Por la sostenibilidad de la CALIDAD Misional Universitaria | UNIVERSIDAD NACIONAL DE COLOMBIA SEDE MEDELLÍN FACULTAD DE MINAS

Sistemas en Tiempo Real

PRÁCTICA # 2: PROGRAMACIÓN CONCURRENTE

1. Introducción

En esta práctica se abordará el tema de concurrencia, específicamente los aspectos básicos relativos al concepto de proceso e hilos mediante el uso de los servicios que proporciona el sistema operativo. En nuestro caso, el sistema de pruebas y desarrollo es un sistema basado en Linux donde se utilizará un estándar de manejo de hilos POSIX.

2. Objetivos

- Adquirir habilidades en la creación y manejo de procesose hilos (threads) concurrentes bajo el estándar POSIX.
- Adquirir habilidades en el uso de mecanismos de sincronización de hilos usando el estándar POSIX.

3. Problemas Propuestos

3.1. Parte 1: Procesos

A. **Fork() simple (12%)** : Copie, compile y ejecute el siguiente código.
   - Explique porque si el valor de "x" en el proceso padre es igual o distinto al del proceso hijo.
   - ¿Existe alguna forma de predecir los ID del proceso que serán asignados al proceso hijo y al proceso padre? Explique porque si o porque no

```
/* Programa bien_facil */

#include  <stdio.h>
#include  <string.h>
#include  <unistd.h>
#include  <sys/types.h>
#include <stdlib.h>
#include <time.h>

#define   MAX_COUNT  10
#define   BUF_SIZE   120

void  main(void)
{
     pid_t  pid;
     int    i;
     int    x=0;


     char   buf[BUF_SIZE];
     fork();
     pid = getpid();
     for (i = 1; i <= MAX_COUNT; i++) {
         x= rand();
          sprintf(buf, "Esta linea es de pid %d, valor = %d, valor de x
es %d\n", pid, i, x);
          write(1, buf, strlen(buf));

     }
}
```

B. **Fork() no tan simple (12%) :** Copie, compile y ejecute el siguiente código.
- indique cuántos procesos se crearon. ¿Es igual a su respuesta anterior? Si no es, ¿por qué?
- Modifique el programa para se creen tres copias únicamente

```
/* Programa bien_facil */

#include  <stdio.h>
#include  <string.h>
#include  <unistd.h>
#include  <sys/types.h>
#include <stdlib.h>
#include <time.h>


#define   MAX_COUNT  2
#define   BUF_SIZE   120

void  main(void)
{
     pid_t  pid;
     int    i;
     int    x=0;


     char   buf[BUF_SIZE];
     fork();
     fork();
     fork();
     pid = getpid();
     for (i = 1; i <= MAX_COUNT; i++) {
         x= rand();
          sprintf(buf, "Esta linea es de pid %d, valor = %d, valor de x
es %d\n", pid, i, x);
          write(1, buf, strlen(buf));

     }
}
```

C. **Fork() complejo (12%):** Copie, compile y ejecute el siguiente código.
- Explique el funcionamiento del programa
- Explique la salida que obtuvo del programa “gnarls”, ¿qué muestra?
- Explique si después de ejecutar el programa “gnarls” 20 veces, considera que el manejo del procesador es justo o injusto.

```
/* programa gnarls */

#include  <stdio.h>
#include  <stdlib.h>
#include  <unistd.h>
#include  <sys/types.h>

#define   MAX_COUNT  20
char *barras[5] = { "*","**","***","****","*****" };

void  ChildProcess(void);                /* prototipo proceso hijo  */
void  ParentProcess(void);               /* prototipo proceso padre */

void  main(void)
{
     pid_t  pid;

     pid = fork();
     if (pid == 0)
          ChildProcess();
     else
          ParentProcess();
}

void  ChildProcess(void)
{
     int   i;

     for (i = MAX_COUNT; i > 0 ; i--)
     {
          printf("\t\th-%s\n",barras[i % 5]);
          sleep(i % 5);
     }
     printf("   *** Fin proceso hijo ***\n");
}

void  ParentProcess(void)
{
     int   i,h;

     for (i = 1; i <= MAX_COUNT; i++)
     {
          printf("\tp-%s\n",barras[i % 5]);
          sleep(i % 5);
     }
     printf("*** Fin del Padre ***\n");
}
```

D. **Comparando tiempos (9%):** Copie y compile los siguientes códigos y responda: ¿Con cuál de las dos opciones obtienes un mejor rendimiento?. Razona la respuesta

El siguiente código muestra un proceso que crea 100 procesos hijos, midiendo el tiempo medio que tarda en la creación de cada uno de ellos.

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int main() {
    struct timeval t0, t1;
    int i = 0;
    int id = -1;
    gettimeofday(&t0, NULL);
    for ( i = 0 ; i < 100 ; i++ ) {
        id = fork();
        if (id == 0) return 0;
    }
    if (id != 0) {
        gettimeofday(&t1, NULL);
        unsigned int ut1 = t1.tv_sec*1000000+t1.tv_usec;
        unsigned int ut0 = t0.tv_sec*1000000+t0.tv_usec;
        printf("%f\n", (ut1-ut0)/100.0); /* Tiempo medio en microsegundos */
    }
}
```

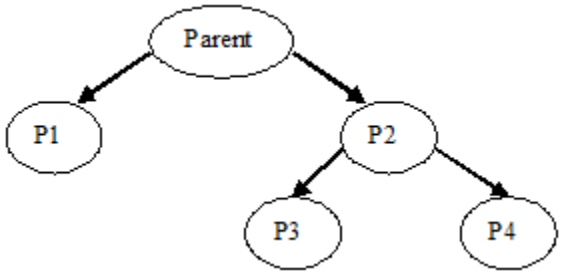
El siguiente código muestra un hilo que crea 100 hilos hijos, midiendo el tiempo medio que tarda en la creación de cada uno de ellos.

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <pthread.h>

struct timeval t0, t1;
double media = 0.0;

void *hilo(void *arg) {
    gettimeofday(&t1, NULL);
    unsigned int ut1 = t1.tv_sec*1000000+t1.tv_usec;
    unsigned int ut0 = t0.tv_sec*1000000+t0.tv_usec;
    media += (ut1-ut0);
}

int main() {
    int i = 0;
    pthread_t h;
    for ( i = 0 ; i < 100 ; i++ ) {
        gettimeofday(&t0, NULL);
        pthread_create(&h, NULL, hilo, NULL);
        pthread_join(h, NULL);
    }
    printf("%f\n", (media/100.0)); /* Tiempo medio en microsegundos */
}
```

E. **Más de Fork() (15%):** Considere el siguiente fragmento de código.

- ¿Cuántos procesos (excluido el padre) son creados? Dibuje el árbol de procesos.
- Corregir el código para replique el árbol de procesos mostrado.
- Modificar el código para asegurar de que el proceso P4 se pone en marcha sólo después de que ha terminado le proceso P1.

```
if ((p1_pid = fork())== 0)
    printf("I am P1 and I am proud of it.");
} else {
    if ((p2_pid = fork()) == 0) {
        p3_pid == fork();
        printf("I am P3. I like it.");
        p4_pid == fork();
        printf("I am P4. Get used to it.");
    } else
        printf("I am the parent process, obey or die!");

}
```



3.2. Parte 2: Hilos(1)

A. **Creación básica de hilos (15%):** Descargue el archivo prueba1.c y compile
> gcc -std=c99 -D_POSIX_C_SOURCE=200112L -o prueba1 prueba1.c

Responda a las siguientes preguntas:

- ¿Da algún error de compilación? Si es así, solucionelo.
- Enumere las funciones de la biblioteca Pthreads que se están usando en este programa.
- ¿Qué hace la función pthread_create? Describa cada uno de sus parámetros y los parámetros de los atributos de una hebra.
- ¿Qué hace la función pthread_exit? Describa cada uno de sus parámetros.
- Explique el significado de cada uno de los argumentos que le ha pasado a gcc para compilar y enlazar el programa.
- Experimente con la creación de más hilos sobre este código, su compilación y el lanzamiento de su ejecución. ¿Cuántos hilos como máximo le permite crear el sistema? ¿Alcanza un punto de saturación? ¿Qué error da el sistema?

B. **Terminación de hilos (10%):** Un hilo puede esperar a que otros hilos terminen. La información sobre si un hilo es o no "esperable" está almacenada en sus atributos (variable de tipo *pthread_attr_t*). En algunas implementaciones este atributo está establecido por omisión, pero en otras no lo está. De forma que si desea escribir código portable, es decir, si desea que la función *pthread_join* funcione siempre como tiene previsto, deberá establecer el valor del atributo correspondiente.

- Descargue el código pruebajoin.c, compílelo y observe su funcionamiento.

- Experimente con el no establecimiento del atributo "detached" y vuelva a compilar y ejecutar el código.
- ¿Concluiría que está establecido por defecto este parámetro en esta implementación de pthreads? Razone su respuesta.

3.3. Parte 3: Comparación entre procesos e hilos (15%)

Se desea desarrollar una aplicación que debe realizar dos tareas que se pueden ejecutar de forma independiente. Los códigos de estas dos tareas se encuentran definidos en dos funciones cuyos prototipos en lenguaje de programación C, son los siguientes:

```
void tarea_A(void);
void tarea_B(void);
```

Se pide:
Programar la aplicación anterior utilizando tres modelos distintos: un programa secuencial ejecutado por un único proceso, un programa que crea procesos para desarrollar cada una de las tareas, y un programa que realiza las tareas anteriores utilizando procesos ligeros. En cualquiera de los tres casos la aplicación debe terminar cuando todas las tareas hayan acabado.

4. Sección Opcional

Sincronización de Hilos: utilización básica de semáforos: Escriba el código de dos hilos que deben alternar de forma estricta su ejecución. Resuelva el problema utilizando semáforos

4. Condiciones de Entrega

Se pide:

- Entrega de un documento : incluyendo las preguntas y sus respuestas
- Entrega de los códigos fuente.

Todo ello en un único fichero ZIP.

La entrega debe realizarse antes del miércoles 18 de abril