In Unix ogni attività è mappata in un processo e con uno spazio di indirizzamento solo locale, infatti Unix segue il modello ad indirizzamento locale (ogni processo ha il suo spazio di indirizzamento e non condivide memoria con nessun altro processo).
Ogni processo ha dunque la visione dei propri dati intesi sia come dati definiti a livello di programmazione che come dati dell'area kernel.

L'unica risorsa condivisa è il **filesystem**, perché come già visto in varie occasioni, effettivamente possiamo avere dei processi che utilizzano e condividono file sul filesystem.
Nel caso in cui i processi siano relazionati il codice non viene replicato per processo in processo. Questo processo di condivisione di codice è sottinteso dal concetto di *shell* e *sottoshell*.
L'ambiente del processo padre è condiviso fra i figli, ma non con altri processi.

Ad ogni utente viene assegnato sicuramente un processo (in generale la shell principale) e ogni utente può generare più processi, ovviamente.
I processi di Unix sono tutti **rientranti**, ovvero che possono eseguire lo stesso codice senza avere problemi (tipo nginx su più utenti, per dire).
La gestione della memoria prevede anche l'utilizzo di memoria virtuale (*swap*, su unix), basata su paginazione. Lo swapping comporta che processi non necessari siano **swappati** su memoria secondaria.

Un processo può essere eseguito in modi diversi:
- Normale (user process)
- Kernel (processo di sistema): la priorità del processo è maggiore. I processi avviati in kernel mode differiscono da quelli normali poiché fanno uso di *primitive* (open, creat…).

Il sistema gestisce i processi tramite i seguenti componenti.

1) La process table
La process table è la tabella dei processi di sistema, che mantiene un descrittore di processo per ogni entry. I record di questa tabella corrispondono a tutti i processi attivi.
Il PCB (process control block) identifica tutte le informazioni di un processo attivo.
Nel PCB si possono trovare:
- PID;
- PPID (pid del padre);
- real uid (user identifier);
- real gid (processo creato da un gruppo dell'utente di cui si tiene traccia nel descrittore);
- effective uid (user id effettivo);
- effective gid (group id effettivo);
- locazioni (puntatori) delle aree utente e di kernel;
- stato del processo (zombie, active, sleep);
- proprietà del processo;
- puntatore al prossimo processo in coda;
- valore del program counter.

2) Tavola dei codici correnti
Mantiene in memoria tutti i codici (text) in esecuzione e i relativi contatori.
Tutte le volte che un processo condivide codice con un altro, si incrementa il valore di processi che condividono quel codice. Quando il contatore arriva a zero (nessun processo condivide quel codice) si dealloca quell'area (si noti che l'area di codice è swappabile).
In caso di paginazione la text table contiene l'indirizzo della page table.

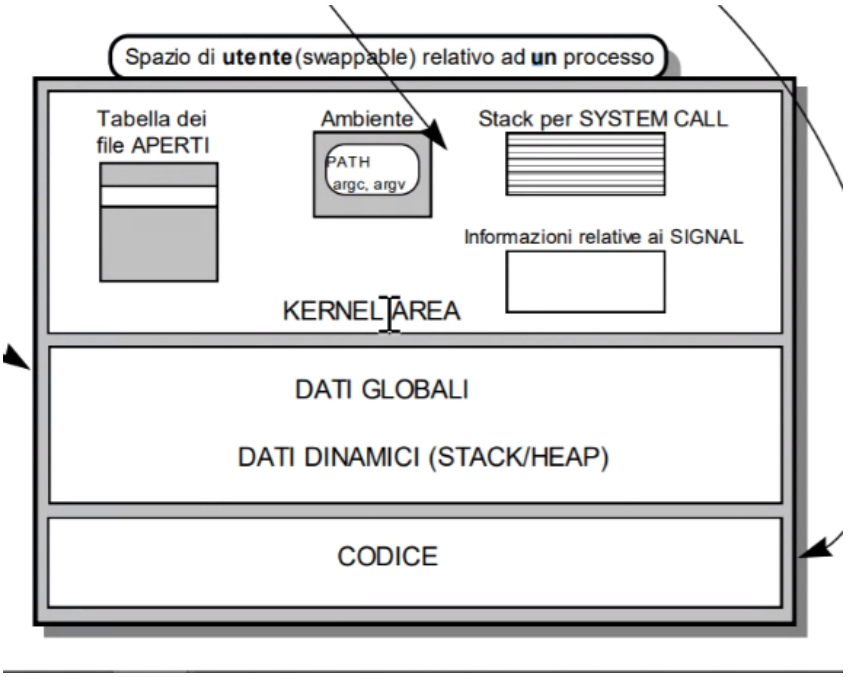
Queste due tabelle non vengono mai **swappate**.
Le tabelle che verranno esposte d'ora in poi saranno swappabili.

3) L'area utente swappabile (per processo)
L'area utente è composta da:
- Area kernel

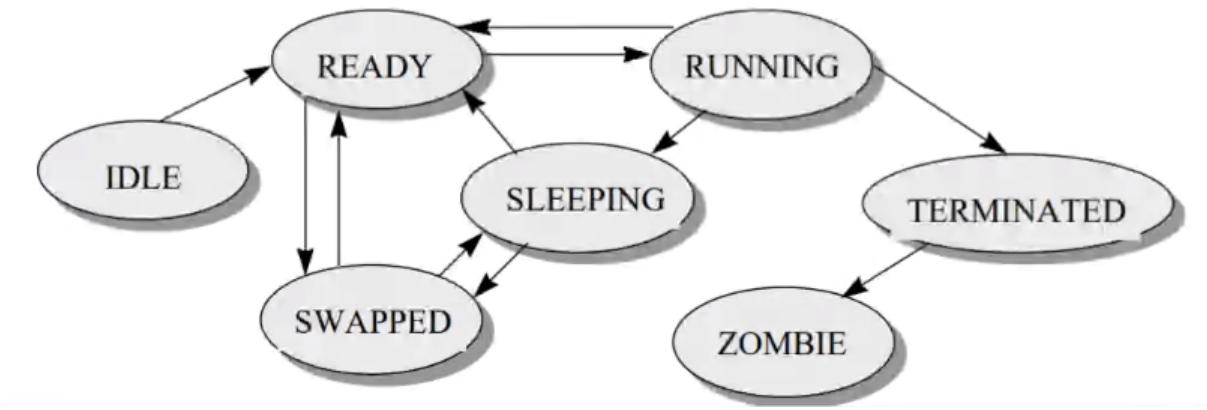
L'area di kernel viene vista come estensione del kernel a livello del singolo processo, accessibile solo durante l'esecuzione delle primitive del sistema operativo.
L'area del kernel è formata da:

a) Stack del kernel: stack usato durante l'esecuzione di primitive (un vero e proprio stack, la pila).
b) System data, composta da:
    i) Environment: argc, argv, env
    ii) User area: informazioni per l'esecuzione del processo (ad esempio: tabella file aperti etc…)

- Dati globali e dinamici
- Codice del processo



Stati di un processo



- Idle: il processo è in creazione;
- Ready: processo pronto per essere schedulato;
- Running: processo in esecuzione (correntemente schedulato);
- Sleeping: processo sospeso (input in attesa o altro)
- Swapped: processo sfrattato dalla memoria centrale (si noti che un processo running non può essere, ovviamente, swappato);
- Terminated: processo terminato, vanno deallocate le varie zone adibite al processo;
- Zombie: il processo è terminato, ma in attesa di consegna del risultato al padre (che non ha aspettato il figlio). Race condition?

Primitive per la gestione dei processi

1. *fork()*

La fork non ha bisogno di alcun parametro e restituisce il pid (al processo padre) del processo figlio (child process).
Dopo la generazione si avranno 2 processi **concorrenti e separati**.
Se si verifica un qualsiasi errore viene restituito -1.
Gli errori possono essere vari:

- Il sistema operativo ha esaurito lo spazio nella process table;
- L'allocazione di spazio swappabile non viene eseguito correttamente;
- …

Se la form ha successo, il **figlio** viene inserito nella tabella dei processi con molti elementi ereditati dal padre (stesso UID e GID e stesso valore del PC).
Il codice e l'area dati (sia kernel che user) sono le medesime, che vengono dunque copiate nella relativa riga della process table.
Viene inoltre inserito il descrittore del figlio nella coda dei processi pronti (ready).