LABORATORIO DI SISTEMI OPERATIVI

Corso di Laurea in Ingegneria Informatica

A.A. 2020/2021

Ing. Domenico Minici


domenico.minici@unifi.it

ESERCITAZIONE 5

Processi in Unix/Linux

System call per i processi

PROCESSI IN UNIX

Caratteristiche dei processi in Unix

- Unix è una famiglia di sistemi operativi multiprogrammati basati su processi
- Il processo Unix mantiene spazi di indirizzamento separati per dati e codice
  - Spazio di indirizzamento dei dati privato
    - Comunicazione fra processi basata su scambio di messaggi
  - Spazio di indirizzamento del codice condivisibile
    - Più processi possono eseguire lo stesso codice
- Unix adotta una politica di assegnamento della CPU ai processi basata sulla divisione di tempo
  - I processi attraversano vari stati

Stati di un processo

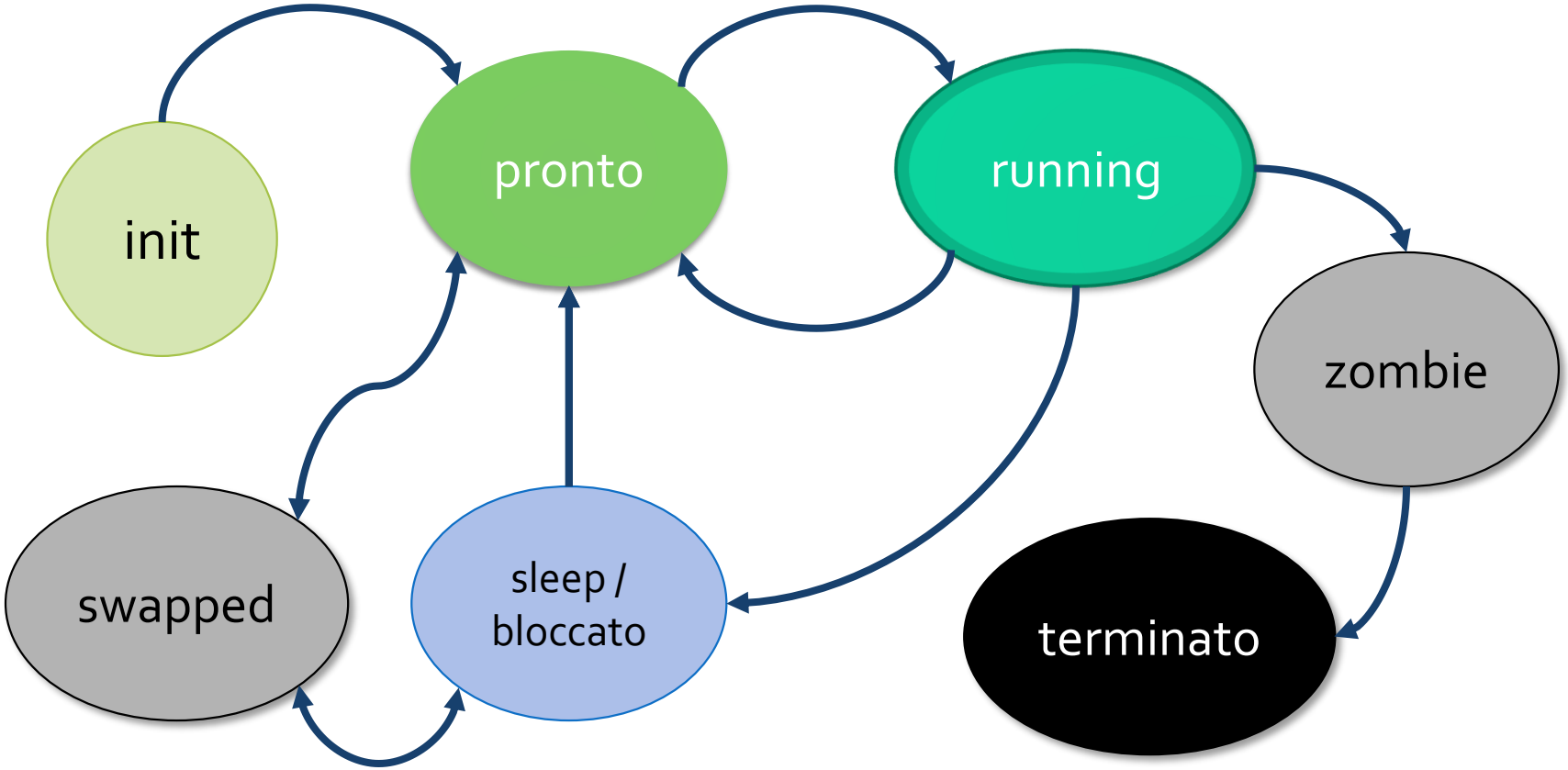


Immagine di un processo Unix

- Il descrittore di un processo (PCB – process control block) è suddiviso in due strutture dati distinte:
  - Process Structure
    - Informazioni indispensabili, sempre in memoria
  - User Structure
    - Informazioni utili solo quando il processo è residente in memoria (soggetta a swap-out)

Immagine di un processo Unix

PROCESS STRUCTURE

- PID
- Stato
- Riferimento ad aree dati e stack
- Riferimento (indiretto) al codice
- PID padre
- Priorità
- Riferimento al prossimo processo
- Puntatore alla User Structure
- ….

USER STRUCTURE

- Copia registri CPU
- Info su risorse allocate
- Info su gestione eventi asincroni (segnali)
- Directory corrente
- Utente proprietario
- Gruppo proprietario
- Argc, argv, path, …
- …

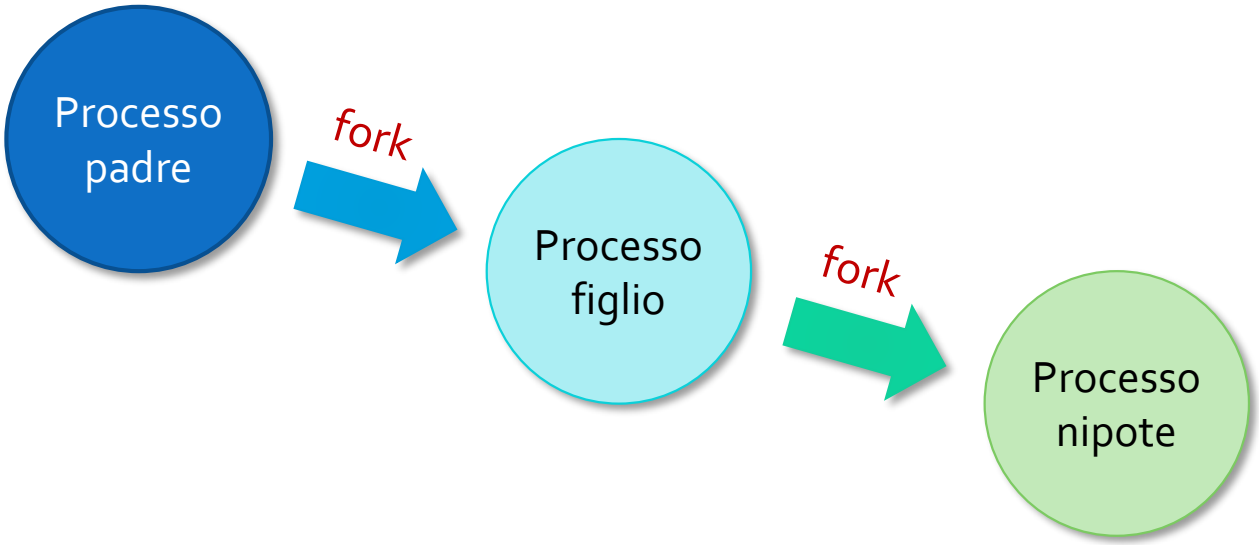
SYSTEM CALL PER I PROCESSI

System call per i processi

- Creazione di processi:
  `fork`
- Terminazione:
  `exit`
- Sospensione in attesa della terminazione dei figli:
  `wait`
- Sostituzione di codice (e dati):
  `exec..`

Creazione di processi – fork

- Ogni processo è in grado di creare dinamicamente processi
  - Lo strumento per la creazione è la chiamata di sistema **fork**
  - Il processo creato (**figlio**) ha uno spazio dati separato, ma condivide con il **processo padre** il codice
  - Ogni processo figlio può a sua volta generare altri processi



Creazione di processi – fork

`pid_t fork(void)`

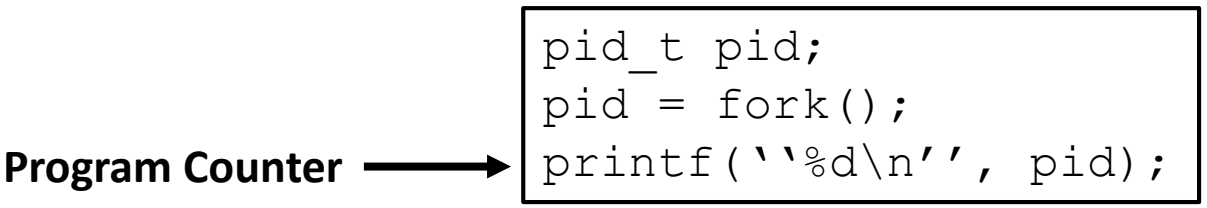
- Non richiede parametri
- Restituisce un risultato (intero) diverso a padre e figlio
  - Al padre: PID figlio, valore negativo se fallisce
  - Al figlio: zero

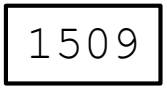
- Il processo figlio:
  - Condivide il codice con il padre
  - Eredita una copia delle aree dati globali, stack, heap, User Structure
    - Ogni variabile è inizializzata con il valore assegnatole dal padre
    - Stesso valore di Program Counter del padre

Creazione di processi – fork

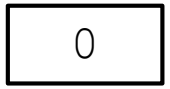
- Dopo la fork, padre e figlio partono dalla stessa istruzione: quella che segue la fork
  - Il loro comportamento può essere differenziato sfruttando il valore di ritorno della fork

ESEMPIO





OUTPUT PROCESSO PADRE



OUTPUT PROCESSO FIGLIO

PID e PPID

```
pid_t getpid()
```

- Restituisce il PID del processo

```
pid_t getppid()
```

- Restituisce il PID del processo padre

Terminazione processi

- Involontaria
  - Azioni illegali (es. accesso a locazioni esterne al proprio spazio di indirizzamento)
  - Interruzioni causate dalla ricezione di segnali

- Volontaria
  - Esecuzione dell'ultima istruzione
  - Chiamata della system call `exit()`

Terminazione processi – exit e wait

`void exit(int status)`

- I processi che terminano volontariamente possono utilizzare la system call **exit**
- È una chiamata senza ritorno (l'esecuzione termina)
- Permette di comunicare al padre lo stato di terminazione

Terminazione processi – wait

`pid_t wait(int *status)`

- Il padre può ottenere lo stato di terminazione del figlio mediante la system call **wait**
- wait ritorna il PID del figlio che è terminato
- status è l'indirizzo della variabile dove verrà salvato lo stato di terminazione del figlio

- Effetto della wait sul processo padre
  - Sospensione del padre se tutti i figli sono ancora in esecuzione
  - Ritorno immediato con informazioni di terminazione se almeno un figlio è terminato (zombie)
  - Ritorno con valore negativo (errore) se non ci sono processi figli

Terminazione processi – wait

`pid_t wait(int *status)`

- Codifica della variabile status
  - Contiene informazioni su come il figlio è terminato, oltre allo stato di terminazione eventualmente fornito dal figlio stesso
  - Se il byte meno significativo di *status è zero, allora la terminazione è stata volontaria
    - In questo caso il byte più significativo contiene lo stato di terminazione

- Macro per gestire status (in modo astratto rispetto alla reale implementazione) definite in `<sys/wait.h>`
  - `WIFEXITED(status)` – ritorna vero se terminata volontariamente
  - `WEXITSTATUS(status)` – ritorna lo stato di terminazione

Sostituzione di codice – exec..()

- Un processo può sostituire il programma (codice e dati) che sta eseguendo utilizzando una syscall della "famiglia" `exec()`:
  - `execl()`, `execle()`, `execclp()`, `execv()`, `execve()`, `execvp()`

execl

`int execl(char* path, char* arg0, …, char* argN, (char*)0)`

- Lista di parametri di lunghezza variabile
  - Terminata dal puntatore nullo
- `path` percorso del comando
- `arg0` rappresenta il nome del programma da eseguire (es. "/bin/ls")
- `arg1 ... argN` rappresentano gli argomenti del comando

- Una chiamata `exec()` è senza ritorno se ha successo
  - Solo in caso di fallimento vengono eseguite le parti di codice che seguono

ESERCIZI

Esercizio 1 – fork

- Scrivere un programma C in cui
  - Viene creato un processo figlio
  - Il processo figlio stampa un messaggio del tipo
    - "Sono X, figlio del processo Y", dove al posto di X viene stampato l'id del processo figlio (PID) e al posto di Y viene stampato il PID del padre (PPID)
  - Il padre stampa il messaggio "Sono il padre. Il PID di mio figlio è: X".
  - Cosa succede se il padre termina prima del figlio o viceversa? Fare degli esperimenti utilizzando la funzione `sleep(interval)`

Esercizio 2 – wait

- Scrivere un programma C in cui
  - Viene creato un processo figlio
  - Il processo figlio stampa un messaggio (es. "Sono il figlio Y") e poi termina fornendo come stato di terminazione il valore 1
  - Il padre attende con wait la terminazione del figlio, poi stampa a video se la terminazione è stata volontaria, ed eventualmente il valore di terminazione ottenuto
    - Provare a fare in modo che il figlio termini in modo involontario e verificare che il padre è in grado di rilevarlo
  - Modificare il programma in modo che il padre generi N figli, e poi provveda ad attendere con la wait ciascuno dei figli generati

Esercizio 3 – execl

- Scrivere un programma C in cui
  - Viene creato un processo figlio
    - Il figlio utilizza execl per sostituire il proprio codice con "ls –l argv[1]", dove argv[1] indica un argomento passato dalla linea di comando
    - In caso di errore nell'esecuzione di execl, il figlio stampa un messaggio di errore e chiama la exit con stato di terminazione 1
  - Il padre attende la terminazione del figlio ed interpreta correttamente la sua terminazione, stampando a video un messaggio informativo