SISTEMA OPERATIVO - GENERALITÀ

Il Sistema Operativo (S.O.) **controlla** e **coordina** l'uso dell'hardware fra i differenti programmi applicativi dei vari utenti

Il SISTEMA OPERATIVO si occupa di GESTIRE le **RISORSE HARDWARE e SOFTWARE**

Esempi di risorse Hw:
- processori (tempo di CPU)
- memorie
- dispositivi periferici (I/O)
- etc.

Il S.O. deve:
a) tenere traccia delle risorse
b) adottare strategie di assegnazione delle risorse

A CHI assegnare la risorsa
QUANDO assegnarla
PER QUANTO TEMPO

c) allocare le risorse
d) recuperare le risorse

Quindi, le **risorse** devono essere assegnate a programmi (specifici) secondo determinate politiche

RISOLUZIONE POSSIBILI CONFLITTI ➔ il S.O. deve garantire l'**integrità del sistema** e ottimizzarne le **prestazioni globali**

GENERALITÀ (segue)

Un S.O. è un **PROGRAMMA DI CONTROLLO**
➔ controlla l'esecuzione dei programmi per prevenire errori ed usi impropri del calcolatore

Le funzioni specifiche di un sistema operativo sono:

1) controllo del/dei processore/i
2) controllo della memoria principale
3) controllo dei dispositivi di I/O

OBIETTIVI DI UN S.O.:

Rendere
- più semplice l'uso di un sistema di calcolo
- efficiente l'uso delle risorse di un sistema di calcolo

REQUISITI FONDAMENTALI DI UN SISTEMA OPERATIVO:
- AFFIDABILITÀ HW/SW
- EFFICIENZA
- PROTEZIONE DEI SINGOLI UTENTI
- ECONOMICITÀ

SISTEMI OPERATIVI MULTIPROGRAMMATI

Una delle CATEGORIE DI SISTEMI OPERATIVI più importanti è quella dei **S.O. MULTIPROGRAMMATI, come sono UNIX e LINUX**

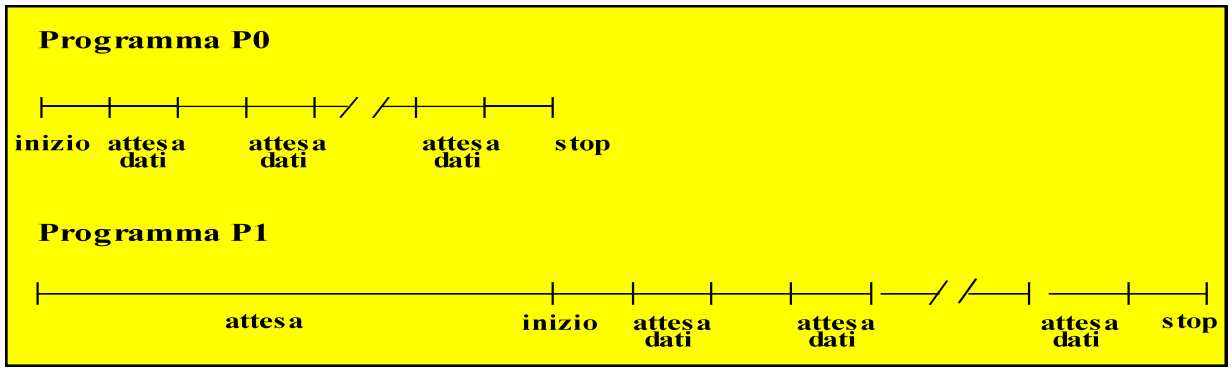
MULTIPROGRAMMAZIONE ➔ più programmi in memoria centrale

Un S.O. multiprogrammato **gestisce** "contemporaneamente" più programmi presenti nella memoria principale

Senza multiprogrammazione:

I due programmi devono essere eseguiti in maniera ***sequenziale***, cioè uno dopo l'altro!

IPOTESI SEMPLIFICATIVA:
una sola CPU



Supponiamo di dovere mandare in esecuzione due programmi *Prg0* e *Prg1*, ognuno con le seguenti caratteristiche:
- durata complessiva di 1 minuto
- 50% di attività di CPU e il 50% di attesa di dati

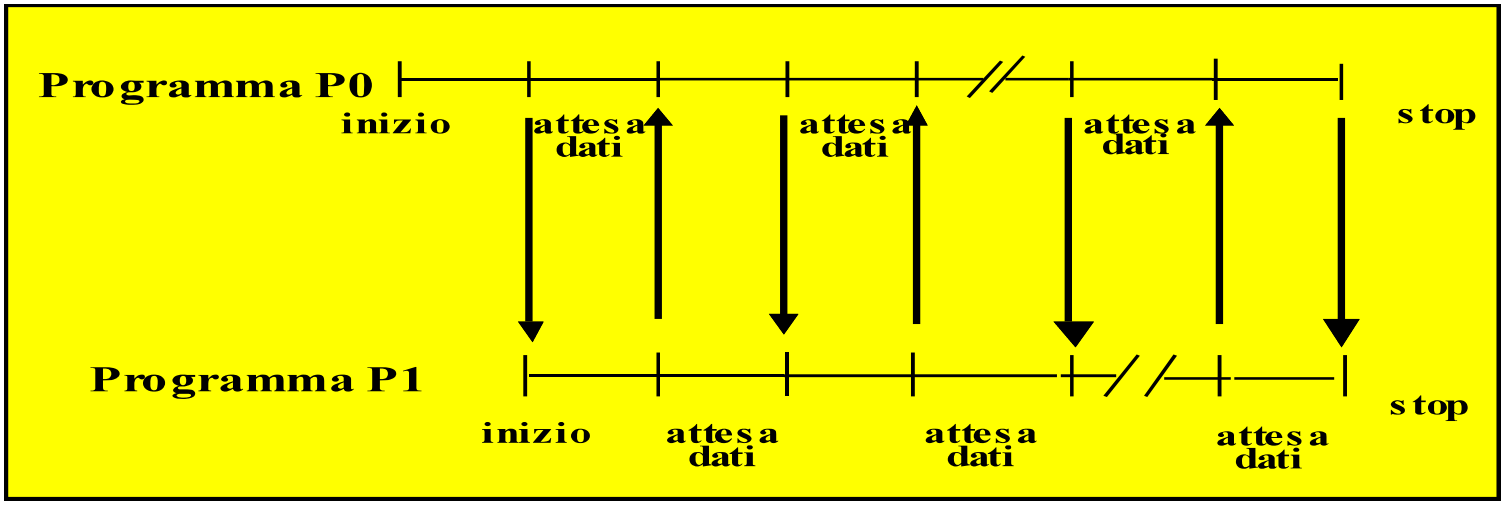
Nel caso di ***assenza di multiprogrammazione***, abbiamo:
• Durata complessiva dei due programmi: 2 minuti
• Occupazione della CPU: 1 minuto (grado di utilizzo = 50%)
• Throughput della CPU (cioè numero di programmi completati nell'unità di tempo): 1 al minuto

S.O. MULTIPROGRAMMATI (segue)

In caso di ***assenza di multiprogrammazione,*** il S.O. si dice **SISTEMA OPERATIVO MONOPROGRAMMATO:** gestisce i programmi in modo sequenziale nel tempo ➔ Tutte le risorse hardware e software del sistema sono dedicate ad un solo programma per volta ➔ **BASSO GRADO DI UTILIZZO DELLE RISORSE**

Con multiprogrammazione:

I due programmi possono essere eseguiti in maniera ***concorrente***, cioè le fasi di esecuzione di uno si possono svolgere grazie alle fasi di attesa dell'altro!



Nel caso di ***presenza di multiprogrammazione***, abbiamo:
- Durata complessiva dei due programmi: 1 minuto
- Occupazione della CPU: 1 minuto (grado di utilizzo = 100%)
- Throughput della CPU (cioè numero di programmi completati nell'unità di tempo): 2 al minuto ➔ caso teorico

Nella realtà:
in genere, grado di utilizzo e throughput più bassi dato che bisogna considerare anche i tempi "sprecati" (*overhead*) che il SO impiega per avvicendare l'esecuzione dei programmi sulla CPU

S.O. MULTIPROGRAMMATI (segue)

VANTAGGIO : Migliore **utilizzazione delle risorse** (riduzione dei tempi morti)

SVANTAGGI MULTIPROGRAMMAZIONE: **Maggiore complessità** del S.O.:

- algoritmi per la gestione delle risorse (CPU, memoria, I/O)
- protezione degli ambienti dei diversi programmi

DEFINIZIONE IMPORTANTE:

Le attività che vengono svolte da un S.O. quando un *programma* è *in esecuzione* determinano un flusso di esecuzione che viene denominato **PROCESSO**

In generale, si possono distinguere:
PROCESSI I/O BOUND ===> attività di I/O rilevante
PROCESSI CPU BOUND ===> attività di CPU rilevante

ULTERIORE TERMINOLOGIA:

- **MULTIPROCESSO** ➔ più processi attivi
- **MULTIUTENTE** ➔ più utenti possono accedere **contemporaneamente**

Possibili combinazioni:

- sistema operativo multiprogrammato e multiutente ➔ **SISTEMI OPERATIVI INTERATTIVI**
  esempio: sistemi a divisione di tempo ==> **UNIX e LINUX**
- sistema operativo multiprogrammato, ma non multiutente; esempio: sistema real-time

S.O. MULTIPROGRAMMATI (segue)

SISTEMA A DIVISIONE DI TEMPO (TIME-SHARING)

Un SISTEMA OPERATIVO ***time-sharing*** fornisce a ciascun utente una porzione di un sistema di calcolo

Ogni utente (multiutenza) ha i propri programmi in memoria (multiprogrammazione)
➔ **Ogni utente ha l'illusione di avere la macchina dedicata completamente a lui**
➔ Astrazione di più **macchine virtuali**

PROBLEMA: SCHEDULING della CPU ➔ **ALGORITMO A SUDDIVISIONE DI TEMPO**

ESEMPIO PRATICO: Supponiamo di avere due utenti (*u1* e *u2*), ognuno dei quali ha scritto un programma in C e ha ottenuto un programma nella sua forma eseguibile (*E1* e *E2*); entrambi gli utenti vogliono mandare in esecuzione gli eseguibili

SENZA MULTIPROGRAMMAZIONE ➔ Il primo utente che riesce a far accettare dal SO di caricare il proprio eseguibile sul sistema, ne ottiene l'esecuzione: p.e., supponiamo che sia l'utente *u2* che riesce a far eseguire *E2*; solo al termine della esecuzione di *E2*, l'utente *u1* riesce a far eseguire *E1*

CON LA MULTIPROGRAMMAZIONE ➔ Entrambi gli utenti potrebbero riuscire a far accettare dal SO di caricare il proprio eseguibile sul sistema e ottenerne l'esecuzione: quindi, "***contemporaneamente***", sia l'utente *u2* riesce a far eseguire *E2* e sia l'utente u1 riesce a far eseguire *E1* ➔ l'esecuzione di *E1* produce un processo (*P1*) nel SO, così come l'esecuzione di *E2* produce un processo (*P2)* ➔ *P1* e *P2* sono due processi attivi nel SO ➔ **P1 e P2 sono due processi indipendenti fra di loro, ma in esecuzione concorrente (cioè contemporanea) nel SO**

STRUTTURA DI UN SISTEMA OPERATIVO

Suddiviso in componenti (gestori) che svolgono specifiche funzioni:

* **PROCESSOR/PROCESS MANAGEMENT** (NUCLEO o KERNEL)
  gestione processore tramite processi, sincronizzazione e comunicazione per processi interagenti
* **I/O SYSTEM**
  gestione dei dispositivi di I/O
* **MEMORY MANAGEMENT**
  gestione della memoria centrale (RAM, ma non solo!)
* **FILE SYSTEM**
  gestione della memoria secondaria
* **PROTECTION SYSTEM**
  gestione degli accessi
* **NETWORKING**
  gestione di sistemi distribuiti
* **COMMAND INTERPRETER SYSTEM**
  gestione interfaccia utente ➔ interprete del linguaggio comandi

---

POSSIBILI PUNTI DI VISTA NEI CONFRONTI DI UN SISTEMA OPERATIVO

PUNTO DI VISTA DELL'UTENTE/PROGRAMMATORE ➔ esterno

PUNTO DI VISTA DEL S.O ➔ interno

GESTIONE DEI PROCESSI

PROCESSO ➔ programma in esecuzione

Il processo è l'unità di lavoro in un Sistema Operativo multiprogrammato

In generale,

un Sistema Operativo gestisce una **collezione di processi:**

* *processi di sistema* cioè processi del S.O. che eseguono codice del sistema

* *processi utenti* che eseguono codice di utente

Questi due tipi di processi possono essere in esecuzione ***concorrentemente***

GESTIONE DEL PROCESSORE

Il S.O. si deve occupare della politica di assegnazione del processore (o dei processori) ai vari processi

➔ SCHEDULING dei processi

FUNZIONI SVOLTE DAL LIVELLO DI GESTIONE DEI PROCESSI/PROCESSORI

* Creazione e distruzione di processi

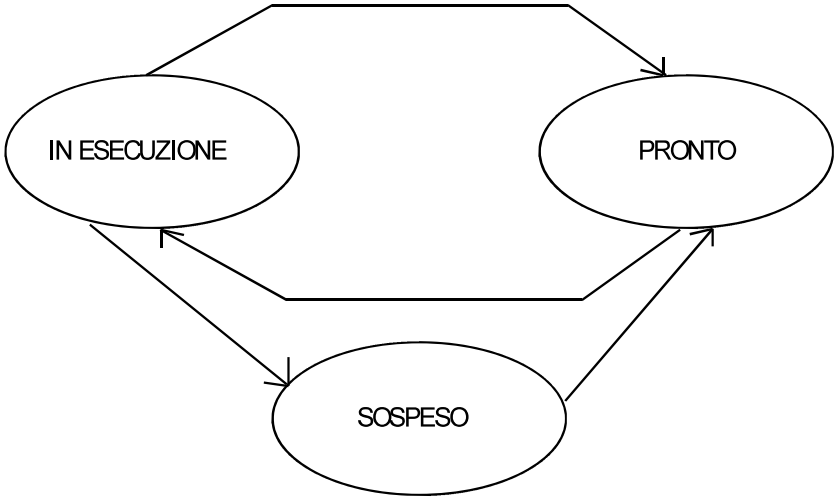
* Sospensione e riattivazione di processi

* Strumenti per la sincronizzazione e la comunicazione di processi

* Scheduling dei processi ➔ vari algoritmi a seconda delle esigenze del S.O. e/o degli utenti

Stati di un processo

- **PRONTO *(READY)*** pronto per la esecuzione
- **IN ESECUZIONE *(RUNNING)*** in esecuzione
- **SOSPESO *(SLEEPING)*** in attesa di un evento



DESCRITTORE DI PROCESSO

La possibilità che la CPU venga commutata da un processo ad un altro RENDE INDISPENSABILE, ad ogni commutazione, SALVARE TUTTE LE INFORMAZIONI del processo (ad esempio, il valore del Program Counter)

➔ OGNI PROCESSO possiede un **DESCRITTORE DI PROCESSO** che mantiene tutte queste informazioni quando il processo non è in esecuzione (PRONTO o SOSPESO)

Il salvataggio delle informazioni di un processo (quando transita dallo stato in esecuzione allo stato sospeso o allo stato pronto) e il ripristino delle informazioni di un altro processo (quando transita dallo stato pronto allo stato in esecuzione) viene detto cambio di contesto/processo (*context switching*)

CODE DI PROCESSI: ***pronti*** / ***sospesi*** } Le code sono code di descrittori di processi!

ASSEGNAZIONE del PROCESSORE (CPU SCHEDULING)

Scheduler: quella parte del S.O. che decide a quale dei **processi pronti** presenti nel sistema assegnare il controllo della CPU

➔ ***Ipotesi semplificativa (come già detto): UNA SOLA CPU***!

Quindi un solo processo si può trovare nello stato IN ESECUZIONE (running), mentre gli altri processi saranno nello stato PRONTO (ready) o nello stato SOSPESO (sleeping)

➔ **Algoritmo di scheduling:**

realizza un particolare **criterio di scelta** tra i processi pronti

➔ lavora sulla **CODA DEI PROCESSI PRONTI**

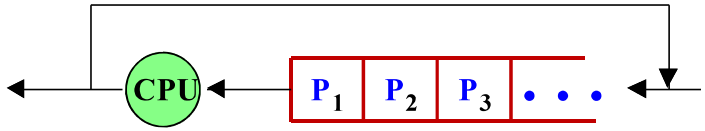
Possibilità degli algoritmi di scheduling

- **scheduling preemptive** ➔ un processo in esecuzione perde il controllo della CPU anche se a livello logico potrebbe proseguire ➔ nel diagramma degli stati di un processo esiste la transizione da running a ready
- **scheduling nonpreemptive** ➔ un processo in esecuzione prosegue fino al rilascio spontaneo della CPU
  - *terminazione*
  - *sospensione per attesa di un evento (per esempio I/O)*

  ➔ nel diagramma degli stati di un processo NON esiste la transizione da running a ready

Algoritmi di (short-term) scheduling

Scheduling Round-Robin (RR) ➔ **ALGORITMO A SUDDIVISIONE DI TEMPO**
La coda dei processi pronti è gestita FIFO, ma ad ogni processo è assegnata la CPU per un quanto di tempo (**QT**) prefissato
➔ **utilizzo di PREEMPTION**



Se il processo P1 non si sospende o termina durante QT, P1 subisce preemption e la CPU viene assegnata a P2 ➔ P1 è inserito in fondo alla coda dei processi pronti

- ➔ Un processo è in esecuzione ***al massimo*** per **un quanto di tempo** (TIME SLICE), definito dal sistema operativo
- ➔ Al completamento del quanto, il controllo passa ad un altro processo, a rotazione
- ➔ Se prima della scadenza del suo quanto, un processo fa una azione sospensiva (ad esempio richiesta di I/O), perde il controllo; a maggior ragione se il processo termina!

Scelta di QT: deve essere sufficientemente grande rispetto al tempo di cambio del contesto (dell'ordine delle decine di msec)

Overhead: Tempo impiegato dal S.O. per trasferire il controllo da un processo ad un altro

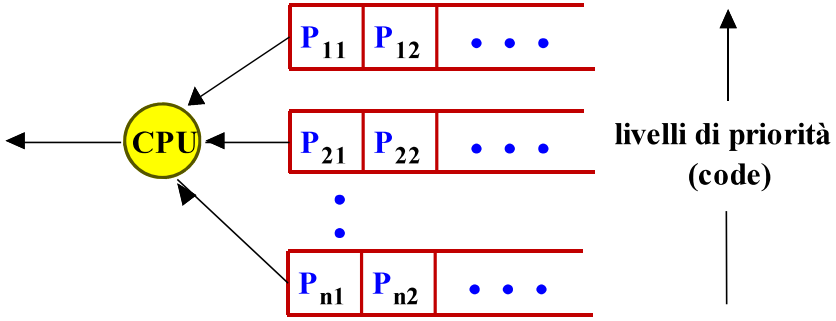
(segue Algoritmi di scheduling)

Riprendiamo l'ESEMPIO PRATICO del lucido 6:
l'esecuzione concorrente dei due processi indipendenti *P1* e *P2* può avvenire, ad esempio, grazie allo scheduling round-robin: ad ognuno dei due processi viene assegnato a rotazione un quanto di tempo fino a che i processi non terminano; chiaramente se mentre è in esecuzione uno dei due processi *P1* o *P2*, questo processo esegue un'azione sospensiva, come ad esempio una operazione di I/O, il processo viene sospeso (transita nello stato SOSPESO) e lo scheduler assegna la CPU al primo processo nella coda dei processi pronti (*P1* o *P2*)

Scheduling a Priorità

In ogni istante, è in esecuzione il processo pronto a priorità massima

Priorità **statica**: fissata alla creazione dei processi in base alle loro caratteristiche



Problema ➔ STARVATION: ad un processo a bassa priorità può non venire mai assegnata la CPU, perché vi sono sempre processi a priorità più elevata presenti nelle code superiori

Scheduling a Priorità dinamica

La priorità può essere modificata durante l'esecuzione dei processi:

- per penalizzare processi che impegnano troppo la CPU
- per evitare fenomeni di starvation

PROCESSI INTERAGENTI

Nell'esempio pratico che abbiamo fatto prima, i due processi *P1* e *P2* erano indipendenti uno dall'altro, ma i casi più interessanti sono quando si generano processi che non sono indipendenti

Infatti, nell'ambito della ***programmazione concorrente***, per risolvere un problema è necessario definire diversi processi (che eseguiranno in maniera **concorrente**) e che dovranno interagire uno con l'altro

Le interazioni possibili fra i processi sono due:

1) **Interazione indiretta o competizione** ➔ i processi hanno necessità di usare lo stesso insieme di risorse
   Ad esempio:
   a) due processi *P1* e *P2* devono eseguire una stampa utilizzando la stessa risorsa/stampante, chiaramente senza che le stampe si mescolino nella stampante; *P1/P2*, prima di poter stampare, deve poter acquisire in modo ***mutuamente esclusivo*** l'uso della risorsa/stampante e al termine della stampa, deve rilasciare la stampante;
   b) due processi P1 e P2 che ciclicamente eseguono un compito assegnato (p.e. lo stesso) devono tenere traccia in un contatore (risorsa condivisa) delle volte che hanno eseguito il compito; al termine della esecuzione di P1 e P2, il contatore deve avere come valore la somma delle esecuzioni del compito svolto dai due processi
2) **Interazione diretta o cooperazione** ➔ i processi hanno necessità di scambiarsi messaggi o notificarsi eventi
   Ad esempio: un processo *P1* esegue un certo compito e deve inviare il risultato del proprio compito ad un altro processo *P2*, che deve ricevere questa informazione prima di poter eseguire il proprio compito (eventualmente utilizzando il risultato inviato da *P1*)

Problema del DEADLOCK

Il deadlock è un problema che si può manifestare quando **processi interagenti** si trovano ad essere bloccati in attesa del verificarsi di condizioni che non possono verificarsi

➔ Il problema del deadlock si può manifestare sia nel caso di interazione di tipo competitivo che cooperativo

ESEMPIO DI DEADLOCK IN CASO DI INTERAZIONE COMPETITIVA (INDIRETTA)

Supponiamo di avere due risorse *R1* e *R2* e due processi *P1* e *P2* che hanno bisogno di usare in modo mutuamente esclusivo *R1* e *R2*: *P1* deve usare prima *R1* e poi *R2* e, viceversa, *P2* deve usare prima *R2* e poi *R1. Se supponiamo ora la seguente sequenza di esecuzione:*

- *P1* acquisisce *R1 (che è libera)* ➔ *R1 viene indicata come occupata*
- *P2* acquisisce *R2 (che è libera)* ➔ *R2 viene indicata come occupata*
- *P1* tenta di acquisire *R2*, ma non riesce dato che è stata assegnata dal SO a *P2* ===> ***P1*** **si BLOCCA** (passa nello stato SOSPESO) in attesa del verificarsi della condizione di rilascio di *R2* da parte di *P2*
- *P2* tenta di acquisire *R1*, ma non riesce dato che è stata assegnata dal SO a *P1* ===> ***P2*** **si BLOCCA** (passa nello stato SOSPESO) in attesa del verificarsi della condizione di rilascio di *R1* da parte di *P1*

➔ **P1 e P2 sono entrambi sospesi e si trovano in deadlock (o blocco critico)**

SOLUZIONI BANALI per prevenire del deadlock:

- *P1* e *P2* devono entrambi usare prima *R1* e poi *R2*
- oppure
- *P1* e *P2* devono entrambi acquisire contemporaneamente *R1* e *R2* e non separatamente

(segue Problema del DEADLOCK)

ESEMPIO DI DEADLOCK IN CASO DI INTERAZIONE COOPERATIVA (DIRETTA)

Supponiamo di avere due processi *P1* e *P2*, *P1* spedisce (in modo sincrono) l'informazione *I1* a *P2* e deve ricevere, a sua volta, l'informazione *I2* da *P2*; *P2* spedisce (in modo sincrono) l'informazione *I2* a *P1* e deve ricevere, a sua volta, l'informazione *I1* da *P1*

P1 spedisce *I1* a *P2*
P1 riceve *da P2*

P2 spedisce *I2* a *P1*
P2 riceve da *P1*

Il problema è che *P1* risulta bloccato (nello stato SOSPESO) perché non può consegnare l'informazione *I1* a *P2*, dato che *P2* risulta bloccato (nello stato SOSPESO) perché non può consegnare l'informazione *I2* a *P1*

➔ **P1 e P2 sono entrambi sospesi si trovano in deadlock (o blocco critico)**

SOLUZIONI BANALI per prevenire del deadlock:

- *P1* e *P2* devono usare uno schema di comunicazione invertito, p.e. *P1*, come prima, spedisce l'informazione *I1* a *P2* e deve ricevere, a sua volta, l'informazione *I2* da *P2,* invece *P2* deve ricevere l'informazione *I1* da *P1* e solo dopo spedisce l'informazione *I2* a *P1*

  P1 spedisce *I1* a *P2*
  P1 riceve *da P2*

  P2 riceve da *P1*
  P2 spedisce *I2* a *P1*
- *P1* e *P2* possono usare lo schema di comunicazione originale, ma devono usare (se disponibile) una spedizione asincrona in modo da non essere bloccati se il partner della comunicazione non ha ancora effettuato la ricezione del messaggio inviato

GESTIONE DEI DISPOSITIVI DI I/O

Nasconde al programmatore i dettagli e le peculiarità dei dispositivi di I/O (periferiche, memoria di massa, etc.)

In particolare: Gestione a controllo di programma ===> polling
Gestione ad interrupt ===> routine di risposta alle interruzioni

FUNZIONI SVOLTE DAL LIVELLO DI GESTIONE DEI DISPOSITIVI DI I/O
* Routine di risposta alle interruzioni
* Gestione dei buffer di I/O

GESTIONE DELLA MEMORIA CENTRALE

Esistono diversi schemi di gestione della memoria, anche in dipendenza dell'hardware disponibile

In generale: gestione degli spazi di indirizzamento dei processi

In particolare: **Memoria virtuale**
===> il programmatore ha l'**illusione** di avere a disposizione una memoria più grande di quella fisica

FUNZIONI SVOLTE DAL LIVELLO DI GESTIONE DELLA MEMORIA
* Allocazione e deallocazione di memoria
* Gestione degli spazi di indirizzamento di ogni processo
* Gestione della memoria virtuale

GESTIONE DELLA ALLOCAZIONE

Il metodo più utilizzato per allocare i programmi eseguibili nella memoria centrale (RAM) è la **paginazione ➔ metodo di allocazione NON contiguo ➔** entità che a livello logico sono contigue non è detto che a livello fisico siano allocate in modo contiguo

- lo spazio di indirizzamento della memoria centrale viene suddiviso in **pagine fisiche** (o *frame* di dimensione DP)
- lo spazio di indirizzamento di un programma eseguibile viene suddiviso in **pagina logiche** (dim. sempre DP)

ALLOCAZIONE DI UN PROCESSO:

si calcola il numero di pagine logiche NL (sempre un intero) di un programma eseguibile di lunghezza DL, come **NL=[DL/DP] ➔** il SO cerca NL pagine fisiche libere (ovunque nella memoria RAM) e vi carica il programma eseguibile

➔ TABELLA DELLE PAGINE (TDP) per ogni processo attivo nel sistema: contiene, per ogni pagina logica, il numero di pagina fisica (#pag-fisica) di dove è stata caricata

Esempio: Sistema con 1 MB di memoria fisica e indirizzi (logici e fisici) di 20 bit
dimensione pagina = 256 byte (100H) ===> 4096 pagine
Supponiamo di avere un processo *P* il cui spazio di indirizzamento logico è di 1008 byte (3F0H) ➔ viene suddiviso in 4 pagine logiche numerate da 0 a 3
Dei 20 bit: 12 serviranno per indicare la pagina (logica o fisica) e 8 per l'offeset all'interno della pagina

TDP di *P* (#pag-fisica in esadecimale, 12 bit):

0	FFC
1	100
2	103
3	FFF

(segue GESTIONE DELLA MEMORIA CENTRALE)

A tempo di esecuzione, un processo *P* genera un indirizzo logico e deve essere calcolato il corrispondente indirizzo fisico

ESEMPIO di TRADUZIONE da indirizzo logico in indirizzo fisico:

1) l'indirizzo logico viene suddiviso, dall'hw di traduzione, in un certo numero di bit per il numero di pagina logica e i restanti bit (dipende da DP) conterranno l'offset all'interno della pagina

003 (#pag-logica, 12 bit)	20 (offset, 8 bit)

2) con il *#pag-logica*, usato come indice, si accede alla TDP del processo e si recupera il *#pag-fisica* ➔ **FFF**
3) si compone il *#pag-fisica* con l'*offset* e quindi si ottiene l'indirizzo fisico che serve per accedere al dato o alla istruzione che serve al processo *P*

FFF (#pag-fisica, 12 bit)	20 (offset, 8 bit)

PROBLEMA: *Overhead* del 100% ==> per ogni accesso in memoria, si deve prima effettuare una lettura (quindi un accesso) alla TDP

SOLUZIONE: **cache delle pagine** detta anche TLB (Translation Look-aside Buffer)
➔ memoria associativa che memorizza per un sottoinsieme dei *#pag-logica il corrispondente #pag-fisica*
➔ se l'accesso alla cache ha successo l'overhead è trascurabile!

La percentuale di volte che un numero di pagina logica/fisica si trova nella cache delle pagine si dice **tasso di successo** (*hit ratio*) [l'insuccesso di chiama TLB miss] ➔ Si usano algoritmi PROBABILISTICI per caricare nella cache i numeri di pagina che hanno la più alta probabilità di servire al processo nel prossimo futuro (si usa ad esempio l'algoritmo **LRU**)

GESTIONE DELLA MEMORIA VIRTUALE

Consente ad un SO di NON caricare nella memoria centrale (RAM) l'**intero** spazio di indirizzamento di un processo attivo

➔ vengono caricate solo delle sue parti, di solito pagine, che sono quelle necessarie per l'esecuzione in un certo intervallo di tempo

➔ le parti/pagine non necessarie saranno disponibile su una zona riservata della memoria secondaria (disco)

La memoria virtuale nel caso più generale si basa su segmentazione paginata (come in **UNIX**):

- lo spazio di indirizzamento logico è suddiviso, in particolare, in un segmento di codice e in un segmento di dati/ stack
- ogni segmento logico è paginato e quindi in memoria centrale (RAM) vengono caricate da disco solo alcune pagine di ogni segmento

VANTAGGIO: il SO può caricare (*parzialmente*) più processi (spazi di indirizzamento)

➔ se un processo *P* ha bisogno di una pagina che non è caricata in memoria centrale, il sistema di traduzione degli indirizzi se ne accorge (nella TDP c'è una colonna in più con **bit di validità** ➔ *page fault*), *P* viene sospeso e lo scheduling seleziona un altro processo da far eseguire

➔ Il SO si occuperà, in modo concorrente, di recuperare la pagina necessaria all'esecuzione di *P* e di copiarla dal disco alla memoria RAM

PROBLEMA: se non ci sono pagine libere, il SO deve effettuare una *sostituzione di pagina*

➔ una pagina fra quelle in RAM è scelta (come *vittima*), ad esempio, con un algoritmo LRU (Least Recently Used) e sfrattata per fare posto alla pagina che serve a *P*

➔ la pagina vittima se è stata scritta (*dirty bit* a 1) deve essere copiata su disco, altrimenti è semplicemente sovrascritta

GESTIONE DELLA MEMORIA SECONDARIA

L'informazione può essere memorizzata su diversi dispositivi: dischi, nastri, cassette, etc.

Ciascun dispositivo di memorizzazione ha le proprie caratteristiche e la propria organizzazione fisica

Il S.O. **astrae** dalle caratteristiche fisiche introducendo il concetto di file e di file system

FILE ➔ unità di memorizzazione logica
insieme di informazioni logicamente correlate
ad esempio:
- programmi
- (codice sorgente, codice oggetto, codice eseguibile)
- dati (numerici, alfabetici, alfanumerici)

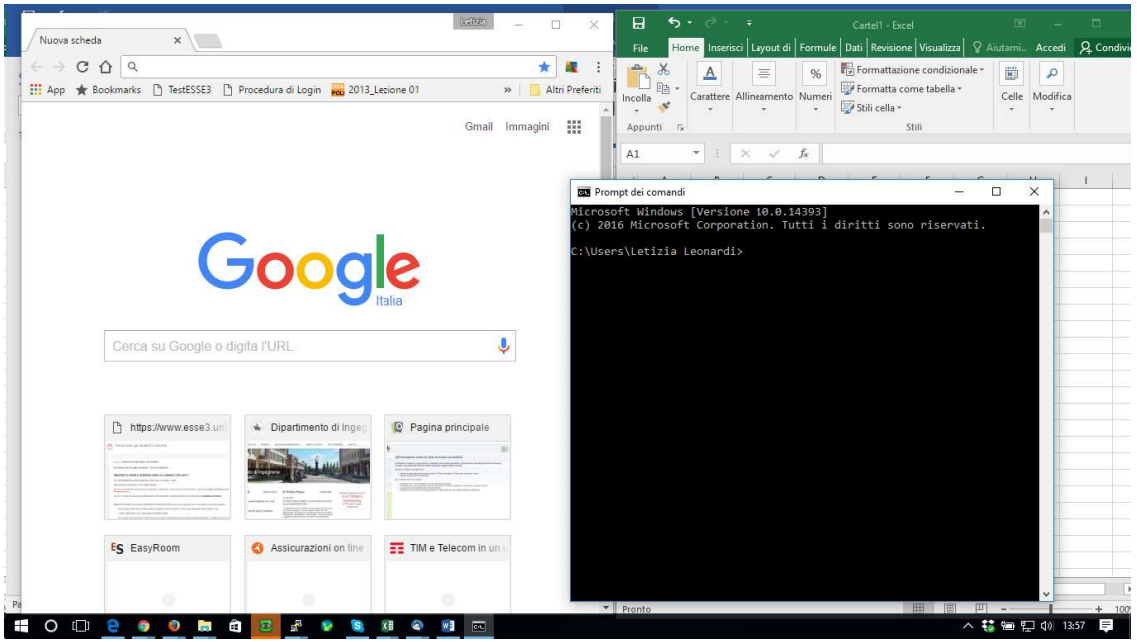
FILE SYSTEM ➔ organizzazione dei file
direttori
* ad un solo livello
* a più livelli

FUNZIONI SVOLTE DAL LIVELLO DI GESTIONE DEL FILE SYSTEM
* Creazione e cancellazione di direttori
* Creazione e cancellazione di file
* Operazioni di trattamento dei file
* Piazzamento dei file sulla memoria secondaria

GESTIONE INTERFACCIA UTENTE

Questo livello **interpreta i comandi** dati al Sistema Operativo ➔ costituisce l'interfaccia fra il S.O. e l'utente

Le interfacce utente si distinguono in:
interfacce testuali e in interfacce grafiche, ad esempio in ***Windows 10***



Anche in **UNIX** e in **LINUX** si possono avere sia interfacce testuali che grafiche ==> ***useremo in particolare una interfaccia testuale per poter sfruttare tutte le funzionalità***

I comandi sono relativi a tutte le funzioni viste:
- gestione dei processi
- gestione dell'I/O
- gestione della memoria
- gestione del file system
- gestione della protezione
- gestione dei sistemi distribuiti