# LINUX KERNEL

**SISTEMA OPERATIVO** → anche def da insieme software di funzioni che supporta l'utente
→ separa applicazioni dall'hardware che utilizzano e si occupa della gestione tra hardware e software (gestore di risorse)

→ primi S.O. si basavano su metodo **batch** = eseguivano elevati volumi di job (applicazione) sui dati di tipo ripetitivo, ovvero solo quando sono disponibili risorse di elaborazione sufficienti.
↳ lento, non interattivo, esecuzione rimandata e accorpata di programmi

→ viene introdotto il concetto di **TIME-SHARING** = suddivisione lavoro di un S.O. in quanti di tempo, assegnati ad ogni processo

CONCETTO SIMILE AI **SIST. REAL-TIME** IN CUI OGNI PROCESSO HA UNA DEADLINE

e la **MEMORIA VIRTUALE** = possibilità di indirizzare più memoria di quella di cui si dispone

→ negli anni 90' nasce l'idea di **OPEN-SOURCE** = programmi e S.O. distribuiti sottoforma di codice sorgente esaminabile e modificabile
es. linux

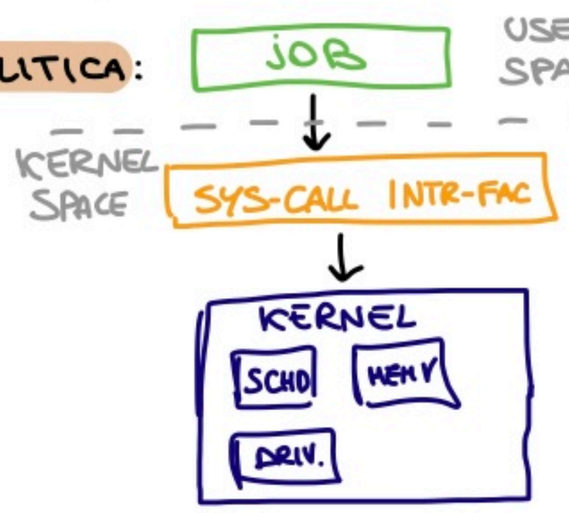
**EMBEDDED SYSTEM** → tipologia di S.O. altamente vincolato (tempo esec, mem, potenza) e formato da dispositivi molto specifici (microcontrollore progettato e codificato per un oggetto specifico).

**REAL-TIME SYSTEM** → formato da task da completare in un periodo fissato, spesso breve. Allo stesso modo deve reagire sempre in un **quanto** di tempo prestabilito.

**MACCHINE VIRTUALI** → astrazione software di un computer fisico dove S.O. gestisce le risorse, ampliando le risorse di una stessa macchina;

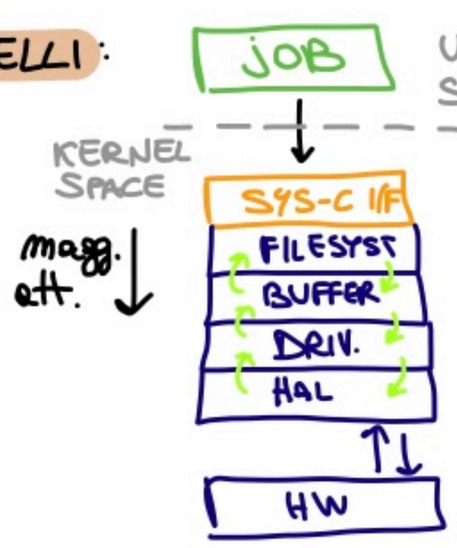
→ le principali componenti del S.O. sono:
1 → **SCHEDULER**, che si occupa di processi e thread;
2 → **MEM. VIRTUALE**, gestisce memoria tram paginazione e segmentazione;
3 → **GESTIONE I/O**, comunicazione con dispositivi e mutua esclusione;
4 → **GESTIONE ICP**, ovvero della comunicazione tra processi;
5 → **FILESYSTEM**, allocazione file e accesso ai dati;
6 → **SHELL**, interazione con utente;

**COSTRUZIONE DI UN S.O.** → **MONOLITICA**:



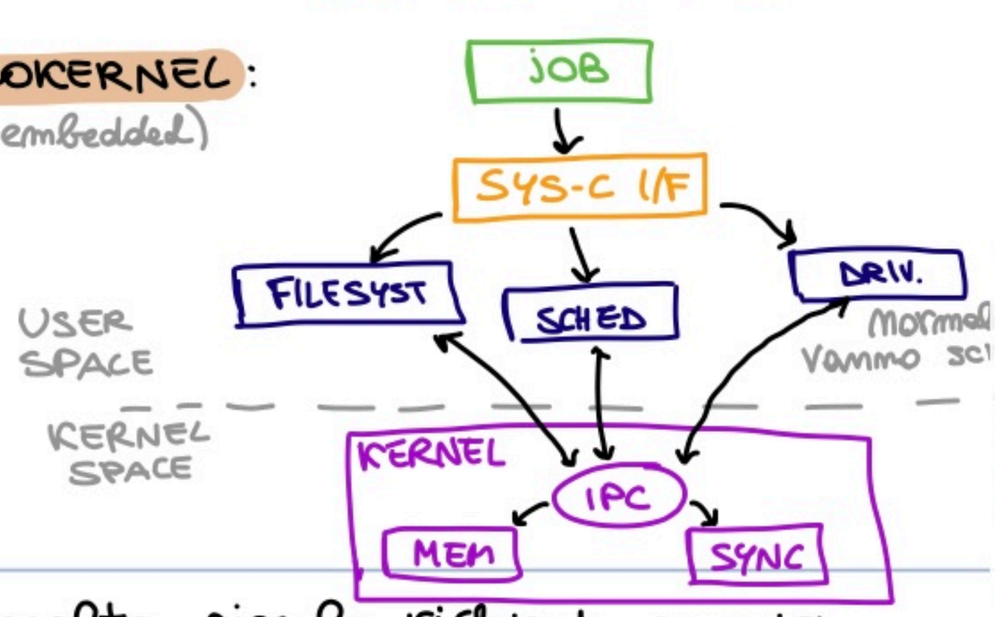

ovvero tutti gli elementi sono nel kernel, per cui è tutto interconnesso e se modifico qualcosa comunica velocemente con il resto.
Se danneggio qualcosa, si danneggia tutto
↳ NON FUNZIONALE

**A LIVELLI**:




parti del kernel divise in base alle loro funzioni
↳ PEGGIO DELLA MONOLITICA

**A MICROKERNEL**: (delle embedded)



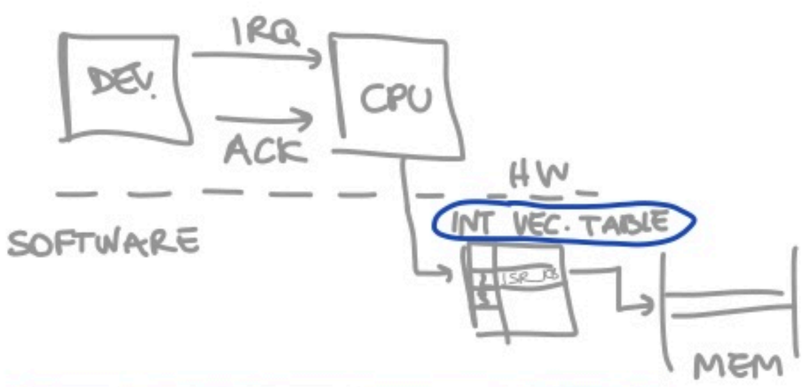

↳ kernel molto piccolo, richiede maggior comunicazione PEGG. PRESTAZ. però se una parte crasha il kernel, tutto rimane salvo

FEDE. PRESTAZ. comunicazione però se una parte crasha il Kernel centrale rimane salvo

KERNEL → nucleo dell' S.O. crea il primo processo INIT, racchiude e raggruppa le funzioni centrali dell 'S.O. virtualizzando i processi (e quindi mem, perif, ...)

↳ Si occupa quindi della gestione delle INTERRUPT, ovvero dei segnali che connettono periferiche al microprocessore;

CHIAMATA SYS-CALL → meccanismo usato dai processi per richiedere un servizio al Kernel del S.O.




→ le interrupt (IRQ) (gestite a liv. HW (da driver) e da Kernel) bloccano lo stato di fetch e fanno passare 5 cicli di clock per 'liberare' la pipelined e interrompere le istr. in esec.

- ISR → interrupt service routine, si occupa di eseguire particolari azioni di interruzioni nei casi specifici;

## OPERAZIONI SU PROCESSI

1 → creazione, alloca memoria al proc. e copia il codice eseguibile;
2 → esecuzione, lo scheduler decide quale processo eseguire;
3 → richiesta di un servizio, traduzione della sys-call per un processo;
4 → sospensione evento, porre in stato di attesa;
5 → sosp. per esaurimento tempo, uguale a pto 4;

↳ STATO = (di un processo) è la fase esecutiva a cui è impostato dal S.O.

DIAG STATI DEI MOMENTI: (senza swap)

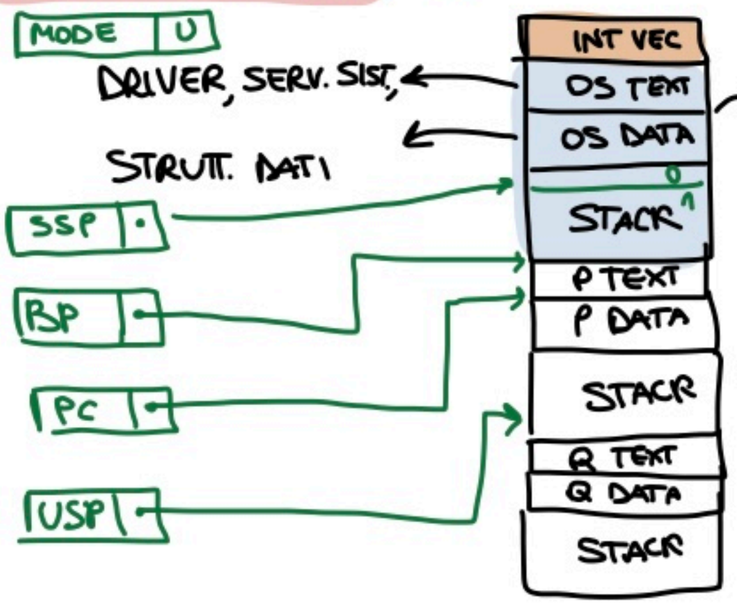

→ processi rappresentati da strutture dati: 1 → process descriptor
2 → enum possibili stati (uso typedef enum)

↳ STRUTTURE DATI

KERNEL → PD (proc. desc.) per tutti i processi (tabella) typedef struct Proc.Desc
→ una var = current process processTable[...] di tipo Proc.Desc

↳ le funzioni che modificano la tab.: (tutte void)
- sleepOn(int event), sospende causa evento;
- change(), cambia evento in running;
- wakeup(int event), ripresa di un proc;
- preempt(), sospende per fine tempo;

ORG. MEMORIA → costituita come segue




→ MEM SISTEMA, contiene tutte le spec. dell' S.O.

→ MEM UTENTE, con le varie mem per ogni processo

→ servono altre info per gestire i processi, salvate in regist
della CPU:
MODO → utente(U), superv(S)
USP, SSP, BP (base pointer), PC (punta alla prossima esec.)
User/system stack pointer

MODE (U o S) → indica con quali diritti/privilegi accedi alla macchina:

S → ha accesso a tutta la mem e può eseguire tutte le istr;

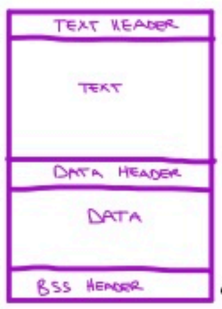
U → ha accesso solo alla parte utente e non all'area riservata a S.O., nel caso in cui avesse bisogno dell'accesso i prog. utenti possono inviare una SYS-CALL (istr. non-privilegiata) all' S.O.

'return address' del programma viene salvato sulla pila come anche il contenuto PSR (proc. status reg.) e nel PC e PSR vengono caricati valori da un vettore di syscall

→ SYS-RET() svolge operazione inversa a sys-call

→ la compilazione di un programma produce un file binario: ELF (exec and link file) in Linux
→ o da utente o da altro processo

→ l'ELF:




↳ quando viene creato un proc. (per es. tram fork()): → strutt. dati inizializzata (es. ProcTab[8].Status=NEW; ProcTab[8].StackPtr=SSP; ....) ovvero PC, BP, USP, SSP, MODE

CREAZ.

→ l'elf viene usato per allocare mem al processo e salvare le var (iniz. e non)

→ passaggi di stato all'esec: PS_NEW → PS_READY, da scheduler lungo periodo entra nella coda dello sched di breve periodo;

ESEC.

PS_READY → PS_RUNNING, gestito da scheduler di breve periodo;

→ quando un prog. viene eseguito PC passa da OS text → proc P text (header):
↳ ProcTab[8].Status=PS_READY; e poi passa a PS_RUNNING

→ quando effettuo una SYS-CALL passa a mode → S e PC punta a OS text fino a fine esecuzione richiesta e poi viene ripristinato a U e PC → Proc. P text.
Durante esec indr e modo di ritorno sono salvati su SSP

→ se devo sospendere un processo prima di ripristinare eseguo la SleepOn() che salva tutto su SSP (i registri) e il PID, passa da PS_RUNNING → PS_BLOCKED e invoca una change()

→ la funz. change():
- completa salvataggio;
- chiama scheduler per scegliere processo;
- sposta BS, SSP, USP passando al contesto del nuovo processo;

SOSP. E RIPR.

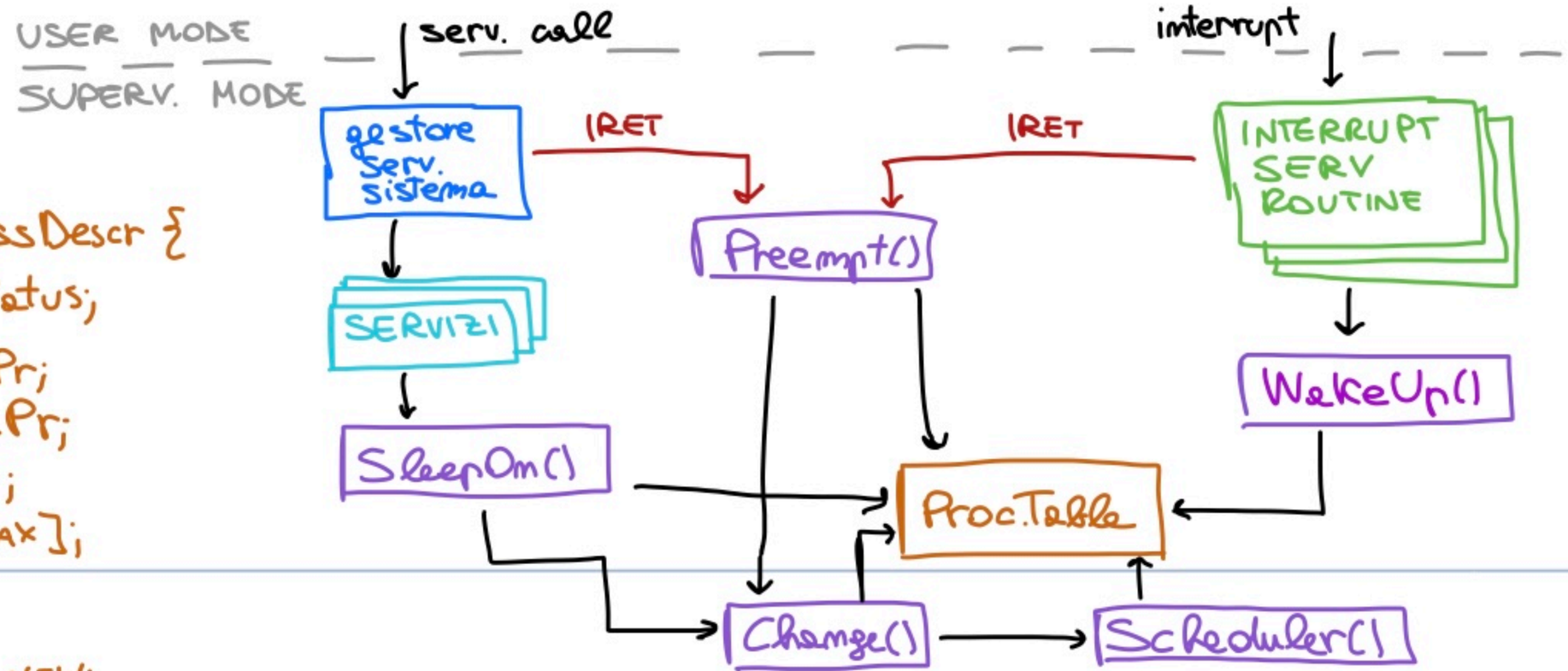
→ quando ritorno al processo bloccato recupero gli indirizzi da SSP (dove ho il vecchio proc. a differenza di USP) e riporto PC al processo bloccato;

→ questo succede se nel processo che ha sost. il vecchio si verifica l'evento (interrupt, time
eventi SO) che il vecchio processo aspettava:
1 → invoco ISR (int. serv. rout);
2 → ISR chiama WakeUp();
3 → ISR termina esec. vecchio processo;

→ periodicamente viene anche chiamata la funzione Preempt() allo scadere di un quanto di tempo

che innesca una sospensione;

STRUTTURA KERNEL →




- typedef struct ProcessDescr {
  ProcessStatus Status;
  Char * StackPr;
  char* BasePr;
  int event;
  int File[MAX];
  ....
  }
- e typedef enum { PS_NEW;
  PS_READY;
  ⋮
  } ProcessStatus;

- ProcessDescr ProcTable[CurrentP];
- int CurrentP;