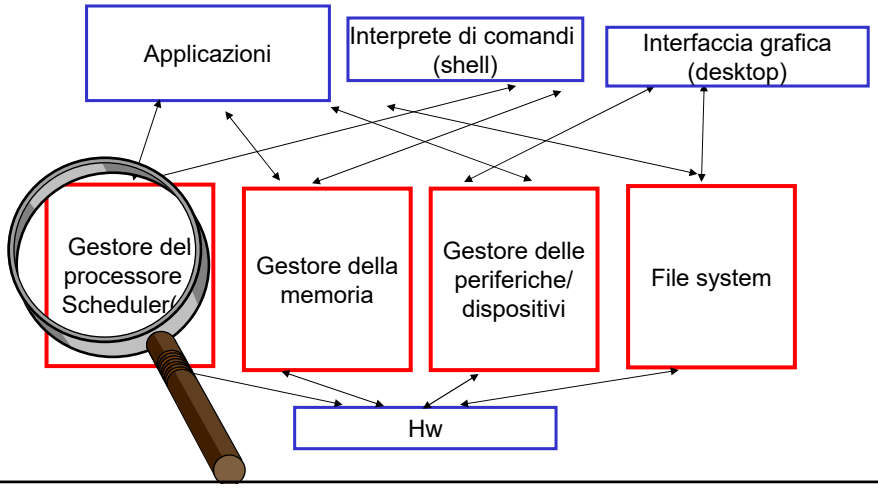
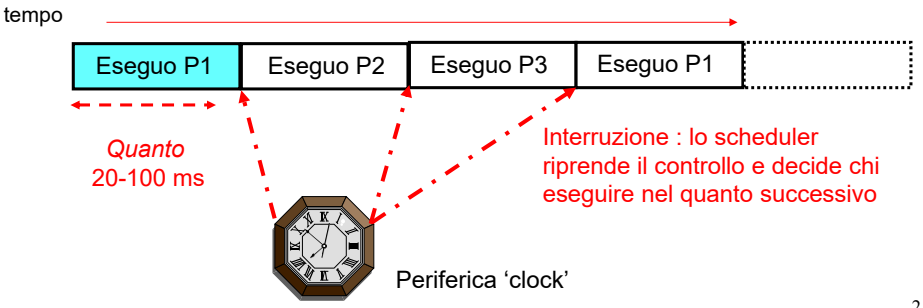
Organizzazione di un SO monolitico



1

Condivisione della CPU (scheduler)

- Lo scheduler (di basso livello) si preoccupa di distribuire il processore fra i processi in maniera trasparente
  - es : 3 processi P1, P2, P3, vengono mandati in esecuzione ciclicamente



2

BOOTSTRAP

- All'accensione della macchina, l'hardware lancia il programma di boostrap …
- … che carica in memoria il blocco di boot da disco, il quale contiene un programma…
- …. che carica in memoria il kernel …
- … e quindi trasferisce il controllo a un entry point (start), che crea il processo 0

3

Creazione dei processi (es:Unix)

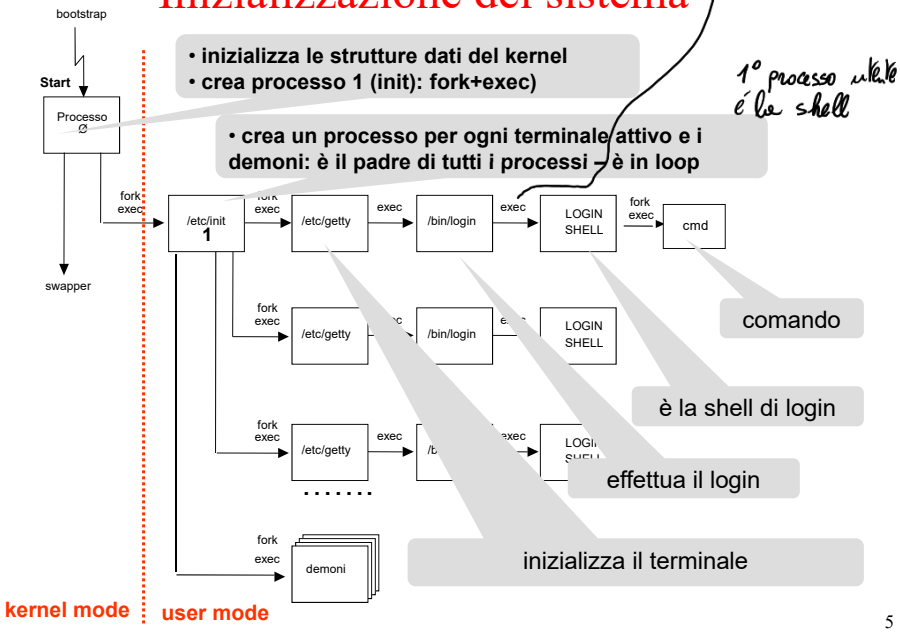
Alla fine del *bootstrap* viene creato un primo processo "init" di inizializzazione, il quale provvede a creare un processo di "login" per ogni terminale, ed un certo numero di processi che eseguono in "background" (per esempio quello per gestire le richieste di stampa, ecc.), che vengono chiamati "daemon" e che sono sempre pronti ad intervenire quando si verificano determinati eventi.

Quando un utente fa il login sul sistema, viene creato un nuovo processo, un interprete di comandi "shell" , per accettare richieste fornite dall'utente in modo interattivo attraverso opportuni comandi.

Quando l'utente chiede di eseguire un programma (per esempio il gcc per compilare un programma in C), l'interprete di comandi crea un nuovo processo che eseguirà il programma gcc. Intanto l'interprete di comandi si metterà in attesa della *terminazione* di tale processo: A questo punto la shell riprende la sua esecuzione, chiedendo un nuovo comando all'utente.

4

Inizializzazione del sistema



5

Creazione dei processi (esempio di shell)

```
while (TRUE) {                                  /* repeat forever */
   type_prompt( );                              /* display prompt */
   read_command (command, parameters)           /* input from terminal */

if (fork() != 0) {                              /* fork off child process */
   /* Parent code */
   waitpid( -1, &status, 0);                    /* wait for child to exit */
}
else {
   /* Child code */
   execve (command, parameters, 0);             /* execute command */
 } }
```

6

Terminazione di un processo

- Un processo termina :
  - Quando conclude correttamente la sua esecuzione attraverso una istruzione esplicita di terminazione
  - Quando c'è un errore che non gli permette di proseguire (es. il file di input non esiste)
  - Quando effettua una operazione illecita
    - es. cerca di accedere alla memoria privata di altri processi
  - Quando viene "ucciso" da un altro processo (attraverso una opportuna system call, per es. kill(pid)).

- In tutti questi casi il processore ricomincia automaticamente ad eseguire il sistema operativo ad un indirizzo prefissato

7

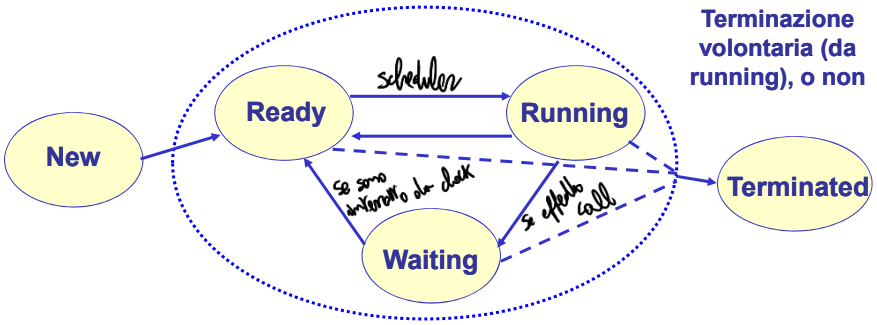
Interruzione momentanea di un processo

- Un processo in esecuzione può cedere il processore al SO nei due casi seguenti:
  - quando il processo stesso esegue una chiamata di sistema per richiedere un servizio da parte del SO (ad esempio per effettuare una operazione di I/O)
  - quando arriva una interruzione hw (ad esempio un'interruzione proveniente da una periferica di I/O o dal clock di sistema)

Periodicamente arriva un'interruzione al processore. Qualunque cosa stia facendo arriva e valeva decide chi sarà il processo running. Passa al mio turno

8

Stati di un processo

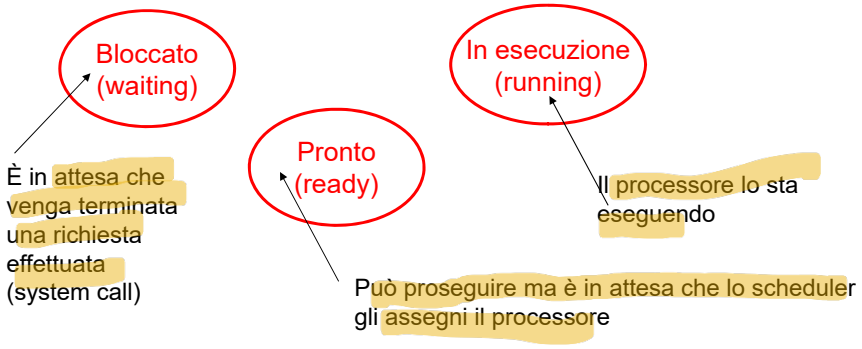


Nuovo: durante la creazione (inoltre potrebbe essere tenuto in "anticamera" prima di allocargli risorse, per limitare il livello di muiltiprogrammazione)

Terminated: il processo è terminato ma il s.o. ne tiene ancora traccia (può servire per informazioni sulla terminazione)

9

Stati di un processo

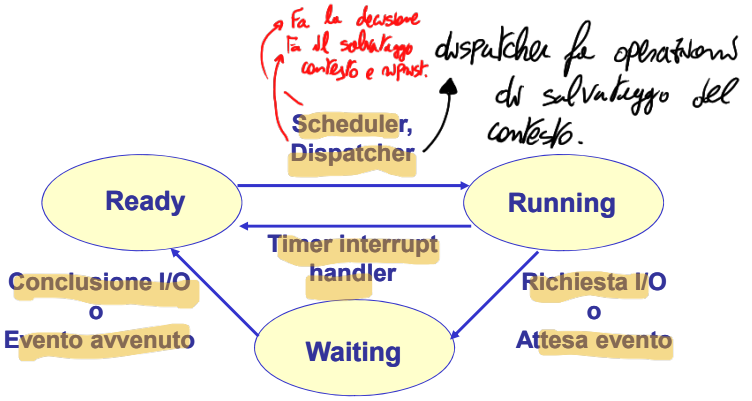


- Come si passa da uno stato all'altro ?

Solo i processi ready possono passare a running !

10

Stati di un processo (transizioni di stato)

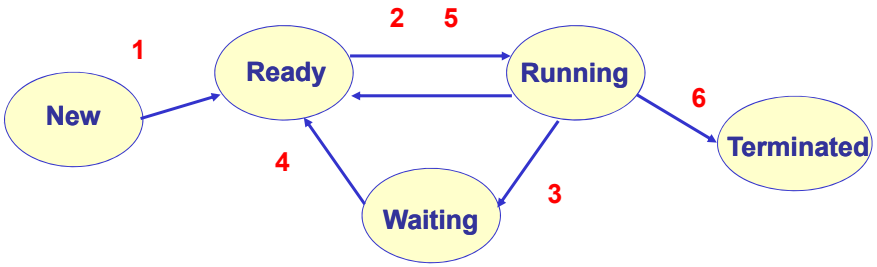


11

Thank you

Esempio: passaggi di stato di un processo P

Eventi	Stati del processo P dopo l'evento
1. creazione	new, ready
2. scelto dallo scheduler	running
3. P chiama read()	waiting
4. interrupt (concluso I/O)	ready
5. P scelto dallo scheduler	running
6. P esegue exit()	terminated



12

Scheduler e Dispatcher

Scheduler: è quel componente del sistema operativo che si occupa di scegliere quale processo deve diventare running tra tutti quelli attivi (la scelta viene attuata in base ad una *politica di scheduling*)

Dispatcher: è quel componente del sistema operativo che attua il *context switch* (caricando nei registri lo stato del processo scelto dallo scheduler per diventare running)

Osservazione: il dispatcher implementa un MECCANISMO (che rende running un processo) mentre lo scheduler implementa una POLITICA (sceglie fra diversi processi ready to run quale far diventare running) ed utilizza un meccanismo (realizzato dal dispatcher) per rendere la scelta operativa.

13

Implementazione di processi (1)

- Le informazioni relative a tutti i processi attivi ad un dato istante sono mantenute nella *Tabella dei processi*:
  - un array di strutture (record)
  - una struttura per ogni processo
    - terminologia : la singola struttura può essere denominata anche PCB (Process Control Block)

14

Implementazione di processi (2)

- Il PCB contiene tutte le informazioni sul processo diverse dal suo spazio di indirizzamento
  - valore PC, SP, PSW, registri generali
  - stato (pronto, bloccato …)
  - informazioni relative ai file che il processo sta utilizzando

15

Implementazione di processi (3)

- Il PCB contiene tutte le informazioni sul processo diverse dal suo spazio di indirizzamento (cont.)
  - informazioni relative alla RAM occupata dal processo
    - es : valore dei registri base e limite …
  - altre informazioni dipendenti dal particolare SO
    - es. quantità di tempo CPU utilizzato di recente (algoritmi di scheduling), informazioni legate a meccanismi di IPC (es. segnali Unix)

16

Strutture dati del sistema operativo per la gestione dei processi

La tabella dei processi contiene l'insieme dei PCB di tutti i processi presenti nel sistema.

Contenuti tipici del PCB:

Process management	Memory management	File management
Registers	Pointer to text segment	Root directory
Program counter	Pointer to data segment	Working directory
Program status word	Pointer to stack segment	File descriptors
Stack pointer		User ID
Process state: Ready, Running, Waiting		Group ID
Priority		
Scheduling parameters		
Process ID		
Parent process		
Process group		
Signals		
Time when process started		
CPU time used		
Children's CPU time		
Time of next alarm		

Registri CPU

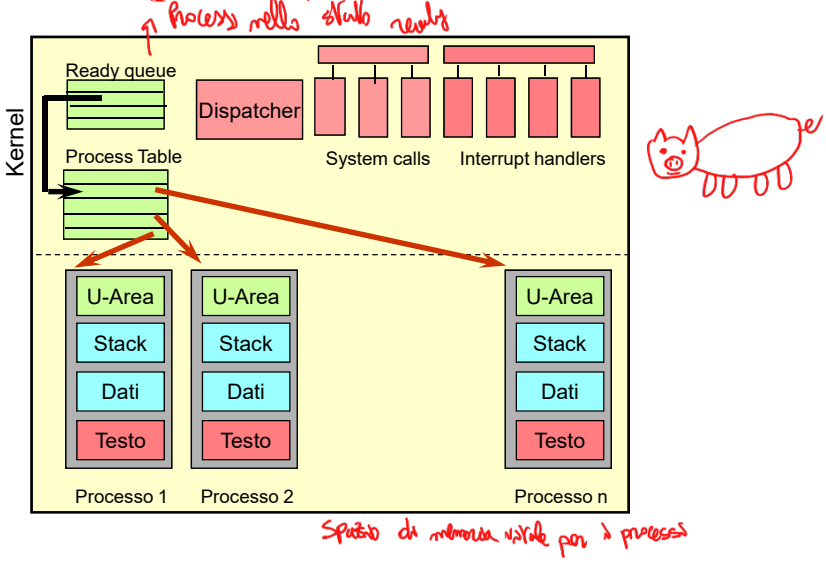
Dati per lo scheduler

Dati per accounting

Informazioni utili per protezione (controllo accesso ai file)

17

Modello (parziale) di riferimento di un S.O.



Lo schedulatore privilegia processi che alternano CPU e I/O

Struttura dati per i process ready e info subito disponibili nella parte residente nella tabella dei process

Strutture dati per la gestione dei processi

Process Table

- si trova nel kernel, ed è residente
- contiene una entry per ogni processo, ed è dimensionata staticamente al momento della configurazione del sistema
- per ogni processo contiene le informazioni che ne permettono la schedulazione, e che devono essere sempre residenti

U-Area (user area)

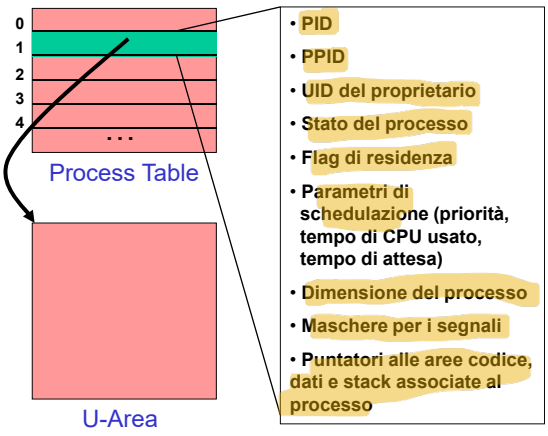
- contiene quelle informazioni necessarie al kernel per la gestione del processo, ma che non è necessario che siano sempre residenti in memoria

Ready Queue

- Liste dei processi ready (una per ciascun livello di priorità)

19

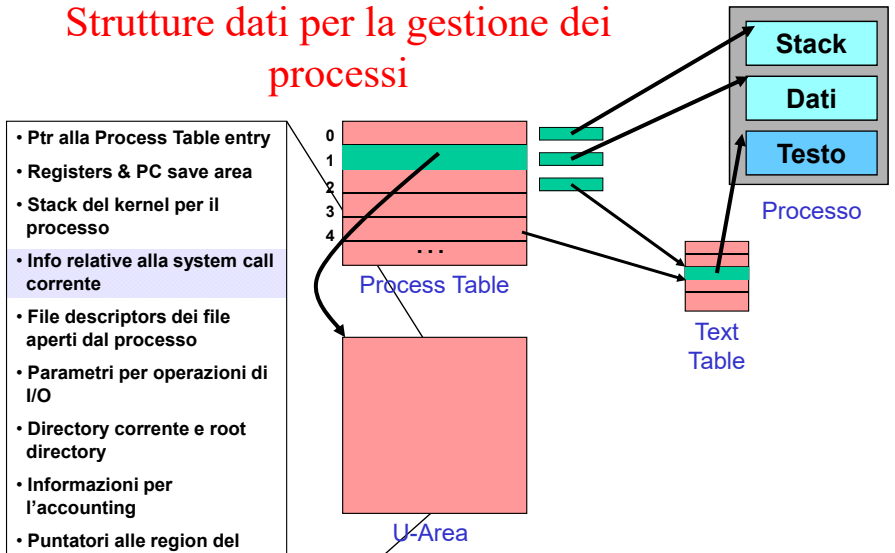
Strutture dati per la gestione dei processi



20

Strutture dati per la gestione dei processi

- Ptr alla Process Table entry
- Registers & PC save area
- Stack del kernel per il processo
- Info relative alla system call corrente
- File descriptors dei file aperti dal processo
- Parametri per operazioni di I/O
- Directory corrente e root directory
- Informazioni per l'accounting
- Puntatori alle region del processo
- ....

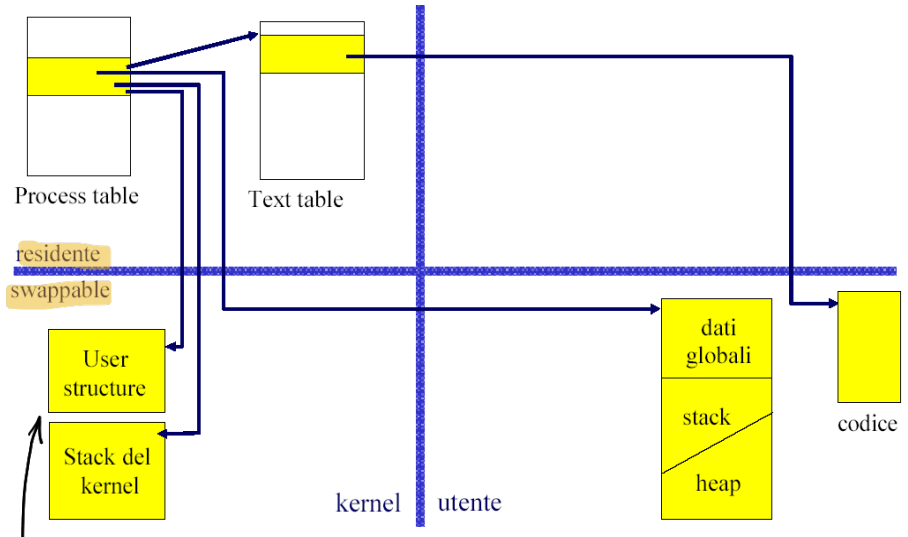


21

Ci sono informazioni per quanto un sistema è running

Una parte nella parte residente, altre le riporta solo un fase running

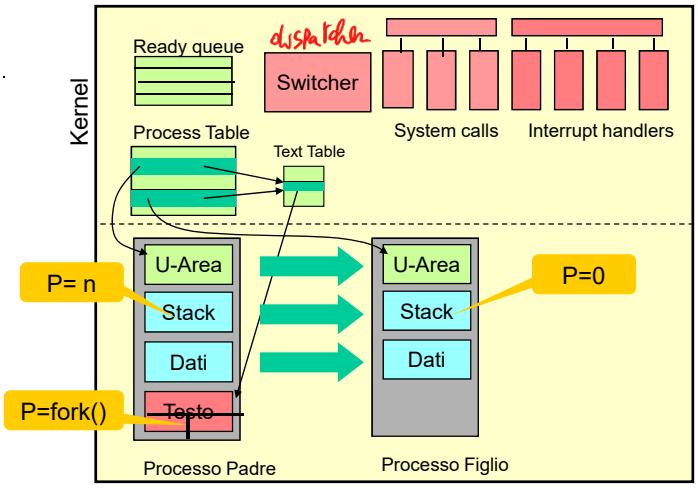
Immagine in memoria di un processo



Strutture dati di gestione | strutture accessibili da kernel o utente

Process (u-Area, process control block)

Creazione di un processo (fork)



SO crea processo con stesso Process Control block (quasi lo stesso: nella parte di gestione ho stesso utente, padre e figlio, PID diversi, PC sono uguali perché senza exec l'istruz. successiva alla fork è gestita da entrambi.

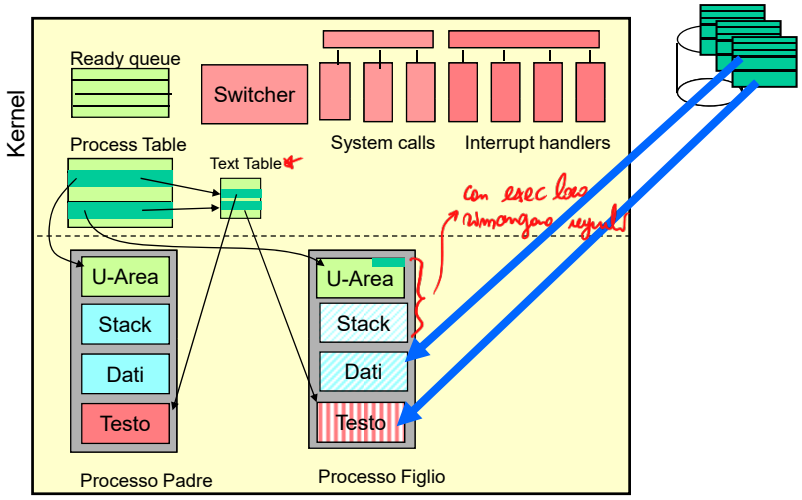
Così come contenuto area stack è lo stesso, ma sono in uno spazio di memoria nuovo, il codice non ha senso copiarlo) Area stack è uguale perché ho stesso percorso di chiamate a funzione.

Che cosa fa il S.O a seguito di una fork

- Alloca una entry nella Process Table per il nuovo processo
- Assegna un PID unico al nuovo processo e inizializza i campi della Process Table entry
- Crea una copia della immagine del processo padre (il testo non viene duplicato ma si incrementa un reference count)
- Incrementa opportuni contatori dei file aperti
- Inizializza i contatori di accounting nella u-area del nuovo processo
- Pone il nuovo processo nello stato di pronto
- Restituisce il PID del figlio al padre, 0 al figlio o -1 in caso di errore

24

Exec



Implementazione di processi (4)

- *Cambio di contesto (context switch)* : è ciò che accade quando un processo passa in esecuzione
  - il processore deve caricare i propri registri interni con le informazioni relative al nuovo processo da mandare in esecuzione
  - l'hw che realizza rilocazione e protezione deve essere aggiornato
    - es : registri base e limite, i sistemi attuali usano meccanismi più sofisticati

26

Implementazione di processi (5)

- Il cambio di contesto è una operazione molto costosa
  - Il costo in sistemi reali è dell'ordine del millisecondo!
  - Quindi lo scambio non può essere effettuato troppo spesso (20-200ms in sistemi reali Unix Linux)

Devo trovare un trade-off: tempo che lavoro molto grande rispetto a tempo schedulatore, ma devo comunque gestire all'esterno interattività.

27