SERVIZI DI SISTEMA

→ all'avvio (bootstrap) devono essere inizializzate alcune strutture dell'S.O.

→ creato proc. **init**, ovvero processo senza padre (ingenerato) che a seguito di config (tram login) crea shell.

PROC. IDLE → primo processo dopo init, viene eseguito quando non vi sono altri proc infatti è il processo a minor priorità. Viene interrotto da interrupt (da R_Int) e non da sospensione da wait

SYSTEM CALL INTERFACE → usate per richiedere servizi al Kernel da parte dei processi chiamanti

funzione system_call() → salva i registri su pila
→ controlla che il num in rax (→ numero serv. richiesto) sia valido
→ invoca **system call service routine**;

↳ quando termina ricarica i registri, invoca schedule() se nec e ritorna a prog. modo U tram SYSRET.

CREAZIONE DEI PROCESSI → uso la **fork()** per processi e **pthread_create()** per thread (→ condivide mem e tab. file aperti)

↳ entrambe create da **SYS_CLONE** (specifica tutte le condiv. tra padre e figlio (mem, codice, quantità))

→ int clone (int *(fn)(void*), void *child_stack, int flags, void *arg, ...)

- int *(fn)(void*): puntat. a funz. fn che prende come arg void* e restituisce int (eseguita da proc. figlio)
- void *child_stack: indr virt pila utente per il proc. figlio
- void *arg: param. per fn

→ pthread_create() invoca clone() → proc. padre e figlio condividono mem (norm. in std POSIX) e file

→ invece sys_clone crea processi similmente a fork():

es.
```
fork(){ ...
        syscall(sys_clone, no flags, 0);
        ... }
```
↳ $SP_{child} = SP_{father}$

→ creo clone() tram sys_clone:

```
int clone(....){
    syscall(sys_clone, ...);
    if(child)
        { fn();
          _exit();   // a diff di exit() non elim. riferim.
        }
    else return;
}
```

ELIMINAZIONE PROCESSI → esistono 2 syscall: `sys_exit()` e `sys_exit_group()`

`sys_exit()` ← cancella 1 processo ← rilascia risorse e ritorna al padre ← invoca schedule() per scegliere nuovo processo

`sys_exit_group()` ↑ cancella tutti i processi ↓ invia a tutti signal terminaz. ed esegue sys_exit()

↳ avviene a seguito di return pthread_exit_group