# Prelazione

Quando è necessario assegnare la CPU a un altro processo?
Il processo può passare da:

1) Running → Waiting
2) Running → Ready
3) Waiting → Ready
4) Running → Termina

(2) e (3) = prelazione

Shortest Job First no prelazione ma la sua variante Shortest Remaining Time First ne fa uso

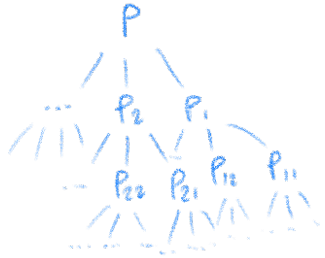
## ALBERO DEI PROCESSI

- Processo Padre produce Processo Figlio tramite FORK
- Processo Figlio termina
- Processo Padre ispeziona le info sulla sua terminazione (EXIT/ABORT) — va in WAIT
- Le info sulla terminazione del processo figlio vengono rimosse

finchè rimangono il processo figlio rimane come "ZOMBIE"

Processi zombie sono responsabili di memory leak, esempio banale:

```
P: { while (1)
       fork();
   }
```



SISTEMA SW ARTICOLATO SU TANTI PROCESSI

DATI $\Longrightarrow$ $P_1$ $\Longrightarrow$ $P_2$ $\Longrightarrow$ $P_3$ $\Longrightarrow$ RISULTATO

Sistema di Pipeline, in questo caso più processi collaborano aumentando le prestazioni, alcuni processi possono lavorare in parallelo

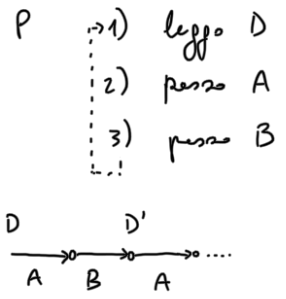
Processi cooperanti (a differenza dei processi competitivi)

Modelli di comunicazione fra processi

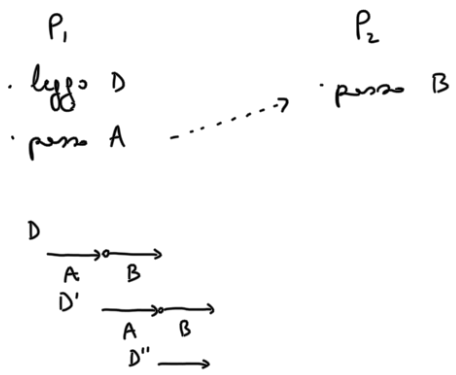
1) A Memoria Condivisa
2) A Scambio di Messaggi

es:

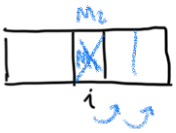
SOLUZIONE MONOLITICA




SOLUZIONE A PROCESSI COOPERANTI




BUFFER DI MEMORIA CONDIVISO




il processo produttore deve aggiornare il suo puntatore a una cella vuota del buffer

| Prod 1 | Prod 2 |
|---|---|
| $M_1$ = DATO | $M_2$ = DATO |
| DATO [i] = $M_1$ | DATO (i) = $M_2$ |

Supponiamo che per via del Round Robin Prod 2 non faccia in tempo ad aggiornare i, $M_1$ salvato da Prod 1 viene sovrascritto da $M_2$ i viene aggiornato sia da Prod 1 che Prod 2 e si salta quindi una cella di memoria (che verrà considerata piena da un Consumatore

Il processo di MUTUA ESCLUSIONE fa sì che il buffer venga utilizzato da un solo processo alla volta


- Lo scambio di messaggi si basa su una coda di messaggi, che può essere visto come una memoria condivisa

SEND & RECEIVE DIRETTE

$P_1 \xrightarrow{mex} P_2$ serve che $P_1$ conosca il PID di $P_2$ e viceversa (processo identifier)

processi fortemente accoppiati

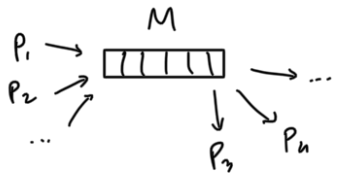
send(m, $P_2$)
receive(m, $P_1$)

$P_1 - P_2$ ← canale di comunicazione

SEND & RECEIVE INDIRETTE

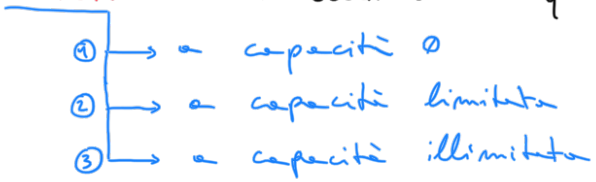
Si utilizza una mailbox come intermediario

send(m, M)
receive(m, M)




processi debolmente accoppiati

BUFFER caratterizzati dalla quantità di messaggi che possono contenere




① Il mittente mantiene il messaggio e rimane fermo finché il processo ricevente non lo recepisce INVIO SINCRONO

② Array di dimensione predefinita. Nel caso il buffer sia occupato e un processo vuole fare il send() di un messaggio si possono implementare due soluzioni

SEND → SINCRONA: si aspetta che si liberi posto
SEND → ASINCRONA: non potendo andare a buon fine termina

RECEIVE → SINCRONA: aspetta che arrivi il messaggio
RECEIVE → ASINCRONA: se non arriva prosegue l'esecuzione (necessario un sistema di POLLING)

SEND SINCRONA + RECEIVE SINCRONA → RANDEZVOUS

## IMPLEMENTAZIONE PER LO SCAMBIO DI MESSAGGI

- SOCKET: CLIENT- SERVER, manda richieste, eroga il servizio
- REMOTE PROCEDURE CALL (RPC)
  Il codice nasconde uno STUB che individua la porta (tramite un match maker) sulla macchina sulla quale deve essere eseguita la procedura
  messaggi strutturati a differenza di quelli a basso livello dei SOCKET
- PIPE → ANONIMA: canale di comunicazione simplex (unica direzione), ordine FIFO
  → CON NOME: duplex, al termine del processo la pipe sopravvive al creatore, a differenza delle pipe anonime