Sistemi Operativi – Lab 5 11.11.19/12.11.19 - A.A. 2019/2020 - Prof. L. Sterpone

Goal: gestione preliminare dei processi, albero di generazione dei processi e control flow graph.

Esercizio 1 (albero di generazione dei processi): Rappresentare il grafo di controllo del flusso e l'albero di generazione dei processi dei seguenti tratti di codice. Indicare inoltre l'output prodotto su video. Verificare il risultato predetto tramite esecuzione.

```
A.for (i=1; i<=2; i++)
{
   if (!fork ())
      printf ("%d\n", i);
}
printf ("%d\n", i);
```

```
B.for(i=3; i>1; i--)
{
   if (fork ())
    printf ("%d\n", i);
}
printf ("%d\n", i);
```

```
C.for (i=0; i<2; i++)
{
   if (fork ())
      fork ();
}
printf ("%d\n", i);
```

```
D.for (i=2; i>=1; i--)
{
   if (!fork ())
      printf ("%d\n", -i);
   else
      printf ("%d\n", i);
}
```

Esercizio 2 (albero di generazione dei processi e control flow graph – simile esame 21.09.16):

Rappresentare l'albero di generazione dei processi e il control flow graph del seguente programma C nel caso in cui x sia pari a 5 e n sia pari a 2. Determinare l'ordine di stampa dei messaggi a video e l'azione della system call `system()`.

Come va modificato il codice per poter gestire il caso in cui un processo figlio non è creato ?

Che cosa succede al variare dei parametri n ed x ?

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4 #include <unistd.h>
5 #include <sys/wait.h>
6
7 int main (int argc, char *argv[]) {
8 int i, n, x;
9 char str[50];
10
11 n = atoi (argv[1]);
12 x = atoi (argv[2]);
13 printf ("run with n=%d\n", n);
14 fflush (stdout);
15 for (i=0; i<n; i++) {
16      if (fork () > 0) {
17         printf ("%d", n-1);
18         sleep(x);
19      } else {
20         sprintf (str, "%s %d %s", argv[0], n-1, argv[2]);
21         system (str);
22           }
23       }
24
25 exit (0);
26 }
```

Sistemi Operativi – Lab 6 18.11.19 - A.A. 2019/2020 - Prof. L. Sterpone

Goal: Utilizzo delle system call exec, fork e system. Approfondimento grafo di precedenza.

Esercizio 1 (system call exec):

Considerando il codice seguente, se ne descriva il funzionamento indicando che cosa produce su video e per quale motivo.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
int main (int argc, char **argv)
{
  int i = 0;
  fprintf (stdout, "%d %d\n", getpid(), ++i);
  execl (argv[0], argv[0], (char *) 0);
  fprintf (stdout, "End program\n");
  return (1);
}
```

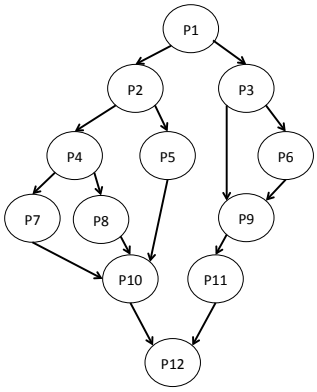
Esercizio 2 (system call fork, exec e system):

Riportare l'albero di generazione dei processi a seguito dell'esecuzione del seguente tratto di codice C. Si indichi inoltre che cosa esso produce su video e per quale motivo.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
int main () {
  char str[100];
  int i;
  fork();
  for (i=0; i<2; i++){
    if (fork()!=0) {
      sprintf (str, "echo system with i=%d", i);
      system (str);
    } else {
      sprintf (str, "exec with i=%d", i);
      execlp ("echo", "myPgrm", str, NULL);
    }
  }
  return (0);
}
```

Esercizio 3 (grafo di precedenza):

Realizzare con le system call fork e wait il grafo di precedenza illustrato nella seguente figura. Ogni processo P produca un messaggio di stampa in cui viene stampato l'indice del processo il PID e il PID del padre. Verificare che le precedenze siano rispettate inserendo delle system call sleep nei vari rami del programma.



Sistemi Operativi – Lab 7 25.11.19 - A.A. 2019/2020 - Prof. L. Sterpone

Goal: gestione dei segnali, creazione dell'handler, sincronizzazione dei processi tramite segnali.

Esercizio 1 (signal e comando kill):

Scrivere un programma C che riceva in input da tastiera due numeri interi, a e b, e ne stampi a video:
- la somma "a+b" solo quando riceve il segnale SIGUSR2;
- la differenza "a-b" quando riceve il segnale SIGUSR1.

Il programma termina quando riceve SIGINT. Utilizzare il comando kill per inviare i segnali al processo.

Esercizio 2 (signal handler):

Scrivere un programma C che riceva in input da linea di comando il PID del programma dell'esercizio precedente ed un comando (vedi tabella sotto) e invii il relativo segnale al processo <PID>:
Comando: segnale
"somma": SIGUSR2
"differenza": SIGUSR1
"fine": SIGINT

Esercizio 3 (signal e pause):

Scrivere un programma C che: crea due figli, ne stampa i relativi PID ed attende che entrambi terminino intercettando SIGCHLD.
Il primo figlio legge i primi 50 byte dal file "son1.txt", li stampa a video e termina.
Il secondo figlio legge i primi 50 byte dal file "son2.txt", e li stampa a video, attende 5 secondi e termina.
Nota 1: Creare i due file son1.txt e son2.txt prima di eseguire il programma.
Nota 2: Osservare l'ordine di visualizzazione delle informazioni. Utilizzando I segnali, "forzare" la visualizzazione dell'intero contenuto di son2.txt prima della visualizzazione di son1.txt.

Esercizio 4 (signal handler):

Scrivere un programma C che:
Crea un figlio;
Intercetta tramite handler apposito i segnali SIGUSR1, SIGUSR2; quando riceve il segnale x, visualizza "Ricevuto il segnale x";
Riceve in input su riga di comando una sequenza di interi x1,x2,…xk.
In un ciclo infinito ad intervalli regolari, invia al processo figlio uno dei segnali ricevuti in input.

Il processo figlio:
Intercetta I segnali SIGUSR1, SIGUSR2 e SIGINT;
Blocca tutti I segnali eccetto SIGUSR1, SIGUSR2 e SIGINT;
Quando riceve SIGUSR1 invia al padre SIGUSR2;
Quando riceve SIGUSR2 invia al padre SIGUSR1;
Quando riceve SIGINT invia al padre SIGINT.

Esercizio 5 (signal handler):

Scrivere un programma C che:
Riceve su riga di comando un intero n, crei n figli ed ad intervalli regolari di 2 secondi visualizzi il proprio PID e il PID del figlio i-esimo. Invii al figlio I-esimo un segnale.

I processi figli:
Quando ricevono il segnale inviato dal padre visualizzino il PID del padre, il proprio PID e l'intero associato al segnale.

Sistemi Operativi – Lab 8 – 02.12.19 - A.A. 2019/2020 - Prof. L. Sterpone

Goal: sincronizzazione dei processi tramite segnali, uso della pipe.

Esercizio 1 (signal):

Scrivere un programma in linguaggio C in cui il processo padre (master) crea due processi figli (figlio 1 e figlio 2), attende la terminazione di entrambi, visualizza un messaggio e termina l'esecuzione.
Il processo figlio 1, apre un file di testo test_1.txt, legge il file una riga per volta e la stampa a video; il processo figlio 2, apre un file di testo test_2.txt, legge il file una riga per volta e la stampa a video.

Nota: Provare a eseguire tale esercizio in una seconda alternativa: supponendo di ottenere un output su schermo interlacciato dei due file.

Esercizio 2 (signal e kill):

Scrivere un programma C che attende all'infinito. Il processo padre dovrà gestire tramite apposito handler due segnali: SIGCHLD e SIGINT e comporarsi nel seguente modo:

- dovrà creare un processo figlio, attendere la ricezione del segnale SIGCHLD e terminare.
- attende anche la ricezione del segnale SIGINT (generato dall'esterno) e invia al figlio a sua volta lo stesso segnale

Il processo figlio si attiva alla ricezione del segnale SIGINT, stampa il suo PID e termina.

Esercizio 3 (pipe):

Scrivere un programma C in cui il padre crea un processo figlio, riceva dal figlio una sequenza di stringhe che visualizza e termina quando il figlio ha terminato la sequenza.
Il figlio dovrà aprire un file di testo, leggere il file una riga per volta, inviare la riga al padre e terminare quando non vi sono più caratteri da leggere nel file. Gestire la comunicazione tra padre e viglio attraverso una pipe.

Esercizio 4 (pipe e system call exec):

Scrivere due programmi in linguaggio C. Il programma prende una stringa in ingresso e la stampa in uscita utilizzando il comando *echo* attraverso una *exec*. Il secondo programma riceve una stringa dallo stdin e la stampa in uscita tutta maiuscola. Utilizzando obbligatoriamente ed esclusivamente questi due programmi convertire una stringa da minuscolo a maiuscolo.

Esercizio 5 (system call signal, kill, pause e sleep):

Si realizzi un programma C che crei due processi figli. Il figlio 1 invia il segnale SIGUSR1 al fratello passando per il padre. Il figlio 2 deve catturare il segnale inviatogli dal fratello e rispondere con il segnale SIGUSR2 sempre tramite il padre. Stampare a video delle stringhe di testo (e.g., come quelle utilizzate a lezione "Sono nel padre", "Sono nel figlio 1",…) per dimostrare il corretto funzionamento del programma.