Tutoraggio di Sistemi Operativi

Matteo Federico

Lezione #3

TOR VERGATA

UNIVERSITÀ DEGLI STUDI DI ROMA

In the last Episode

Scrivere un programma C che mantiene in memoria dati inseriti dall'utente, il programma dovrà mantenere questi dati in un unico array di char di grandezza 30.
Il programma dovrà supportare l'inserimento e il salvataggio dei seguenti tipi di dato: char, int, long, double.
L'utente deve poter inserire questi dati in qualsiasi ordine e quantità che desidera, ogni dato inserito dovrà essere salvato all'interno dell'array immediatamente dopo il precedente.
Quando l'utente inserisce un dato che che non entra nello spazio rimanente sull'array, il programma deve uscire e stampare tutti i byte nell'array in esadecimale.

TOR VERGATA

Side Quest

Side Quest 1

Stampare i dati nel formato inserito!

Side Quest 2

Non usare funzione “memcpy” o simili

Side Quest 3

Invece di terminare il programma quando arriviamo alla fine dell’array, mettere i byte extra all’inizio dello stesso e continuare a salvare da lì

Potete inviare le vostre soluzioni a
matteo.federico@uniroma2.it.
Scrivete nell oggetto della mail [SO] challenge 1.
Vedremo le più interessanti (in positivo e negativo) la prossima lezione.
(Sarò anonimo nel mostrare le soluzioni)

TOR VERGATA

Problema riscontrato - 1

```
1  int assunzioneInput(char appoggio_char[30], int *appoggio_int, long
       *appoggio_long, double *appoggio_double){
2      printf("inserire input:");
3      if(scanf("%d", appoggio_int) == 1){
4          char next = getchar();
5          if(next == '.'){
6              scanf("%lf", appoggio_double);
7              *appoggio_double = *appoggio_double + (double)*
       appoggio_int;
8              return 4;    // era un double
9          }
10         else{
11             ungetc(next, stdin);
12             return 2;    // era un int
13         }
14     }else{
15         if(scanf("%ld", appoggio_long) == 1)
16            return 3;
17         else
18             if(scanf("%s", appoggio_char) == 1)
19                 return 1;}
20     return 0;
21 }
22
```

Problema riscontrato - 2

```
1  int scelta, offset = 0, dimensione;
2  printf("Benvenuti nella Side Quest 2!\n");
3  while(1){
4      printf("Spazio usato %d / 30 bit\n", offset);
5      printf("\nDimmi il tipo di dato (1:int, 2: float, 3: char, 0:
       esci): ");
6      scanf("%d", &scelta);
7      if(scelta == 0) break;
8      if(offset + dimensione > 30){
9          printf("Spazio insufficiente\n");
10         break;
11     }
12
```

Fork Quest

Processi

I processi sono entitá gestite dai sistemi operativi per permettere il multitasking delle attivitá in esecuzione.
Sia linux che Windows utilizzano questo metodo per gestire dei "task" all'interno dei nostri sistemi.

```
1  #include <stdio.h>
2  #include <unistd.h>
3  int main() {
4      fork();
5      fork();
6      fork();
7      printf("Hello\n");
8      return 0;
9  }
10
```

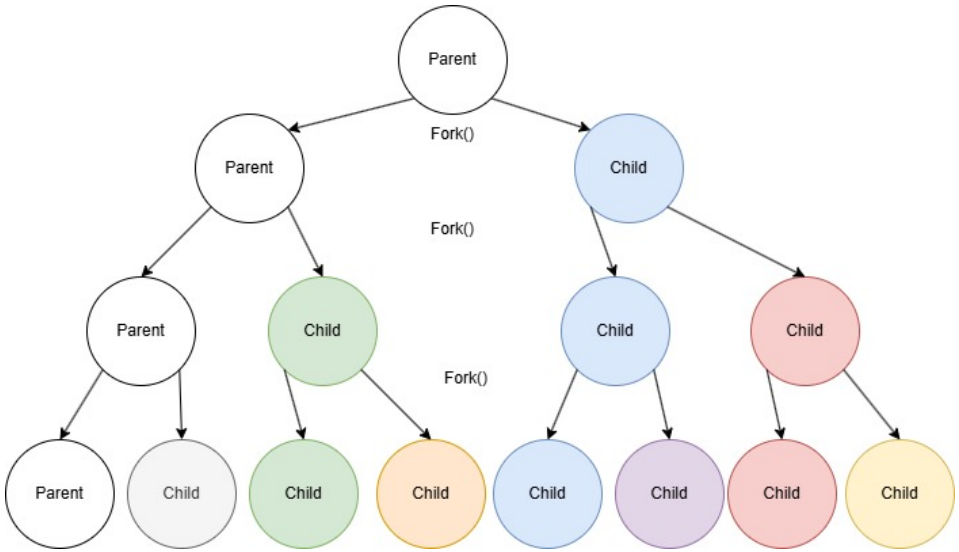
?

Quanti processi vengono creati dal seguente codice? (contando il processo principale)

Fork Quest

Risposta

8 processi coesistono nel sistema!



Fork

Come avete visto a lezione la creazione di un processo tramite fork, permette la duplicazione di tutte il programma. i processi figli che vengono a crearsi vedono lo stesso Instruction Pointer. Per tanto ogni nuovo processo inizierá la sua esecuzione dall'istruzione successiva alla sua fork.

TOR VERGATA

pseudo-Grep Quest

grep

Grep (Global Regular Expression Print) è un programma a riga di comando nei sistemi Unix-like utilizzato per cercare stringhe di testo o modelli (espressioni regolari) all'interno di file o Standard Input.

Quest

Creare un programma C che prende da linea di comando una parola e 2 file. Il programma deve creare un processo figlio che conti la ripetizione della parola data all'interno del primo file mentre il programma padre deve contare le parole all'interno del secondo file. Ognuno stampa su std output il numero di parole trovate

Side Quest

Il padre deve riuscire in qualche modo a scoprire il conteggio del figlio e deve riuscire a stampare la somma dei conteggi di entrambi i due file.

Challenge 2

Gara di corsa

Scrivere un programma C multi-thread che simula una gara di corsa.
Il programma deve supportare un numero N di corridori, ognuno rappresentato da un thread separato.
Ogni thread/corridore, alla partenza, deve attendere un numero casuale di secondi, SOLO IL PRIMO a terminare è dichiarato vincitore della gara dal main thread.
Gli altri terminano senza fare niente.
Non sono richiesti meccanismi di sincronizzazione tra i thread, ignorare situazioni anomale causate da variabili condivise.

Side Quest Challenge 2

Side Quest 1

Assegnare un nome ad ogni thread e stampare il nome del thread vincitore e non il numero

Side Quest 2

Use malloc per gestire tutte le strutture necessarie

Side Quest 3

Aggiungere classifica da stampare alla fine da parte del main thread (ignorare ancora problemi di sincronizzazione).

Potete inviare le vostre soluzioni a
matteo.federico@uniroma2.it.
Scrivete nell oggetto della mail [SO] challenge 1.
Vedremo le più interessanti (in positivo e negativo) la prossima lezione.
(Sarò anonimo nel mostrare le soluzioni)

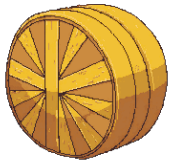
TOR VERGATA

Challenge 3

Pseudo-grep 2: La vendetta

Scrivere un programma C multi-thread che prende tramite linea di comando una parola e N files.
Il programma deve creare N thread dove ognuno deve prendere in input un solo file e cercare il numero di volte in cui la parola e' stata trovata.

Side Quest Challenge 3



Side Quest 1

Il main thread deve aggregare il numero di ripetizioni trovate.

Side Quest 2

Ogni thread mostri a schermo non solo il numero ma anche la riga e del file in cui ha trovato la parola. (potete assumere che se la parola sia a cavallo tra due righe restituire la righa su cui inizia).

Potete inviare le vostre soluzioni a
matteo.federico@uniroma2.it.
Scrivete nell oggetto della mail [SO] challenge 1.
Vedremo le più interessanti (in positivo e negativo) la prossima lezione.
(Sarò anonimo nel mostrare le soluzioni)

TOR VERGATA