Primer Certamen
Tiempo 11:30 hrs - 13:00 hrs.
Responder un problema por página

1.- (30 puntos Shell Script) Usted recordará el script sde: Save Disk Space, el cual reemplaza los archivos de un directorio por su versión comprimida cuando éstos no han sido accedidos por un tiempo. Luego de un tiempo de usar este script, usted desea acceder a un archivo en su cuenta. Para buscarlo de entre las versiones comprimidas (de extensión *.tar.gz), se le pide desarrollar el script expand.sh el cual recibe como argumento el nombre del archivo a buscar y el directorio a partir del cual se inicia la búsqueda.
$ expand.sh <dir> <file>
Una vez encontrado el primer archivo con este nombre, expand.sh muestra la ruta (path) y nombre del archivo comprimido que lo contiene.
Revise la opción -t de tar.

```
#! /bin/bash
dir=$1
file=$2
for i in $dir/*.tar.gz
do
        if tar -tf $i | grep $file
        then
                echo "Your file $file is in $i"
                exit
        fi
done
for i in $dir/*
do
        if test -d $i
        then
                $dir/$0 $i $file
        fi
done
```

Otra solución pudo ser (con elementos nuevos en el manejo de strings):

```
#! /bin/bash
dir=$1
file=$2
for i in $dir/*
do
        if test -d $i
        then
                $dir/$0 $i $file
        elif [ ${i: -6} = "tar.gz" ] && tar -tf $i | grep $file
                then
                        echo "Your file $file is in $i"
                        exit
        fi
done
```

2.- (35 puntos Manejo de Procesos) La creación de procesos toma tiempo, como también la invocación a alguna de las funciones exec. Para medir el tiempo ocupado en la creación de un proceso hijo y luego su "mutación" a otro, un Sansano sugiere correr como proceso hijo un programa de nombre "nada" el cual contiene una función main vacía.

Cree un programa en C (tiempo) que muestre el tiempo requerido para crear un proceso hijo, mutarlo al programa "nada" y esperar a que éste termine. Para ello Sansano sugiere medir el tiempo justo antes de invocar a fork y luego cuando el proceso hijo mutado a "nada" termina. Su programa "tiempo" muestra por pantalla la diferencia entre estos dos tiempos.
Ayuda: Para medir el tiempo considere usar el llamado gettimeofday().
Nota: No cuestione la calidad del método sugerido por Sansano.

```
#include <sys/wait.h>
#include <stdio.h>
#include <stdlib.h> /* for exit */
#include <unistd.h>
#include <sys/time.h>

struct timeval tf;

static void sig_child(int signo) {
  gettimeofday(&tf, NULL);
}

int main(void) {
   pid_t pid;
   int status;
   int timespan;
   struct timeval ti;

   if (signal(SIGCHLD, sig_child) == SIG_ERR)
      exit(-1);
   gettimeofday(&ti, NULL);
   if ((pid = fork() ) < 0)
      exit(-1);
   if (pid == 0) {  /* child */
     if (execlp("nada","nada", (char *) 0) < 0)
        exit(-1);
   }
   pause();
   timespan = 1000*(tf.tv_sec-ti.tv_sec)+ tf.tv_usec-ti.tv_usec;
   printf("Time span: %d [us]\n", timespan);
   wait(&status);  /* we can omit it since we finish */
}
```

3.- (35 puntos) Contexto: En arquitecturas de varios núcleos, varias hebras pueden ser ejecutadas en forma paralela. Para verificar esto en forma experimental, otro Sansano sugiere correr una aplicación que ejecuta una tarea paralelizable la cual se pueda descomponer en un número variable de hebras. Luego sugiere correr esta aplicación varias veces con distintos número de hebras y en cada caso medir su tiempo de ejecución. El análisis de estos tiempos permitiría concluir el número de hebras paralelas que corren en una máquina.
Pregunta: Cree una aplicación (coreNum) que calcule la suma de los enteros del 1 al 1.000 usando k hebras. Es decri cada hebra debe hacerse cargo de una rango de números pasados como parámetros y entregar su resultado en el parámetro retornado. La aplicación y muestra por pantalla: valor de la suma total y el número de hebras usadas. La aplicación se ejecuta usando:
$ coreNum <k> , donde k es el número de hebras a utilizar para realizar la suma.

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#define MAXNUM 100000

typedef struct  {
   int a, b, sum;
} PARAM;

void * partialSum(void * arg) {
  int i;
  PARAM * task=(PARAM*)arg;
  task->sum=0;
  for (i=task->a; i<= task->b; i++)
   task->sum+=i;
  return((void*)&(task->sum));
}

int main(int argc, char * argv[]) {
  int err, t, k, a, workSize, sum;
  pthread_t * tid;
  PARAM *task;
  int* pSum;

  k = atoi(argv[1]);
  tid = (pthread_t*) calloc(k, sizeof(pthread_t));
  task = (PARAM *) calloc(k, sizeof(PARAM));
  workSize = MAXNUM%k==0 ? MAXNUM/k:MAXNUM/(k-1);
  a=0;
  for (t=0; t<k ;t++) {
     task[t].a=a+1;
     a+=workSize;
     task[t].b= a<=MAXNUM ? a:MAXNUM;
     err = pthread_create(tid+t, NULL, partialSum, task+t);
     if (err != 0)
       exit(-1);
  }
  sum=0;
  for (t=0; t<k;t++){
    pthread_join(tid[t], (void**)&pSum);
    sum+=*pSum;
  }
  printf("La suma es %d y se usaron %i hebras.\n", sum,k);
}
```