04 - Processi UNIX

Processi UNIX

UNIX e' un sistema operativo multiprogrammato a divisione di tempo, l'unita' di computazione e' il processo.

Caratteristiche del processo UNIX:

- processo pesante con codice rientrante
  - dati non condivisi
  - codice condivisibile con altri processi
- funzionamento dual mode
  - processi di utente (**modo user**)
  - processi di sistema (**modo kernel**)

diverse potenzialita' e, in particolare, diversa visibilita' della memoria

Modello di processo UNIX

Ogni processo ha un proprio spazio di indirizzamento completamente locale e non condiviso --> Modello ad Ambiente Locale

Eccezioni:

- il codice puo' essere condiviso
- il file system rappresenta un ambiente condiviso

Stati di un processo UNIX

Come nel caso generale

- **Init**: caricamento in memoria del processo e inizializzazione delle strutture dati del SO
- **Ready**: processo pronto
- **Running**: processo usa la CPU
- **Sleeping**: processo e' sospeso in attesa di un evento
- **Terminated**: deallocazione del processo della memoria

In aggiunta

- **Zombie**: processo e' terminato, ma e' in attesa che il padre rilevi lo stato di terminazione
- **Swapped**: processo (o parte di esso) e' temporaneamente trasferito in memoria secondaria

Processi Swapped

Lo scheduler a medio termine (swapper) gestisce i trasferimenti dei processi

- da memoria centrale a secondaria (dispositivo di swap): **swap out**
  - si applica preferibilmente ai processi bloccati (sleeping), prendendo in considerazione tempo di attesa, di permanenza in memoria e dimensione del processo (preferibilmente i processi piu' lunghi)
- da memoria secondaria a centrale: **swap in**
  - si applica preferibilmente ai processi piu' corti

Rappresentazione dei processi UNIX

Il codice dei processi e' rientrante --> piu' processi possono condividere lo stesso codice (text)

- codice e dati sono separati (modello a codice puro)
- SO gestisce una struttura di dati globale in cui sono contenuti i puntatori ai codici utilizzati, (eventualmente condivisi) dai processi: text table
- L'elemento della text table si chiama text structure e contiene:
  - puntatore al codice (se il processo e' swapped, riferimento alla memoria secondaria)
  - numero dei processi che lo condividono...

Process Control Block (PCB): il descrittore del processo UNIX e' rappresentato da 2 strutture dati

- **Process structure**: informazioni necessarie al sistema per la gestione del processo (a prescindere dallo stato del processo)
- **User structure**: informazioni necessarie solo se il processo e' residente in memoria centrale

Process Structure

Contiene, tra le altre, le seguenti informazioni:

- **processo identifier** (**PID**): intero positivo che individua univocamente il processo
- stato del processo
- puntatori alle varie aree dati e stack associati al processo
- riferimento indiretto al codice: la processo structure contiene il riferimento all'elemento della text table associato al codice del processo
- informazioni di scheduling (es: priorita', tempo di CPU, ...)
- riferimento al processo padre (PID del padre)
- info relative alla gestione di segnali (segnali inviati ma non ancora gestiti, maschere)
- puntatori al processo successivo in code di scheduling (ad esempio, ready queue)

- puntatore alla user structure

User structure

Contiene le informazioni necessarie al SO per la gestione del processo, quando e' residente:

- copia dei registri di CPU
- informazioni sulle risorse allocate (ad es. file aperti)
- informazioni sulla gestione di segnali (puntatori a handler, ...)
- **ambiente** del processo: direttorio corrente, utente, gruppo, argc/argv, path, ...

Immagine di un processo UNIX

Immagine di un processo e' insieme aree di memoria e strutture dati associate al processo

- Non tutta l'immagine e' accessibile in modo user:
  - parte di **kernel**
  - parte di **utente**
- Ogni processo puo' essere soggetto a swapping: non tutta l'immagine puo' essere trasferita in memoria
  - parte **swappable**
  - parte residente o **non swappable**

Componenti

- **process** structure: e' l'elemento della process table associato al processo (kernel, residente)
- **text**: elemento della text table associato al codice del processo (kernel, residente)
- area **dati globali di utente**: contiene le variabili globali del programma eseguito dal processo (user, swappable)
- **stack, heap** di utente: aree dinamiche associate al programma eseguito (user, swappable)
- **stack del kernel**: stack di sistema associato al processo per le chiamate a system call (kernel, swappable)
- **user structure**: struttura dati contenente i dati necessari al kernel per la gestione del processo quando e' residente (kernel, swappable)

PCB = process structure + user structure

- **Process structure (residente)** : mantiene le informazioni necessarie per la gestione del processo, anche se questo e' swapped in memoria secondaria
- **User structure**: il suo contenuto e' necessario solo in caso del processo (stato running); se il processo e' soggetto a swapping, anche la user structure puo' essere trasferita in memoria secondaria

System call per la gestione di processi

Chiamate di sistema per

- creazione di processi: fork()
- sostituzione di codice e dati: exec...()
- terminazione: exit()
- sospensione in attesa della terminazione di figli: wait()

Creazione di processi: fork()

La funzione fork() consente a un processo di generare un processo figlio:

- padre e figlio condividono lo STESSO codice
- il figlio EREDITA una copia dei dati (di utente e di kernel) del padre

fork() non richiede parametri, restituisce un intero che:

- per il processo creato vale 0
- per il processo padre e' un valore positivo che rappresenta il PID del processo figlio
- e' un valore negativo in caso di errore (la creazione non e' andata a buon fine)

Effetti della fork()

- Allocazione di una nuova process structure nella process table associata al processo figlio e alla sua inizializzazione
- Allocazione di una nuova user structure nella quale viene copiata la user structure del padre
- Allocazione dei segmenti di dati e stack del figlio nei quali vengono copiati i dati e stack del padre
- Aggiornamento del riferimento text al codice eseguito (condiviso col padre): incremento del contatore dei processi, ...

Relazione padre-figlio in UNIX

Dopo una fork():

- **concorrenza**
  - padre e figlio procedono in parallelo
- **lo spazio degli indirizzi e' duplicato**
  - ogni variabile del figlio e' inizializzata con il valore assegnatole dal padre prima della fork()
- la **user structure** e' duplicata
  - le risorse allocare al padre (ad esempio, i file aperti) prima della generazione sono condivise coi figli

    - le informazioni per la gestione dei segnali sono le stesse per padre e figlio (associazioni segnali-handler)
    - il figlio nasce con lo stesso program counter del padre: la prima istruzione eseguita dal figlio e' quella che esegue immediatamente fork()

Terminazione di processi - exit()

Un processo puo' terminare:

- involontariamente
    - tentativi di azioni illegali
    - interruzione mediante segnale
    - --> salvataggio dell'immagine nel file core
- volontariamente
    - chiamata alla funzione exit()
    - esecuzione dell'ultima istruzione

La funzione exit() prevede un parametro (status) mediante il quale il processo che termina puo' comunicare al padre informazioni sul suo stato di terminazione (ad esempio esito dell'esecuzione). E' sempre una chiamata senza ritorno.

Effetti della exit():

- chiusura dei file aperti non condivisi
- termianzione del processo
    - se il processo che termina ha figli in esecuzione, il processo init adotta i figli dopo la terminazione del padre (nella process structure di ogni figlio al pid del processo padre viene assegnato il valore 1)
    - se il processo termina prima che il padre ne rilevi lo stato di terminazione con la system call wait(), il processo passa nello stato zombie

wait()

Lo stato di terminazione puo' essere rilevato dal processo padre, mediamente la system call wait()

int wait(int *status)

- parametro status e' l'indiruzzo della variabile in cui viene memorizzato lo stato di terminazione del figlio
- risultato del prodotto wait() e' pid del processo terminato, oppure un codice di errore (<0)

Effetti della system call wait(&status)

Il processo che invoca la wait() puo' avere figli in esecuzione:

- se tutti i figli non sono ancora terminati, il processo si sospende in attesa della terminazione del primo di essi
- se almeno un figlio e' gia' terminato ed il suo stato non e' stato ancora rilevato (cioe' e' in stato zombie), wait() ritorna immediatamente con il suo stato d terminazione (nella variabile status)
- se non esiste neanche un figlio, wait() NON e' sospensiva e ritorna un codice di errore (valore ritornato <0)

wait(): rilevazione dello stato

In caso di terminazione di un figlio, la variabile status raccoglie stato di terminazione; nell'ipotesi che lo stato sia un intero a 16 bit:

- se il byte meno significativo di status e' zero, il piu' significativo rappresenta lo stato di terminazione (terminazione volontaria, ad esempio con exit)
- in caso contrario, il byte meno significativo di status descrive il segnale che ha terminato il figlio (terminazione involontaria)

wait(): status

E' necessario conoscere la rappresentazione di status

- lo standard POSIX.1 prevede delle macro (definite nell'header file <sys/wait.h>) per l'analisi dello stato di terminazione. In particolare
  - WIFEXITED(status): restituisce vero se il processo figlio e' terminato volontariamente. In questo caso la macro WEXISTATS(status) restituisce lo stato di terminazione.
  - WIFSIGNALED(status): restituisce vero se il processo figlio e' terminato involontariamente. In questo caso la macro WTERMSIG(status) restituisce il numero del segnale che ha causato la terminazione.

System call exec()

Mediante fork() i processi padre e figlio condividono il codice e lavorano su aree dati duplicate. In UNIX e' possibile differenziare i codici dei due processi mediante una system call della famiglia exec: execl(), execle(), execlp(), execv(), execve(), execvp(), ...

Effetto principale di system call famiglia exec:

- vengono sostituiti codice ed eventuali argomenti di invocazione del processo che chiama la system call, con codice e argomenti di un programma specificato come parametro della system call

Effetti dell'exec()

Il processo dopo exec()

- mantiene la stessa process structure (salvo le informazioni relative al codice):
  - stesso pid
  - stesso pid del padre
  - ...
- ha codice, dati globaloi, stack e heap nuovi
- riferisce un nuovo text
- mantiene user area (a parte PC e informazioni legate al codice) e stack nel kernel:
  - mantiene le stesse risorse (es: file aperti)
  - mantiene lo stesso environment (a meno che non sia execle o execve)

Inizializzazione dei processi UNIX

- init genera un processo per ogni terminale (tty) collegato --> comando **getty**
- getty controlla l'accesso al sistema: exec del comando **login**
- in caso di accesso corretto, login esegue la **shell** (specificata dall'utenet in /etc/psswd)

Interazione con l'utente tramite shell

- Ogni utente puo' interagire con la shell mediante la specifica dei comandi.
- Ogni comando e' presente nel file system come file eseguibile (direttorio /bin).
- Per ogni comando, shell genera un processo figlio dedicato all'esecuzione del comando.

Relazione shell padre-figlio

Per ogni comando, shell genera un figlio; possibilita' di due diversi comportamenti:

- il padre si pone in attesa della terminazione del figlio (esecuzione in foreground); es: `ls -l pippo`
- il padre continua l'esecuzione concorrentemente con il figlio (esecuzione in background): `ls -l poippo &`

Gestione degli errori: perror()

Convenzione:

- in caso di fallimento, ogni system call ritorna un valore negativo (tipicamente -1)

- in aggiunta, UNIX prevede la variabike globale di sistema *errno*, alla quale il kernel assegna il codice di errore generato dall'ultima system call eseguita. Per interpretarne il valore e' possibile usare la funzione *perror()*:
    - perror("stringa") stampa "stringa" seguita dalla descrizione del codice di errore contenuto in errno
    - la corrispondenza tra codici e descrizioni e' contenuta in <sys/errno.h>