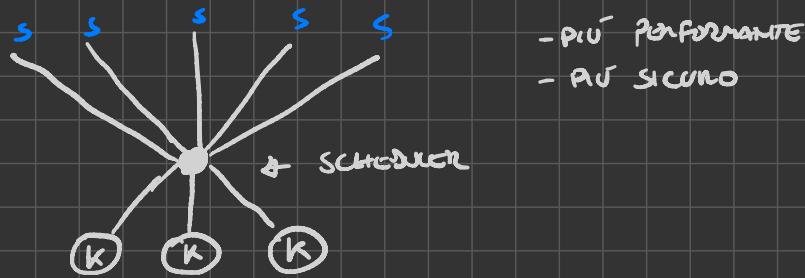


- MAPPATURA MOLTO SICURA
- MOLTO POMPOSE

↳ ONE TO ONE → MAPPATI SU UNO A UNO IN THREAD DIVENTANO KERNEL THREAD

- MENO SICURA
- MOLTO PErFORMANTE

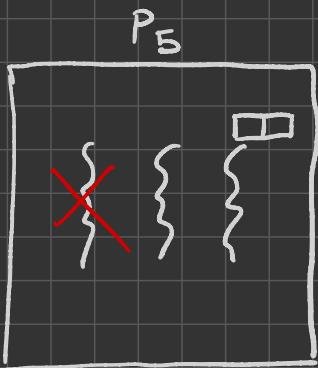
## MANY TO MANY



## PROBLEMI NON BANALI PROGRAMMI MULTITHREADING

### ↳ CANCELLAZIONE THREADS

- ↳ SE DEVO TERMINARLO DEVO PRIMA COMUNICARE  
SE NON CREA CONFLITTI
- ↳ CANCELLAZIONE ASINCRONA NO
- ↳ CANCELLAZIONE DIFFERITA SI



SLIDE 5 PAG 18

### ↳ GESTIONE SEGNAZI → A LIVELLO LIBRERIA MULTITHREAD

### ↳ SCHEDULER ACTIVATION → UP CALLS → CHIAMA IL SISTEMA OPERATIVO ALLO SCHEDULER DEL MULTITHREADS

## LINUX THREADS

- ↳ LIBRERIA CLONE() crea altri filoni di esecuzione che posso condizionare determinate cose decisive del programmatore

FORK() NON CONDIZIONA NEL MULTITHREAD  
THREAD() CONDIZIONA TUTTO

## JAVA THREADS

- ↳ ESTENSIONE CLASSE
- ↳ IMPLEMENTAZIONE DI INTERFACCIA

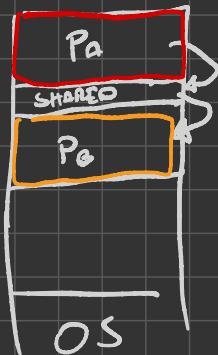
# COMUNICAZIONE TRA PROCESSI

→ PROCESSI COOPERANTI  
→ INDEPENDENTI

→ CONVENIENTE  
→ SPEED-UP COMPUTAZIONALE  
→ CONSOLIDAMENTE INFORMATIVI

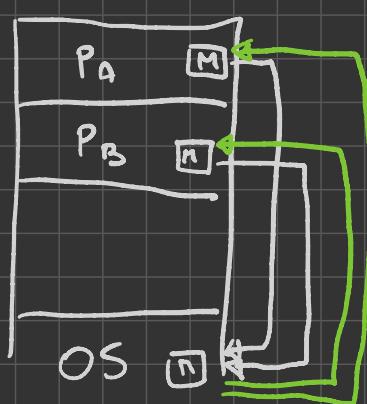
## SHARED MEMORY

- MOLTO VELOCE
- MOLTE DI DATI MOLTO GRANDI
- POCO SICURA



## MESSEGE PASSING

- MESSAGGI PIÙ PICCOLI
- MOLTO SICURA
- PIÙ LENTA



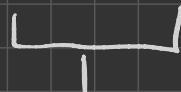
System call

P<sub>A</sub>, P<sub>B</sub>

OS

La memoria viene dedicata tramite una system call, anche le dimensioni quindici 4 system call

Ogni richiesta è una system call  
↳ 10 richieste 10 system call



processi per comunicare devono conoscere trasmittente poche e figlio

crea una memoria

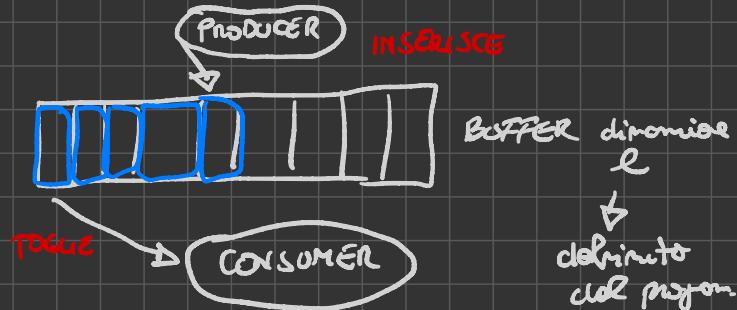
ha accesso alla memoria

Possibilità di fare un monogno broadcast

# PARADIGMA CONSUMATORE - PRODUTTORE

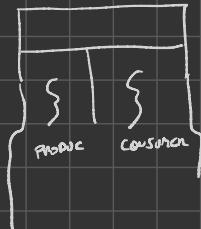
Procedi collaborano e un task ottimizzando i vantaggi

Producer veloci e estremo



Consumatore, più o meno contante  
codifiche dinamiche

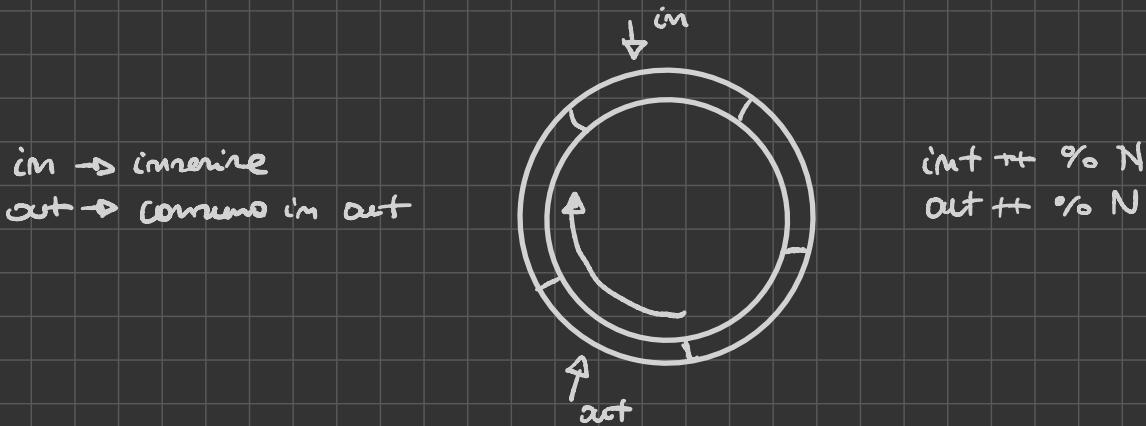
Procedi 2 threads



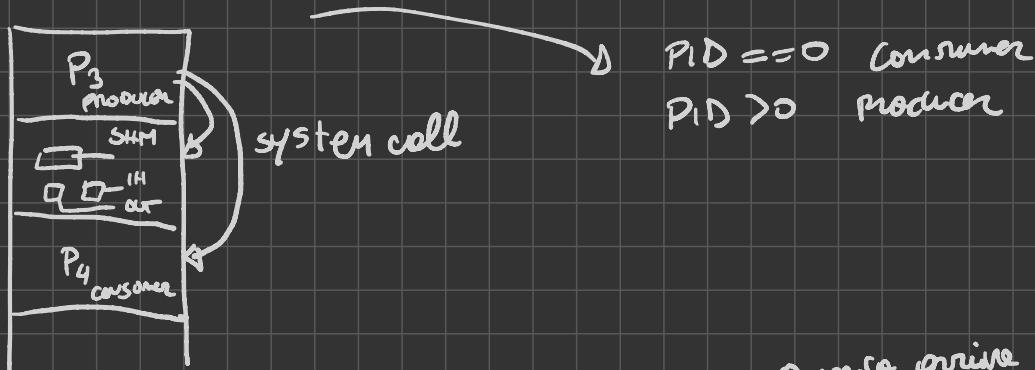
UNBOUNDED  $\rightarrow$  PRODUCER SOUDASCHIENE LA MEMORIA ANCONTA  
CONSUMATOR TORNA A UN DEFAULT ] PARADIGMI  
LETTA

BOUNDED  $\rightarrow$  PRODUCER, CONS NON SOUDASCHIENE LA MEMORIA

SHARED MEMORY SOLUTION BUFFER CIRCOLARE



ARCHITETTURA PROGRAMMA



Quando arriva l'interrupt

Vorrebbe condizionare protette e condizionare  
" " simononizzate

SHARED MEMORY  $\#include <sys/shm.h>$

shm\_id  
shm\_get

shmop  
attach  
shmat  
detach

shmop\_remove] - solo pedire

MESSAGE PASSING  $\rightarrow$  IN ANOMIA (INTENT)

Attraverso system call  $\rightarrow$  ogni dato  $\rightarrow$  send receive

Creare un link  
di sincronizzazione

(dirette  
ricevute e mandate  
si devono conoscere  
Pache-fijo)

P send  
Q receive

- MOLTO SICURA
- LENTA
- NO CONNEZIONE

GESTITE  
DAL SISTEMA  
OPERATIVO

attraverso delle API

send  
receive

se non mi convenga (mail box)

SEND(A, message)  
RECEIVE(A, message)

- unicast
- broadcast
- multicast

$\rightarrow$  PARIGIUS BLOCCANTE  $\rightarrow$  finché non ho ricevuto  
si blocca (synchro)

(asynchronous)

$\rightarrow$  NON BLOCCANTE

Sincronizzazione del buffer

- 1) (rendevous BLOCCANTE) zona capacità
- 2) Bounded capacity (produttore - consumatore)
- 3) Senza capacità (unbounded)

$\rightarrow$  per scrivere quando il pacchetto è pronto

$\rightarrow$  tempi molto brevi < 50 ms

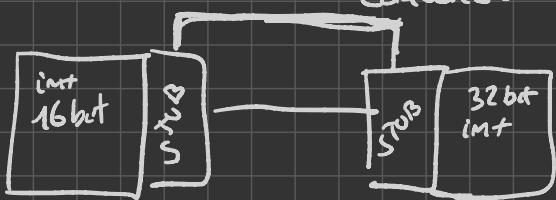
Metodologia  $\rightarrow$  POLLING  $\rightarrow$  RECEIVE, RECEIVE BURSE FORCE

$\rightarrow$  INTERRUPT  $\rightarrow$  FUNZIONE APPENA ATTIVATA L'INTERUPT PROCESSA IL DATO  
che attende

$\nabla$  tempi > 50ms

# CLIENT - SERVER

- Sockets  $\rightarrow$  161.25.19.8 : 1625 solo byte
- Remote Procedure Calls  $\rightarrow$  FUNZIONE CHE VENNE INVOCATA IN REMOTO  
CONVERSIONE (MARSHALLS)

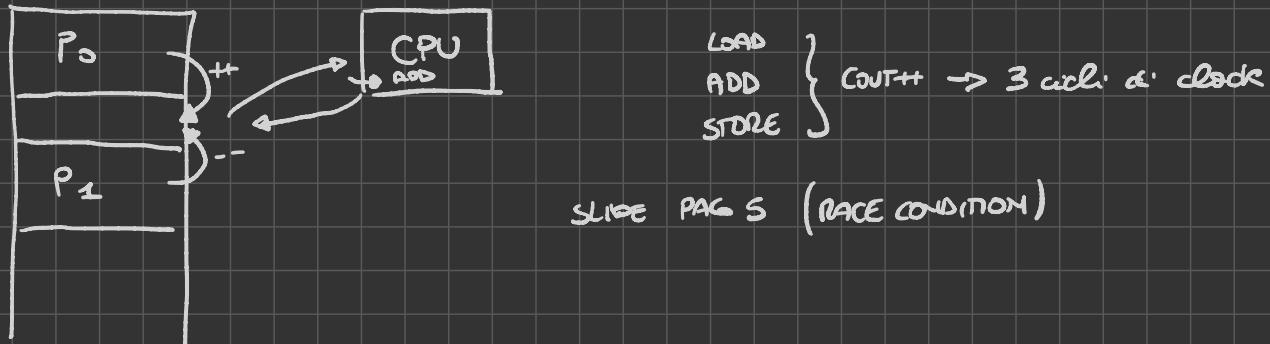


- Remote Method invocation (JAVA)

Stesso concetto in macchina virtuale JAVA

16/10

Problema della sezione critica  $\rightarrow$  memoria condivisa



Progettare ++; consumo --;  $\rightarrow$  imbarco allo scheduler o vuole con il volo grande o è sbagliato  $\rightarrow$  uno che non è formattato corretto

Oggi giorno anche il codice di Load vuole più di un archi di clock

Ho struttura dati condivise  $\rightarrow$  problema della sezione critica (sempre)  
ACCESSO LETT. SCRITT.

Come risolvere il problema?

- 1) MUTUA ESCLUSIONE  $\rightarrow$  nelle sezioni critiche mi entro una volta sola se c'è P0 mon ante P1
- 2) PROGRESSO  $\rightarrow$  NON bisogna impedire di entrare ai processi nelle sezioni critiche es 10 processi vogliono entrare dentro che ne entra 1  
LO DEATH LOCK  $\rightarrow$  serve entrambe le caratteristiche

3) ATTESA LIMITATA → BISOGNA GARANTIRE CHE TUTTI I PROCESSI ENTRINO  
↳ ATTESA LIMITATA APPUNTO → ATTIVANDO UN CARNISH

↳ DI SOLITO È IL SISTEMA OPERATIVO CHE DECIDE  
↳ ALLOCAZIONI DI AGEING

SOLUZIONE DI PETERSON → LOAD E STORE ATOMICHE

↳ UNICO CYCLE DI CLOCK

↳ ALLOCARE DUE VARIABILI  
INT TURN;  
BOOLEAN FLAG [2]; → array di processi 1, 2

↳ REMAINDER SECTION → ESSENTEALE PER LE DUATIT LOCK

FUNZIONE ATOMICA → non interromponibile → esempio quelle che chiudono un circuito di clock  
(potendo essere anche eseguite anche su più cicli di clock)

↳ DISABILITARE L'INTERRUPT → ALL'INTERNO DELLA SEZIONE CRITICA  
(SOLA ALL'INNDO DI UN SINGOLO CORE)

↳ MUTUA ESCLUSIONE ATOMICA A LIVELLO HW

↳ ISTRUZIONE → TEST AND SET → A = FALSE; ↳ RITORNA FALSE  
↳ SWAP ↳ SEMPRE TRUE

while (true) {

    while (test\_and(&lock));  
    /\* critical section \*/  
    lock = false;  
    /\* remainder \*/

} ↴

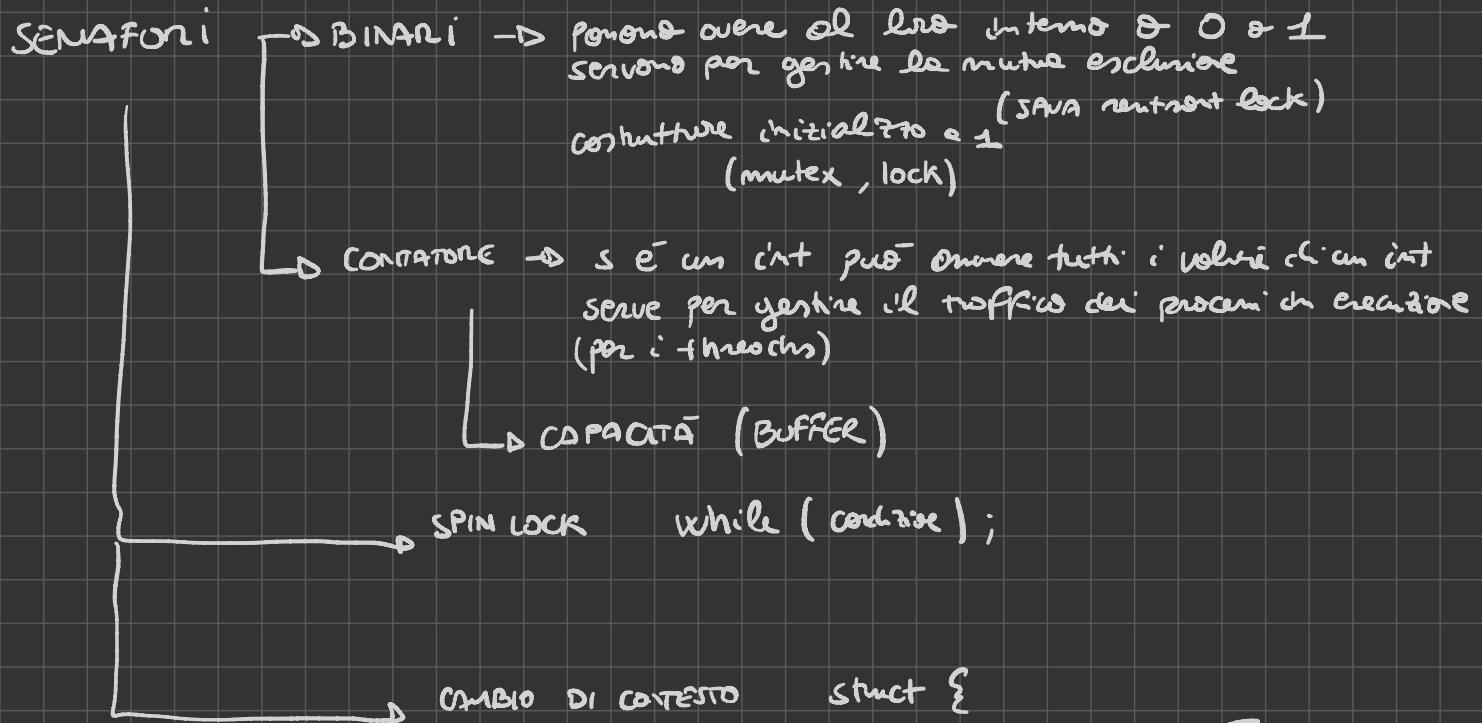
SEMIFORI (ingegni alto livello) (oggetti HW)

↳ TOOL-HARDWARE per la sincronizzazione  
↳ wait(); → S numero di processi S--;  
↳ signal(); → S++;

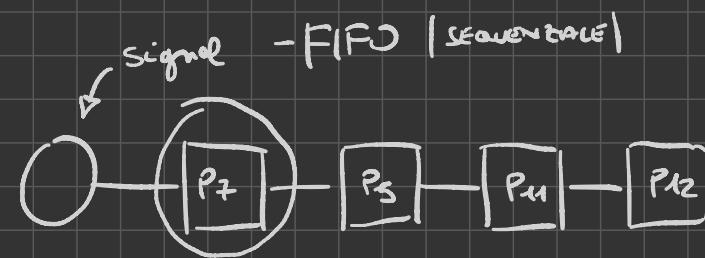
↓  
semfne mutex

while (true) { wait(mutex); signal(mutex); }

17/10/2023

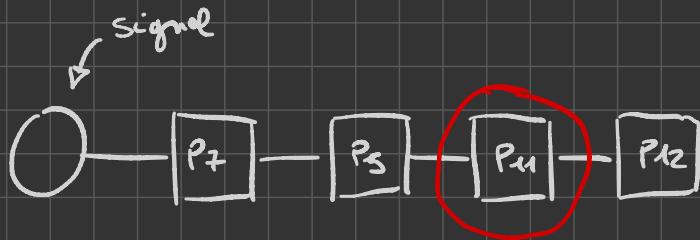


struct {  
 int value;  
 struct process \* list; } semaforo;



} NON SI SPARECANO CICLI  
DI CPU

- RANDOM → PROBABILISTICA → NON SEQUENZIALE



CON UN SOLO CORE NON HA SENSO AVERE LO SPIN-LOCK SOLO CAMBIO DI CONTESTO

INVECE HA SENSO SE LO SPIN-LOCK È INFONDO AL TIMER - INTERRUPT, INFATTO

DI TEMPO

- IN PIÙ RISVEGLIO IN BASE AL CONTESTO

# DEADLOCK AND STARVATION $\rightarrow$ (NON DOMINATIVO)

$\hookrightarrow$  (NON GARANTISCE IL PROGRESSO)

$\rightarrow$  ATTESA IN CUI IL SEGNALE CHE PUÒ GENERARE LO SBLOCCO, SONO IN ATTESA

$\hookrightarrow$  DOBBIANO DISTURGERE IL PROGRAMMA PER USCIRE DAL DEADLOCK

(MAI ALTERNARE LA RICHIESTA DELLE RISORSE)

$\hookrightarrow$  DEADLOCK CIRCOLARE

RISORSE CONSUMABILI  $\rightarrow$  USA E GETTA

$\hookrightarrow$  DEADLOCK CON UNA RECIFE BLOCCANTE

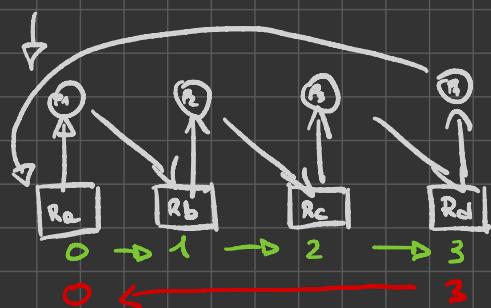
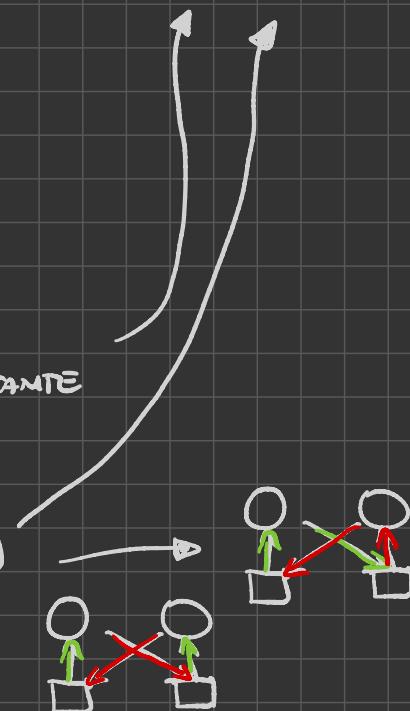
Condizioni

DEADLOCK  $\rightarrow$  MUTUA ESCLUSIONE

$\rightarrow$  HOLD AND WAIT (RICHIESTE DI RISORSE)

$\rightarrow$  NO PREEMPTION

$\rightarrow$  ATTESA CIRCOLARE



TUTTA LA CATENA PONTE  
ALLA CHIUSURA

SOLUZIONE VI È UNA REGOLA : CERTO NUMERO DI RISORSE E PROCESI

LI CHIAMO 0, 1, 2, 3 (INDICE CRESCENTE)

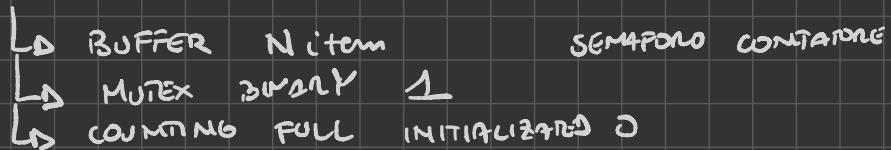
DARNO CHIEDERE IN ORDINE CRESCENTE

LA CENA DEI 5 FILOSOFI  $\rightarrow$  1/10 SI GENERA IL DEADLOCK

ESISTE ALGORITMO CHE RIMUOVE IL DEADLOCK  $\rightarrow$   $O(m^m)$

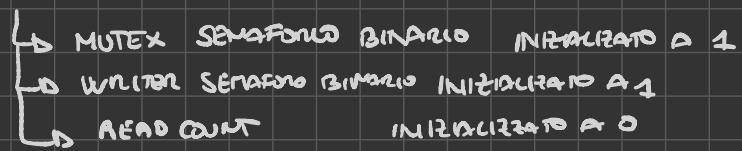
STARVATION  $\rightarrow$  I PROCESSI NON VENGONO MAI ESEGUITI PER CAPO DI ALTRI PROGRAMMI

## PROBLEMA PRODUZIONE - CONSUMAZIONE



24/10/2023

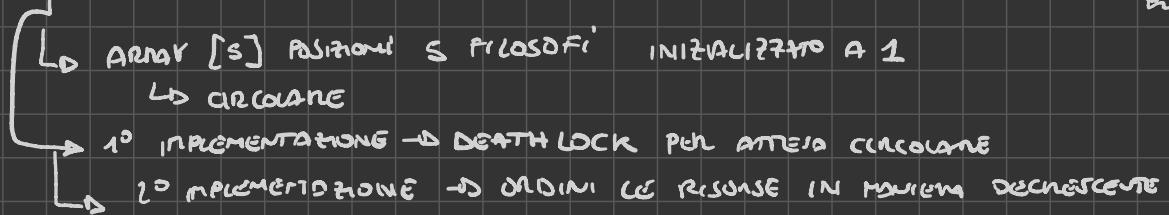
PROBLEMA READERS-WRITER PROBLEM → BASI DATI → LETTURA IN PARALLELO  
 ↳ REGOLAMENTARE SCrittura UNA VOLTA



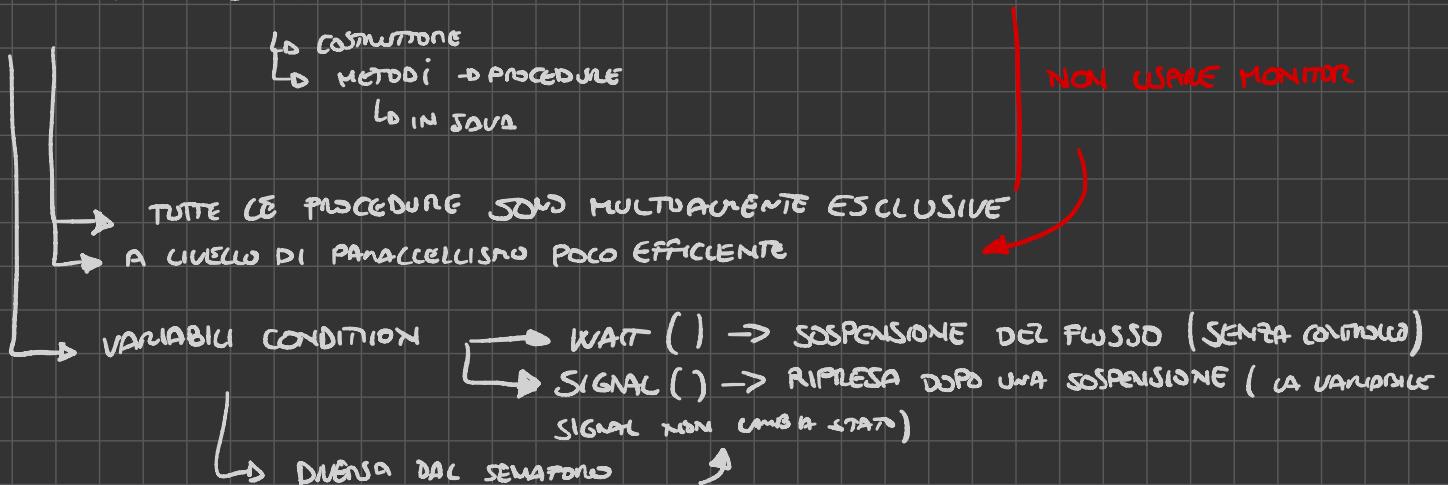
WRITER  
 WAIT (WAIT)

SIGNAL (WAIT),

PROBLEMA → LA CENA DEI 5 FILOSOFI → RISORSE SCARSE → risorse << flussi in sec.



MONITOR → OGGETTO → DI ALTO LIVELLO



31/10/2023

PROBLEMA DEL BARBIERE CHE DORME → NON CONSUMA CPU

↳ GESTIONE DI SERVIZI

↳ DEMONE

↳ ORDINE FIFO DELLA CODA DELLE RICHIESTE  
ORDINE DI POSTI DISPONIBILI CODA DI ATTESA N

2 SEMAFORI CONTATORE

↳ CUSTOMERS = 0  
↳ BARBIERE = 0

1 SEMAFORO BINARIO

↳ MUTEX

INTRO INIT NUMBER OF RESOURCES

BARBIERE CHE DORME PRIORITA' RIO

SOLARIS → CLOCK

↳ MUTEX ADATMUI  
↳ VARIABILI CONDIZIONI

WINDOWS → WASCHENDUE INTERRUPT

↳ SPIN-LOCK  
↳ CODE ATTESA

UNIX STANDARDS

→ PThreads → JAVA → REENTRANT-LOCK  
↳ VARIABILI CONDIZIONI

SCHEDULING DEI PROCESSI

MAXIMIZZARE METRICHE DI PERFORMANCE

↳ CPU INTENSIVE → BATCH (PROCESSI CHE NON INTERAGISCE CON L'UTENTE)  
↳ I/O INTENSIVE

BISOGNA cercare di alternare i processi CPU / I/O

LONG TERM - SCHEDULER CI INTERESSA POCO

PREEMPTION  
NON PREEMPTION

SHORT TERM - SCHEDULER → PUNTI DI SCHEDULI → 1 PROG. SWITCH DA RUNNING WAITING  
2 PROG. SWITCH DA RUNNING READY  
3 PROG. SWITCH DA WAITING TO READY  
4 PROG. TERMINA

DISPATCH  $\rightarrow$  ATTUAZIONE DELLA DECISIONE DELLO SCHEDULER

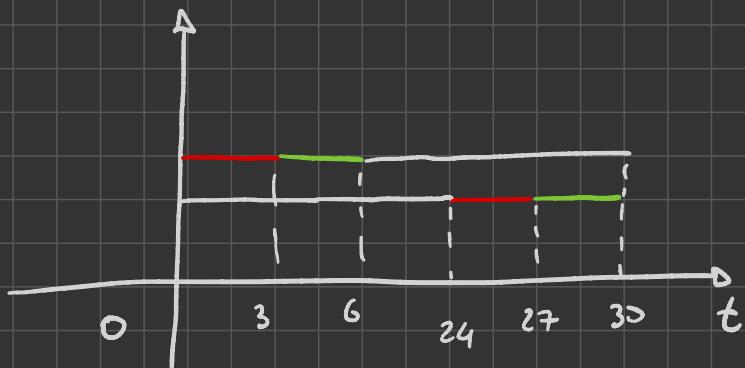
### OPTIMIZATION

Criteri di SCHEDULING  $\rightarrow$  ① CPU - OPTIMIZATION MAX

- ② THROUGHPUT  $\rightarrow$  NUMERO DI PROCESSI COMPLETATI IN UNA UNITÀ DI TEMPO MAX
- ③ TURNAROUND TIME  $\rightarrow$  TEMPO IN CUI È INIZIATO E FINITO IL PROGRAMMA MIN
- ④ WAITING TIME  $\rightarrow$  SOMMA IN CUI IL PROCESSO HA DURATO IN AREE DI QUEUE MIN
- ⑤ RESPONSIVE - TIME  $\rightarrow$  TEMPO IN CUI SI Vede IL CAMBIAMENTO

(NON È POSSIBILE OTTIMIZZARE TUTTI QUESTI PUNTI)

ALGORITMI DI SCHEDULING (FCFS) FIRST-COME-FIRST-SERVICE



$$\text{WAITING TIME } P_1 = 0 \quad P_2 = 24 \quad P_3 = 27$$

$$(0 + 24 + 27) / 3 = 17$$

$$P_2 = 0 \quad P_1 = 6 \quad P_3 = 3$$

$$(6 + 0 + 3) / 3 = 3$$

- LA SUA PERFORMANCE DIPENDE DALL' ARRIVO DEI PROCESSI

- EFFETTO CONUSCUO

ALGORITMO DI SCHEDULER (SJF) SCHEDULER-JOB-FIRST

PREEMPTIVE  $\rightarrow$  22 ms MIGLIORE  $\rightarrow$  ORARIO 5 ms  $\rightarrow$  NON FA NULLA

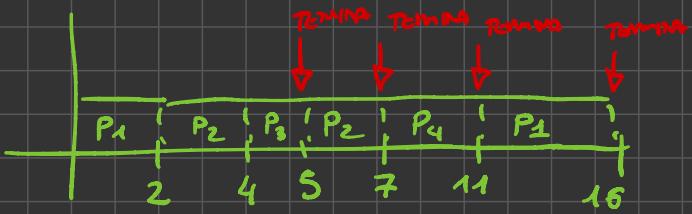
NON PREEMPTIVE  $\rightarrow$  22 ms MIGLIORE  $\rightarrow$  ORARIO 5 ms  $\rightarrow$  MANDA IN ESECUZIONE QUANDO DA 5 ms

(PROBLEMI STANNAZIONI)

PROCESS	ARRIVAL TIME
P <sub>1</sub>	0.0
P <sub>2</sub>	2.0
P <sub>3</sub>	4.0
P <sub>4</sub>	5.0

BURST TIME
7 (1)
4 (3)
1 (2)
4 (4)

PREGEMPTIVE  
NON PREGEMPTIVE



$$T_{\text{run}} - T_{\text{exec}} - T_{\text{arrivo}} = W_{TP_4} = 16 - 4 - 5 = 7 \quad \text{Avg. time } (7+6+3+0)/4 = 4$$

$$W_{TP_2} = 12 - 4 - 2 = 6$$

$$W_{TP_3} = 8 - 1 - 4 = 3$$

$$W_{TP_1} = 7 - 7 - 0 = 0$$

$$W_{TP_1} = 16 - 7 - 0 = 9$$

$$\text{Avg. time } (9+1+0+2)/4 = 3$$

$$W_{TP_2} = 7 - 4 - 2 = 1$$

OTTIMO FORMALE

$$W_{TP_3} = 5 - 1 - 4 = 0$$

$$W_{TP_4} = 11 - 4 - 5 = 2$$

PROBLEMI DI STARVATION  $\rightarrow$  RISOLTI CON PARAMETRI DI AGING

LENGTH OF NEXT CPU BURST  $\rightarrow$  PREVISIONI

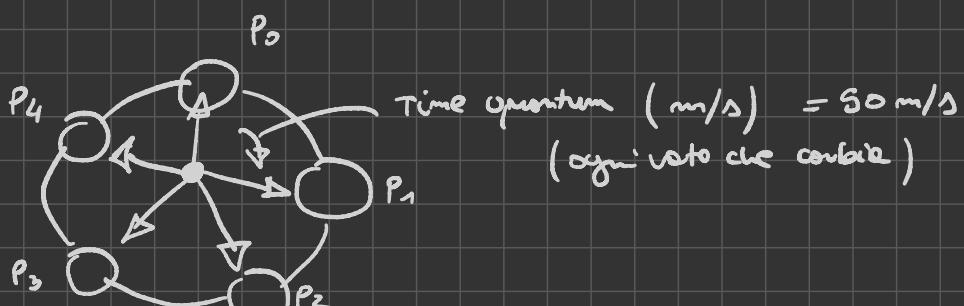
ANDAMENTO PROCESSI NON LINEARE  $\rightarrow$  NON CASUALE  $\rightarrow$  DIFFICILE DA PREDIRE  
 $\hookrightarrow$  USANDO EVENTO PRECEDENTE  $\rightarrow$  COMPLICATO

ALGORITMO PER PRIMA VOLTA  $\rightarrow$  STESSI PROBLEMI DI SJF  $\rightarrow$  STARVATION

$\downarrow$   
SOLUZIONE  
AGING

ALGORITMO DI SCHEDULING (RR) Run-Robin

Processi vengono messi in una lista circolare



$(m-1) * q = \text{ospetto quanto riprende la CPU}$   
↳ MOLTO UNCE N DI POSSIMO PREVEDERE L'ORARIO DI QUEE

Procom real-time

$q \rightarrow \text{longe} \rightarrow \text{FIFO}$  }  $q \rightarrow \text{piccolo} \rightarrow \text{cambio di contesto}$   $q \rightarrow \text{finora in maniera empirica}$

07/11/2023

RR with time quantum = 20

Procom Burst time

P <sub>1</sub>	53
P <sub>2</sub>	17
P <sub>3</sub>	68
P <sub>4</sub>	24



NON SI POTEVA RESETTARE IL TIMER INTERRUPT  
IN TEMPO PRECISE -> UNA VOLTA  
OGGI -> SI CAMBIA DINAMICAMENTE RUN-TIME

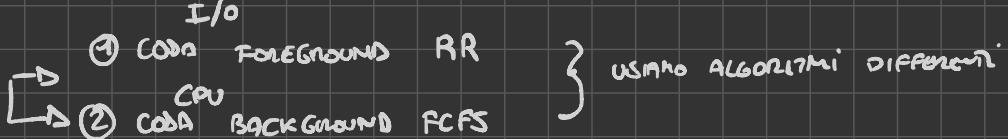
SOFT REALTIME -> CPU PERIODICA -> MEDIA -> AUDIO E VIDEO

BISOGNA USARE SOLO UN SOTTOINSIEME DI PROCESSI -> REAL TIME -> MOLTO EFFICIENTE

CPU INTENSIVE -> FCFS

AREA DI QUEUE POSITIONING

CODE MULTI PRO LIVELLO



↳ REGOLE SPOSTAMENTO DI PROCESSI NELL'E CODE -> SIA UPGRADE SIA DOWNGRADE

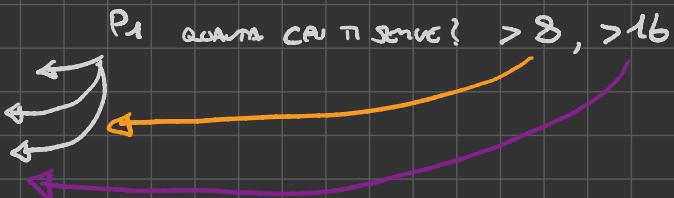
↳ LINUX 256 CODE

MULTI LIVELLO E FEEDBACK QUEUE -> TUTTE LE CARATTERISTICHE PRECEDENTI

↳ Dove entrano i processi nuovi?

↳ 3 QUEUE

RR 8 ms  
RR 16 ms  
FCFS

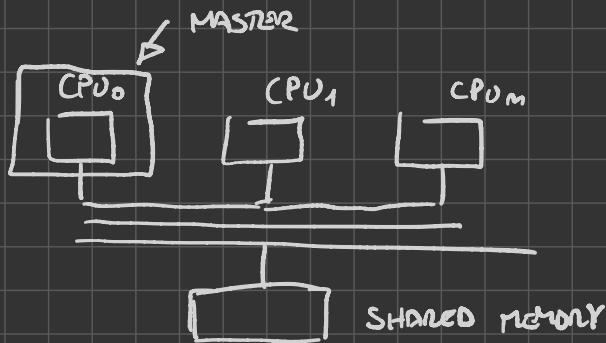


I PROCESSI POSSONO CAMBIARE PRIORITÀ SE DA FCFS PRENDERE POCO CPU PUÒ RISULTARE LE CODE

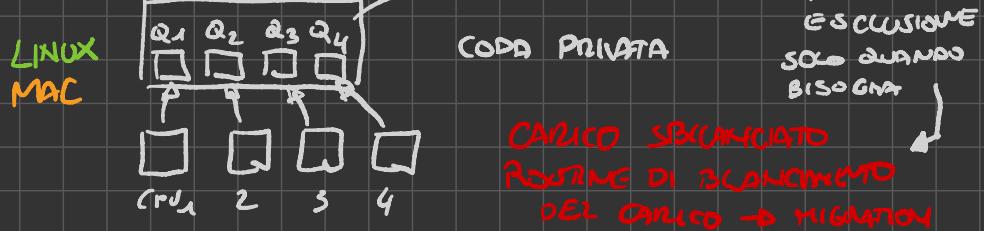
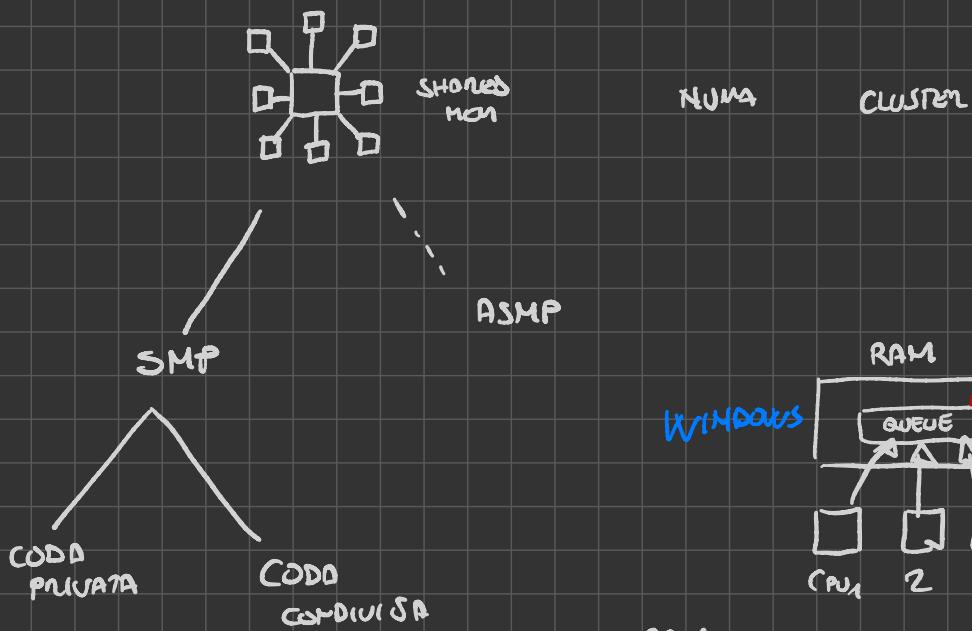
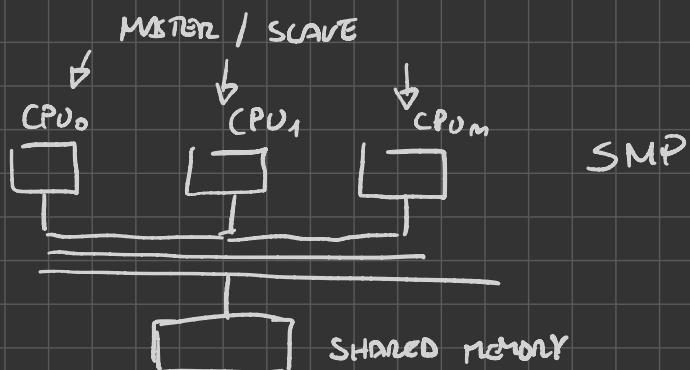
## SCHEDULING DI UN SISTEMA CON MOLTI PROCESSORI

→ SISTEMA OPERATIVO, processi

- ↳ ASMP → 1 CORE → MASTER → TUTTI GLI ALTRI SLAVE
- ↳ SMP → TUTTI FANNO UN PÒ DI TUTTO
- ↳ PROGRAMMING CLASSIC.



ASMP → PER IL CALCOLO NON GENERALE  
PUNTA SU



## BILANCIMENTO DEL CARICO

PULL MIGRATION → CODE PRIVATE → QUANDO IL PROCESSORE E' IN APOD

PUSH MIGRATION

LO SENZA CAUSO

↳ DEMONE → CARICO DEL PROCESSORE

SI SCEGLIE IN BASE AL NEXT BURST TIME → CPU INTENSIVE PIÙ AFFINI AL PUSH / PULL

HARD AFFINE → NON SI SPosta DALLA CPU IN CUI È STATO PRESSO

I CPU INTENSIVE SONO PIÙ AFFINI ALLA PULL - PUSH MIGRATION

14/11/2023

## SCHEDULER LINUX

→ PROCESS DESCRIPTOR → PROCESS CONTROL BLOCK (PCB)

↳ FONDAMENTALE PER I NOSTRI ALGORITMI DI SCHEDULER

↳ CODE DOPPIAMENTE LINKATE

→ RUNNING → PROCESSI AREA DI CUEQUE

↳ INTERRUPTIBLE → PROCESSO IN ATTESA

↳ UN INTERRUPTIBLE → PROCESSO IN ATTESA NON INTERRUPTIBILE

→ ZOMBIE → PROCESSI QUANDO TERMINANO NON HANNO IL PADRE → INIT SI OCCUPA DI METTERLI IN STOPPED

→ STOPPED → PROCESSI FERMATI

→ TIME-SHARING → CPU → BENE PREZIOSO → SUDDIVIDE IN FETTE IL TEMPO (SLICES)

→ DYNAMIC → PRIORITATING (PAGING)

→ DISTINGUE TRA

↳ REAL-TIME

↳ E NON REAL-TIME

→ PRIORITÀ ALL' I/O INTENSIVE → IL SO CI METTE UN PO' A CAPIRE IL COMPORTAMENTO DEL PROGRAMMA

→ GRAN NUMERO DI SYSTEM CALL PER CAMBIARE IL COMPORTAMENTO DELLA MACCHINA

↳ SALVATE IN FILE DI CONFIGURAZIONE

## TIME-SHARING ( SLICES )

TIME QUANTUM → ASSEGNATO AD OGNI PROCESSO → MULTIPLO TIME-INTERRUPT

↳ OGNI PROGRAMMA AD OGNI ENTRATA IN CPU CONSUMA IL TIME QUANTUM → FINITO IL TEMPO

↳ NON STARVATION

↳ PERIODICAMENTE VIENE RIASSEGNAUTO ( SCADE UN'EPOCA )

↳ VARIA OGNI VOLTA

↳ NON UGUALE A TUTTI → QUANTUM + A I/O INTENSIVE

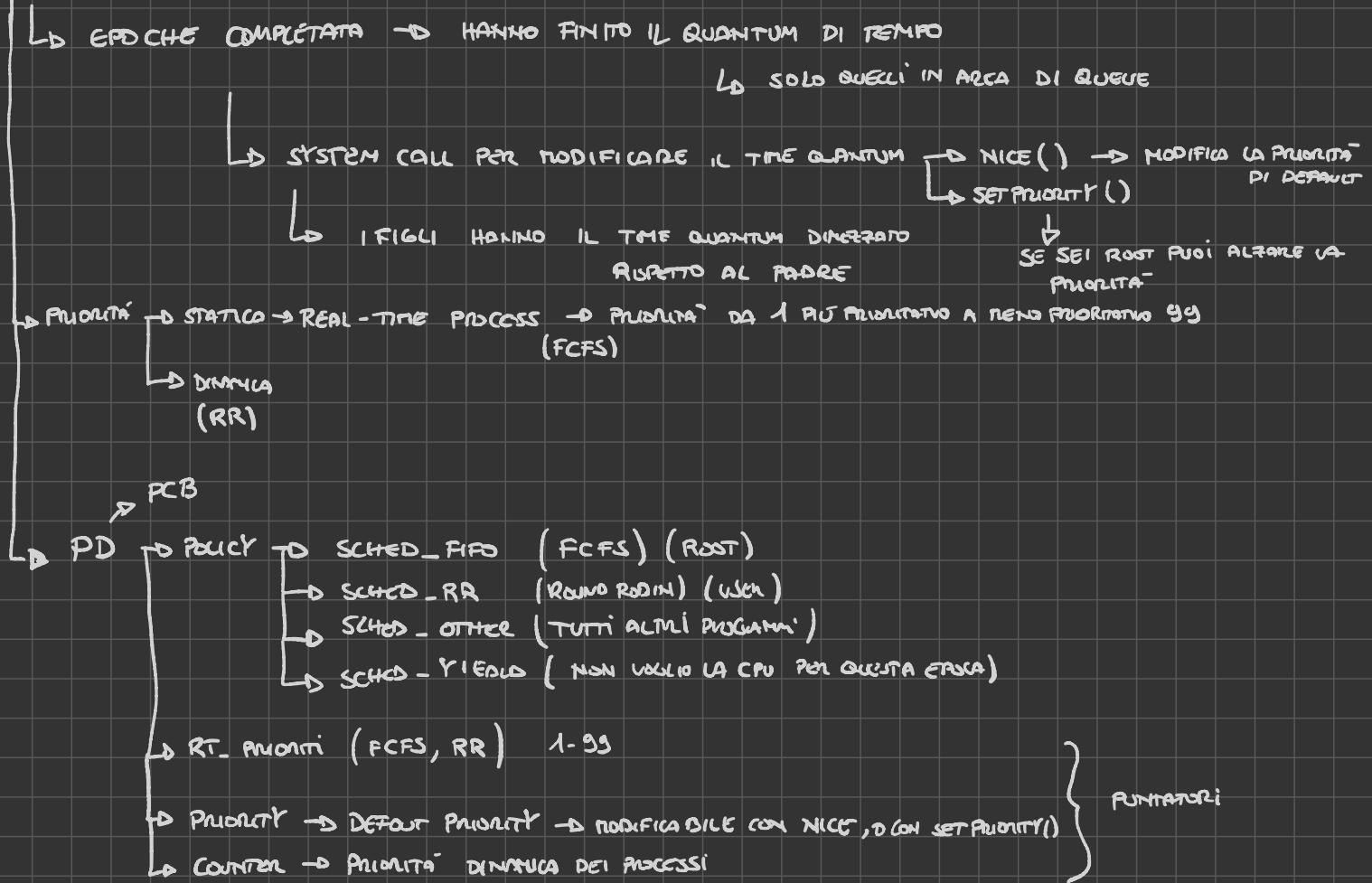
↳ QUELLI CHE NON SI TROVANO NELL' AREA DI CUEQUE DI ATTESA  
CHE HANNO FINITO IL TIME QUANTUM ( Oltre quelli che non  
hanno il time quantum a 0 )

I/O INTENSIVE AVANTAGGIATI RISPETTO AD CPU INTENSIVE

PRIMA EPOCA CRITICA → CPU INTENSIVE CONSUMANO SUBITO TUTTA IL LORO TIME QUANTUM

I/O INTENSIVE NON INTERATTIVI → RETE BACKGROUND ( SCARICARE PARCHETTI DALLA RETE )

## SCHEDULER 2.4.X



FUNZIONE DI SCHEDULE run-queue → ASSEGNA CPU → TRAMITE LA GOODNESS

↳ OGNI VOLTA CHE VI È UNA (system call, I/O, semaphore)



$C = goodness(prv, p)$

↳ process p  
↳ process prime

$C = -1000$  run-queue è vuota non contiene nemmeno process → FULL MIGRATION

$C = 0$  ho terminato il quanto di tempo

$0 < C < 1000$  processi convenzionali CPU intensive

$C \geq 1000$  processi I/O intensive (run-time)

Funzione scheduler → rimuovere in ogni punto dello scheduling

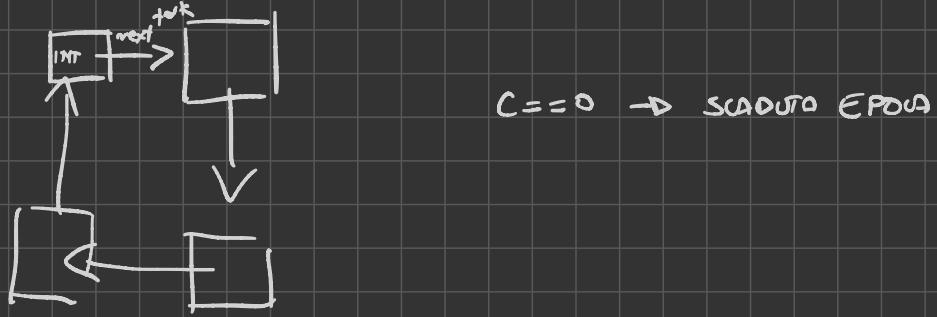
prv → ultimo processo in CPU

controlli

↳ se c'è un RR e ha terminato il quanto di tempo, gli ridei il tempo e lo sposta in fondo alla coda circolare

↳ se arriva un segnale e un programma ha ottenuto lo metto in running

↳ se non c'è in running ti rimuovi dalla lista di queue



21/11/2023

LINUX SCHEDULER

- ↳ + I/O INTENSIVE
- ↳ - CPU INTENSIVE

QUANTUM TIME → TEMPO IN CUI IL PROGRAMMA PRENDE LA CPU

CODE (RR, FIFO) → NON VENGONO VALUTATI

EPOCA → TUTTI I PROCESSI NELLA CODA DI QUEUE SONO TERMINATI

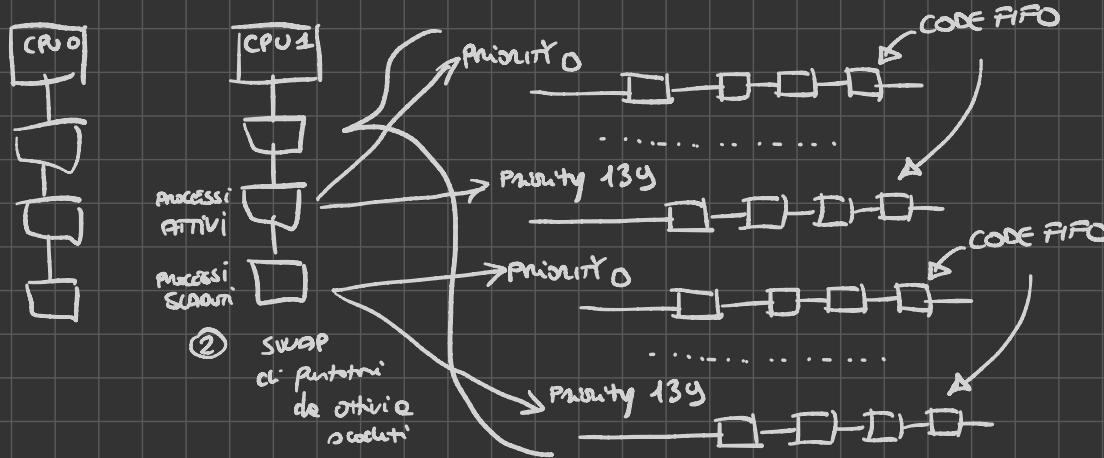
GOODNESS → COMPARA 3 CICLI ALLO SCADERE DELL'EPOCA  $O(m)$

PROBLEMI

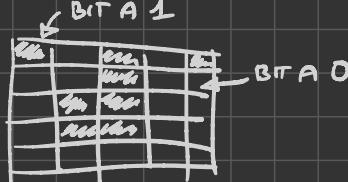
- ↳  $O(m)$  ricerca della GOODNESS () ①
- ↳ I/O intensive in background (database, Network)
- ↳ ERICA VARIABILE ②

2.6.X SCHEDULER

- ↳ UPGRADE →  $O(1)$  complexità costante
- ↳ SMP CODE PRIVATE
- ↳ COMPLICAZIONE SCHEDULER
- ↳ IMPLEMENTAZIONE DI FUNZIONI PIÙ INTROVABILI CON LE CODE (di ogni processo)



Priorità BITMAP (0 - 139) CODE

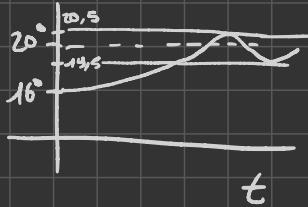


ISTRUZIONE MACCHIA DATA UNA STRINGA DI BIT  
IN UN CICLO DI CLOCK RIDA IL PRIMO 1

## COMPLETELY FAIR SCHEDULER (CFS)

### FAIRNESS

- SMP
- CORE PRIVATE
- VALORE NUMERICO CHE TIENE TRACCE DI OGNI CYCLE DI CLOCK PER OGNI PROCESSO (Virtual time) → a livello mono-secondi
- SOGLIA DI ISTANZI



modulare il cambio di contesto

→ PREEMPTION TIME E' VARIABILE

### BILANCIMENTO DEL CARICO

- cacheflush\_time  $\left[ \frac{\text{cache size\_KB} \cdot (\text{cpu\_clock\_in\_ns})}{5000} \right]$
- now si sposta se il TEMPO DI CPU < cache flush
- PUSH - DEMONE
- PULL - CORE IN RIDUCE

28/11/2023

ALLOCAZIONE MEMORIA SISTEMA OPERATIVO  $\rightarrow$  MMU  $\rightarrow$  GESTIONE LE MEMORIE

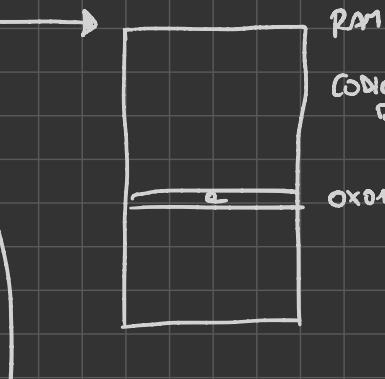
LD CIP

↳ BINDING DELLE ISTRUZIONI IN MEMORIA

↳ CONVERSAZIONE DA INDIRIZZI

ES  $i = 0;$   $\rightarrow$  CONVERSAZIONE IN CELLA DI MEMORIA FISICA  
0x00001

- TEMPO COMPILAZIONE
- TEMPO CARICA/MENTO
- TEMPO DI ESECUZIONE



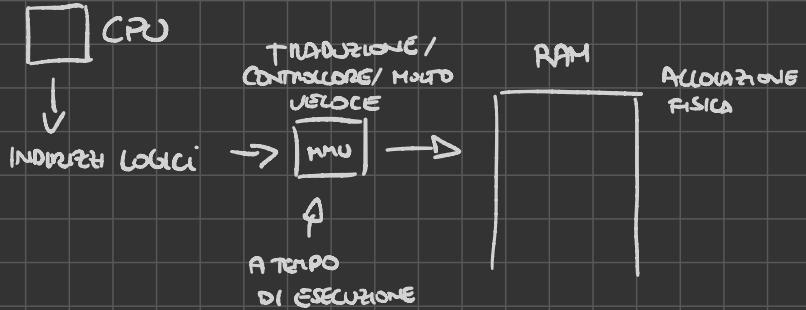
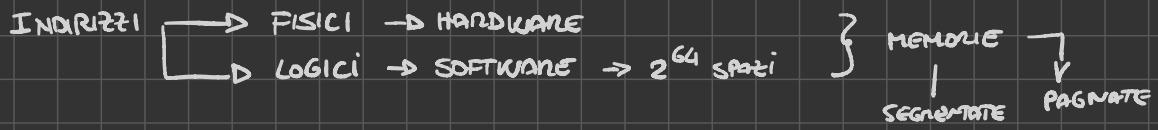
CODICE ASSOLUTO SI ALLOCA LA MEMORIA DOVE DICE IL COMPILATORE, SENZA MMU  
TUTTA LA MEMORIA DISPONIBILE  
AL PROGRAMMA, NON SI POSSONO DARE PIÙ PROGRAMMI.  
ALTRIMENTI SI PANNEGGIANO

AVVIENE SOLO AL MIGRAMENTO DEL PROGRAMMA E IL SISTEMA OPERATIVO A SCEGLIERE, MA UNA VOLTA NON SI SPosta  
@ base + 0x00000000 SI SCEGLIE UN'UNICA BASE ESI FA L'OFFSET.

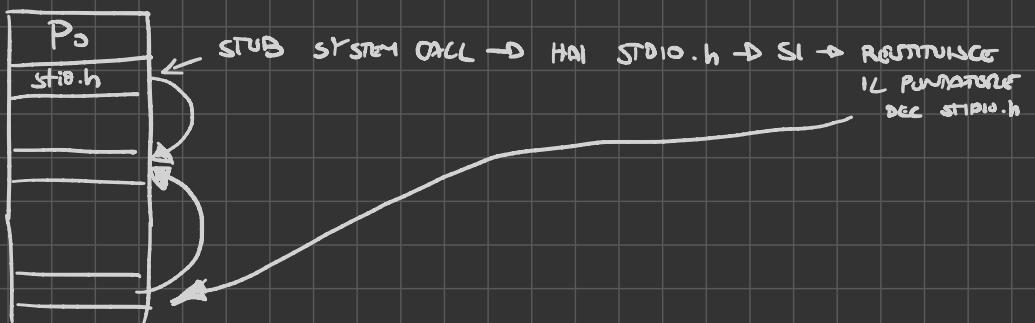
SOLO LA PRIMA VOLTA IN CUI ACCEDO ALLA VARIABILE LA ALLOCDO NELLA RAM

- ↳ POSSO CARICARLO IN QUALSIASI POSTO
- ↳ PUÒ ESSERE MUOSO IN QUALSIASI POSTO
- ↳ SI PUÒ ALLOCARE LA MEMORIA ANCHE NON IN MODO CONTIGUO

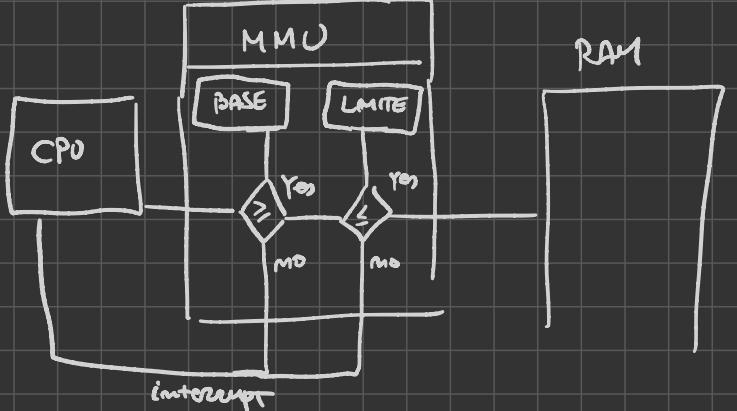
0x00000000 punti indirizzo di memoria disponibile



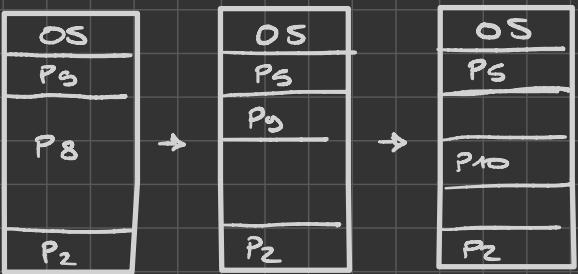
Quando ogni programma carica DINAMICAMENTE porzioni di programma in base alle sue funzioni



GESTIONE MEMORIA RAM IN MANIERA CONTIGUA  
 → REGISTRO BASE  
 → REGISTRO LIMITE



STATISTICAMENTE SE SPARTIAMO QUESTO MECCANISMO  
 EFFETTUEREMO UN MECCANISMO → PAGINAZIONE



TI DIVIDE LA MEMORIA A METÀ → ESTERNA  
 → INTERNA

Algoritmo di allocazione memoria contigua

- FIRST-FIT
  - BEST-FIT
  - NEXT-FIT
  - WORST-FIT
- ≥ 50% DI OVERHEAD  
 ↓  
 DEFRAGMENTAZIONE

PAGINING → CON DIMENSIONE PAGINA DI 2

→ TIRADURIONE → TABELLA DELLE PAGINE

PIAGGIBILITÀ 1KB ←

A = 12 KB  
 B = 7 KB

4 KB	E	A
4 KB	E	B
4 KB	E	A
4 KB	E	B
4 KB	E	A



12/12/2023

TABELLE → TABEGLI DIRETTA → ALLOCAMENTO DI TUTTA LA MEMORIA

↳ GENERATICA → NON PIÙ IN USO

↳ HASHED → 12 GATE → 12 INDIRIZZI → FUNZIONE HASH → GENERA TOKEN

GRANDE VANTAGGIO

↳ PUÒ GENERARE COLLISIONI

STESMO NOME

COMPLESSITÀ  
O( $n$ ) LISTA DI COLLISIONI

→ SU TUTTO IL SISTEMA

PID + PAGE NUMBER



HASH TABLE



PID PAGE OFFSET

↳ INVERTED PAGE TABLE

↳ INDICIZZATA NEL

NEL PAGE NUMBER, MA NEL

FRAME NUMBER, HANNA TABERCA NOLO RIDOTTA

↳ L'ACCESSO NON È DIRETTO, BISOGNA  
SCORRIRE TUTTI I FRAME NUMBER QUINDI

O( $m$ )

↳ CREAZIONE DI LISTE PER LA CONDIZIONE DI SURGENZA'

## SEGMENTAZIONE

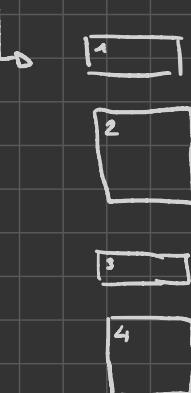
SEGMENTI LOGICI → DI OGNI PROGRAMMA → - MAIN

- PROCEDURE

- FUNZIONI

- METODI

↳ BLOCCO DI DIMENSIONE VARIABILE



→ ALLOCATOR METTE LE PAGINE PIÙ CONTINUE POSSIBILI

↳ SEGMENTAZIONE + PAGINAZIONE → DOPPIA TRADUZIONE

PENTIUM → CPU → INDIRIZZO LOGICO → SEGMENTATION UNIT → LINEAR ADDRESS → PAGINAZIONE

↓  
INDIRIZZI  
FISICI

↓  
Memoria  
FISICA

AMD64 → USANDO 48 bit → CON CAPACITÀ DI ESPANSIONE

LE INDIRIZZI CANONICI, NON SI MODIFICA LA  
CATENA DEL SOFTWARE