

Gestione dei processi nel sistema operativo Unix

(Bach: the Design of the Unix Operating System (cap: 6, 7, 8))

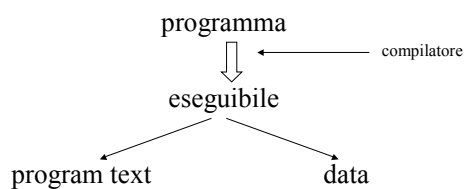
1

Argomenti

- Processi
- Strutture dati associate ai processi
- boot, init, shell
- Process Scheduling

2

Un eseguibile Unix è fatto da:

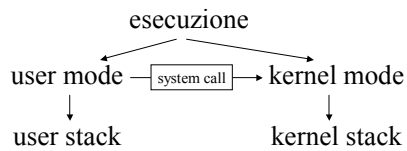


3

All'esecuzione:

- viene caricato il codice in memoria (con la execl) diviso in tre parti dette **regioni**:

regioni {
 program text
 data
 stack (allocato dal kernel)



4

Strutture dati per la gestione dei processi:

- **Process table** (globale)
- **User-area** (una per ogni processo)
- **Region table** (globale)
- **Per-process Region table** (una per ogni processo)

5

Process table:

- Array di record di dimensione fissa
- PID = numero della entry nell'array
- Sempre in memoria primaria.
- Ogni entry contiene:
 - puntatori alla u-area e alla per-process region (pregion) table
 - dimensioni del processo
 - user-id (uid), pid, ppid
 - stato del processo
 - pending signals (inviati con una kill e non ancora serviti)
 - priorità
 - parametri di accounting

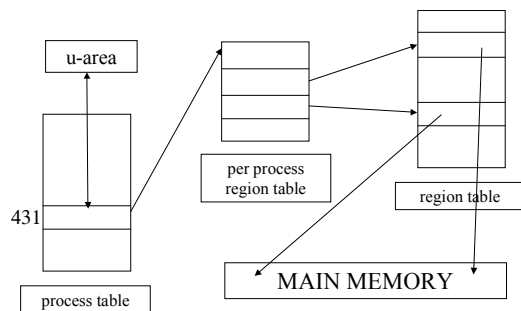
6

User-area (U-area)

- Ce n'è una per ogni processo, ed è usata solo quando il processo è attivo
- Contiene:
 - puntatore alla process table entry corrispondente
 - real user id (ruid), effective user id (euid)
 - tty (da cui è stato lanciato il processo)
 - current directory (da cui è stato lanciato il processo)
 - tabella dei file aperti per il processo
 - array di gestione delle signals (che fare per ogni signal)
 - timer fields
 - return values for system calls

7

strutture di gestione dei processi



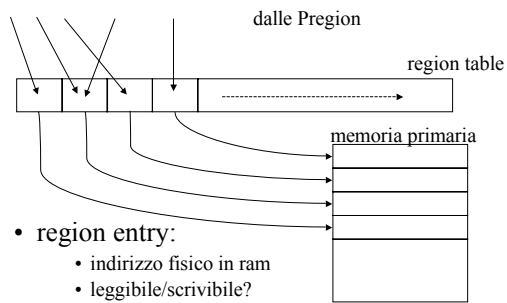
8

REGIONI:

- **REGIONE:** area contigua dello spazio di indirizzamento logico di un processo.
- **PREGION:** per process region table:
 - program text (r) →
 - data (rw) →
 - stack (rw) →
 - shared. mem (r/w) → } alla region table
- allocata nella proc. table, nella u-area, o separatamente

9

Region table:

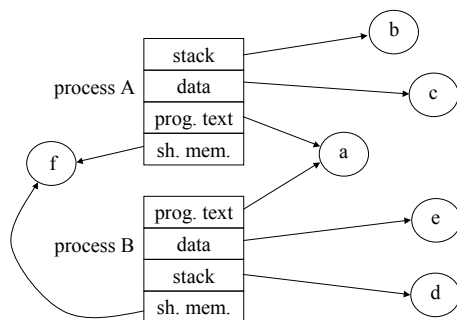


- **region entry:**

- indirizzo fisico in ram
- leggibile/scrivibile?
- condivisa?
- region counter (quanti la stanno usando)

10

Pregion tables: solo codice e shared memory possono essere condivise



11

Lo sticky bit

- Obiettivo: velocizzare lo start-up dei programmi
- comando unix: **chmod +t** program-file
- Un eseguibile con lo sticky bit settato non viene rimosso dalla memoria primaria anche se nessun processo lo sta usando
- solo il super-user può settare lo sticky bit (perché?)
- Quand'è che lo sticky bit non ha effetto?

12

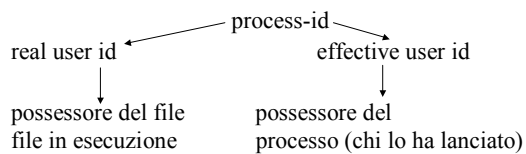
Lo sticky bit

- Lo sticky bit viene disabilitato quando:
 - all'apertura del file in scrittura
 - `chmod -t program`
 - `rm file`
 - `umount filesystem of file`
 - kernel out of space in swap device

13

set-uid option (bit S)

- **Obiettivo:** concedere *temporaneamente* i privilegi di un utente ad un altro



- normalmente, i privilegi di un processo (quali permessi possiede) sono quelli dell'utente che ha lanciato il processo

14

set-uid bit (bit S)

- In alcuni casi questo può creare dei problemi. Consideriamo un programma PIPPO appartenente all'utente A:

```
.....
open("pluto", "w");
.....
```

} pippo.c

- `-rwx-r-xr-x` A pippo
 - `-rwx-----` A pluto
- notate le protezioni del file pluto

15

set-uid bit (bit S)

- Un altro utente B esegue pippo:

pippo.c:open("pluto","w")

-rwx-r-xr-x A pippo
-rwx----- A pluto

ERRORE

- Effective UID (P) = utente B, e B non ha i permessi per scrivere nel file pluto

16

set-uid bit (bit S)

- se settiamo il bit S: **chmod +s pippo**

• -rwsr-xr-x A pippo

bit s

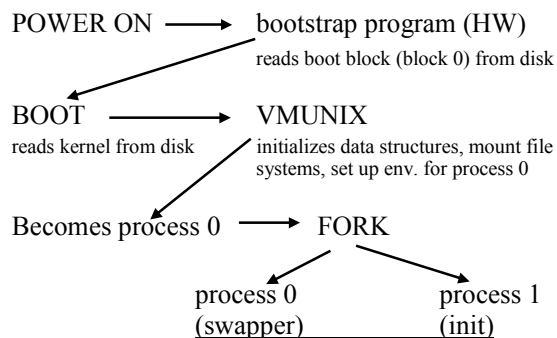
- ecco cosa succede, se un utente B esegue pippo:

– effective_uid(P) = A

- Molti comandi Unix hanno bisogno del bit S settato per funzionare correttamente: passwd, login, ... **Ma attenzione alle sh di proprietà di root con bit S settato!!!**

17

bootstrap Unix



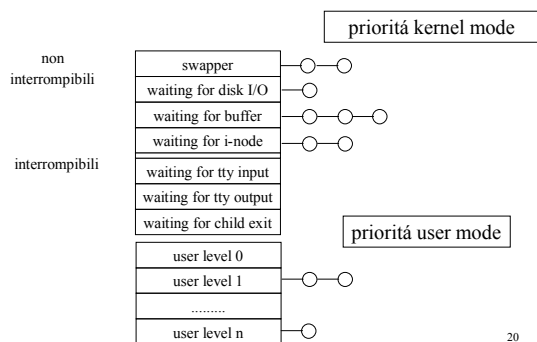
18

Scheduling della CPU

- round robin con multilevel feedback
- clock interrupt ogni 50/100 millisecondi
- azione: seleziona il processo con priorità più alta e *ready to run in memory*
- in caso di parità sceglie il processo che ha aspettato più tempo
- se non ci sono processi aspetta fino al prossimo interrupt

19

parametri di scheduling



20

parametri di scheduling

- **ogni quanto di tempo consumato:**
 - $\text{cpu_usage}(\text{proc}) := \text{cpu_usage}(\text{proc}) + \text{quanto}$
- **ogni secondo, per ogni processo:**
 - $\text{cpu_usage}(\text{proc}) := \text{cpu_usage}(\text{proc}) / 2$
- **ogni sec., per ogni proc. ready to run (user mode)**
 - $\text{priority} := \text{cpu_usage}(\text{proc}) / 2 + \text{base_priority}(\text{proc})$

il nice!

21

scheduling

- Alcuni unix permettono di allocare la CPU in base a classi di utenti o gruppi di utenti:
 - utenti ROSSI, BIANCHI: 50% tempo di CPU
 - gruppo DOCENTI: 30% tempo di CPU
 - gruppo STUDENTI: 20% tempo di CPU
- unix standard non é adatto per il real time:
l'algoritmo di scheduling non garantisce che un dato processo sia selezionato entro un limite di tempo fissato.

22
