EXEC

giovedì 30 marzo 2023 16:09

Definizione di immagini di memoria - famiglia di chiamate exec

Ma se l'unico modo che noi abbiamo su UNIX per lanciare un nuovo processo è prendere un processo che è già attivo e in questo processo P chiamare una fork() per generare il P', COME FACCIO ALLORA SUL MIO LAPTOP UNIX A TENERE APERTO TEAMS, una shell e così via?
Noi lanciamo istanze di una stessa applicazione che è già attiva. Questo dobbiamo risolverlo.

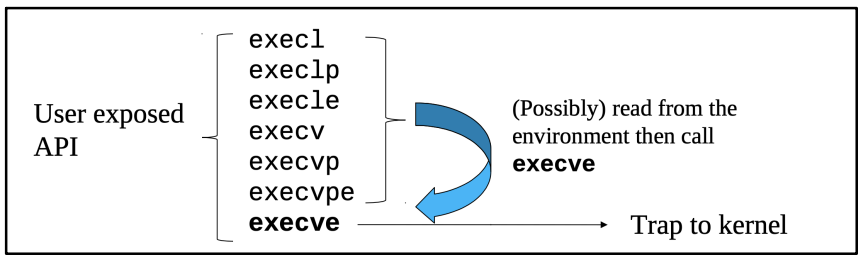
Per definire qual'è il contenuto di un address space di un'applicazione abbiamo una serie di servizi di sistemi operativi UNIX che si chiama famiglia di chiamate "exec".

L'unica system call vera e propria si chiama "execve".

Abbiamo varie funzioni che fanno parte dello standard di sistema ma una di queste è veramente una system call. Molte altre invece fanno un lavoro preliminare e poi chiamano la system_call.

L'attivazione di un programma eseguibile generico (non un clone del programma correntemente in esecuzione) avviene su sistemi Unix tramite la famiglia di chiamate exec

Attenzione: non avviene il CLONAGGIO.
Questo è quello che succede quando chiamiamo una di queste funzioni.
Abbiamo un processo attivo al quale è associato un address space, questo processo avrà un codice numerico K, T sarà il codice numerico del parent, possiamo cambiare il valore dei dati di questo processo? Cambiare il testo?
Queste API fanno esattamente questo.



Rinnoviamo il contenuto dell'address space, ma attenzione, il processo è lo stesso!
Avremo TEXT', DATA' come contenuti rinnovati.

Abbiamo sempre P in esecuzione, col suo codice K figlio di T.

La exec è una famiglia che ci permette di cambiare il contenitore di memoria del suo address space, ossia viene cambiato il vestito ad un processo.

Prima il processo P eseguiva l'applicazione X, successivamente avremo Y all'interno dell'address space dello stesso processo. Non ci sono clonaggi. Prima o poi dovremo rinunciare ad eseguire X, perché lo perderemo.

Chiaramente quando cambiamo il contenitore di memoria andiamo anche ad associare un CPU-SNAPSHOT corretto al suddetto processo, in modo tale che quando questo processo riprende il controllo della CPU ovviamente mi porta ad eseguire la prima istruzione macchina del nuovo programma contenuto all'interno dell'address space.Queste informazioni sono scritte nell'elf dell'eseguibile che sto cercando di includere nell'address space.

Queste API ovviamente le deve chiamare il processo già attivo che vuole cambiare il codice.

"La famiglia di funzioni exec sostituisce l'attuale immagine di processo con una nuova immagine di processo."

-Manuale Ufficiale

```
NAME
     execl, execle, execlp, execv, execvp, execvP - execute a file

LIBRARY
     Standard C Library (libc, -lc)

SYNOPSIS
     #include <unistd.h>

     extern char **environ;
```

```
int execl(const char *path, const char *arg, ...);

int execlp(const char *file, const char *arg, ...);
```

Puntatore a carattere

```
int execle(const char *path, const char *arg, ..., char * const envp[]);
int execv(const char *path, char *const argv[]);
int execvp(const char *file, char *const argv[]);
int execvpe(const char *file, char *const argv[], char *const envp[]);
```

Array di puntatori a carattere.

Tipicamente il primo parametro di queste API è un puntatore a carattere. Questo puntatore a carattere ovviamente ci va ad identificare un nome che noi associamo ad un programma che vogliamo far partire all'interno dell'address space. Quindi il programma che deve sostituire il contenuto corrente dell'address space.
In alcuni casi identifichiamo un "PATH NAME" ed in altri casi un "FILE NAME".

"L'argomento iniziale per queste funzioni è il percorso di un file che deve essere eseguito."

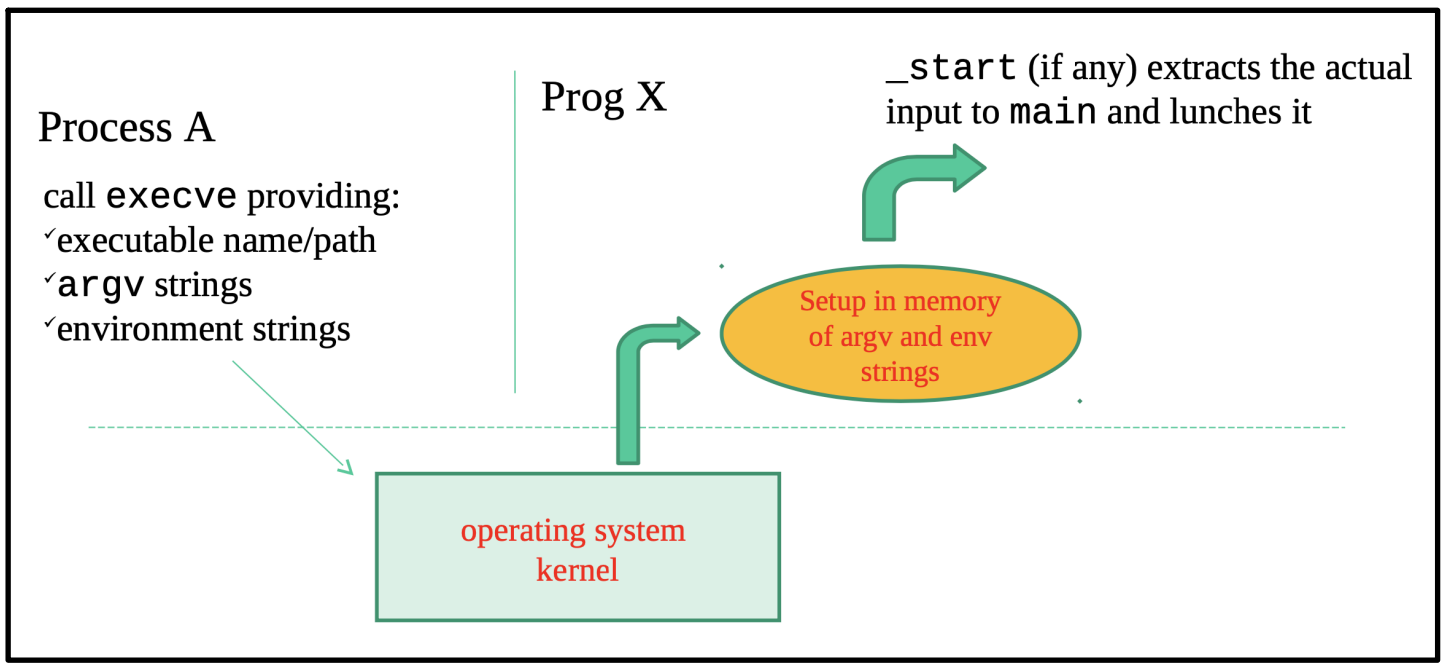
-Manuale Ufficiale

Quando noi identifichiamo un file_name lo identifichiamo in maniera diretta/ univoca;
Quando noi identifichiamo un path_name noi lo identifichiamo in maniera non univoca.

I puntatori che vengono passati come argomento corrispondono alle stringhe che devono essere inserite nell'istanza del nuovo address space che viene allocato per il processo.

Tutte queste API vanno a fare un lavoro in funzione di ciò che noi abbiamo all'interno dell'ambiente: quando noi passiamo un path, chiaramente possiamo indicare qualche cosa che deve essere eseguito a ciò che è nell'ambiente, stessa cosa con un file_name.

Quando abbiamo un'applicazione A in esercizio e quindi un processo A in esecuzione possiamo chiamare execve, per esempio, per chiamare il kernel e in modo tale da sostituire il contenuto dell'address space: possiamo specificare qual è il nome/path del programma che deve partire, ed inoltre stringhe ambientali ed argomenti.

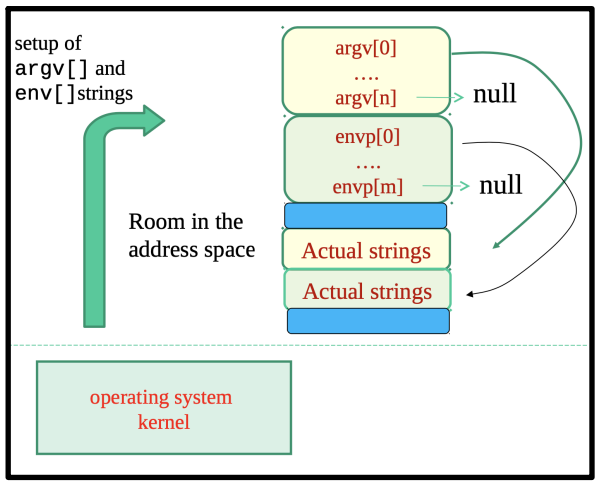


Quando noi eseguiamo una exec, e quindi mandiamo in esercizio all'interno di un address space un nuovo codice, il kernel riceve i parametri in input dal processo stesso che sta chiamando la exec, e all'interno dell'address space di questo processo, va ad inserire le stringhe che rappresentano i parametri ambientali, le stringhe che rappresentano i parametri per il main, da una parte l'array di puntatori alle stringhe ambientali, da una parte l'array di puntatori alle stringhe per il main e poi, nella CPU ,QUANDO CI DA IL CONTROLLO, all'interno dei registri di CPU ci va a scrivere dove questi array sono posizionati in memoria, e questa informazione viene comunemente estratta dagli starters per poi passare i parametri al main che noi andiamo a chiamare.

All'interno del nostro codice gli starters vanno a reperire gli indirizzi di memoria di dove sono questi due array ARGV[] ed ENVP[] in modo tale che possiamo utilizzare questa informazione all'interno del nostro programma.

Possiamo andare a stampare le stringhe che rappresentano le informazioni ambientali utilizzando environ, e nel main possiamo usare ARGV che viene passato.

Abbiamo un processo attivo A che sta eseguendo un certo programma Y, esso chiama una system_call per cambiare il programma in esecuzione all'interno dell'address space, il processo è lo stesso.
Stiamo cambiando soltanto il programma attivo all'interno del suo address space. Il vecchio address space viene assolutamente eliminato.



API per accedere ad argomenti ed ambiente

argv[] } main

envp[] → `char ** environ;` ⟸ from `unistd.h`

`getenv`
`putenv`
`setenv`
`unsetenv`

Abbiamo la possibilità di chiamare un API che quando prende il controllo se tra tutte le stringhe nella zona ambientale, va a verificare se c'è una stringa che interessa proprio alla mia applicazione.

VALORE DI RITORNO

"Se una qualsiasi delle funzioni exec() ritorna, si sarà verificato un errore. Il valore restituito è -1 e la variabile globale errno verrà impostata per indicare l'errore.
Altrimenti se l'esecuzione non ha prodotto anomalie e problemi, è evidente che non ritorna il controllo, poiché abbiamo cambiato il vestito ad un processo."

-Manuale Ufficiale 24 gennaio 1994

"Il const char *arg0 e i successivi puntini di sospensione nelle funzioni execl(), execlp() ed execle() possono essere considerati come arg0, arg1, ..., argn. Insieme descrivono un elenco di uno o più puntatori a stringhe con terminazione null che rappresentano l'elenco di argomenti disponibile per il programma eseguito. Il primo argomento, per convenzione, dovrebbe puntare al nome del file associato al file in esecuzione. L'elenco di argomenti deve terminare con un puntatore NULL"

-Manuale Ufficiale 24 gennaio 1994