Trabajo Práctico Nº1

Sistemas Operativos

Contenido

Condiciones para Aprobar

Entregable

Para la evaluación del presente trabajo, deben realizar los siguientes puntos:

- **Mail :** Enviar en formato digital con asunto “TP1: Sor1-1S-2019” e indicar en el cuerpo los integrantes del grupo. Adjuntar el código fuente de su implementación y un informe del trabajo realizado, dificultades encontradas y pseudocódigo de las soluciones propuestas.
- **Defensa/Demo :** El dia de la entrega debe mostrar a su docente la solución enviada funcionando en la computadora del laboratorio.

Puntaje / Calificación

Se califica con las notas:

★ I (insuficiente),
★ A- (aprobado menos, no puede tener dos A- en la cursada),
★ A (aprobado)
★ A+ (aprobado más, redondea para arriba la nota final en caso de promocionar)

Recuperatorio

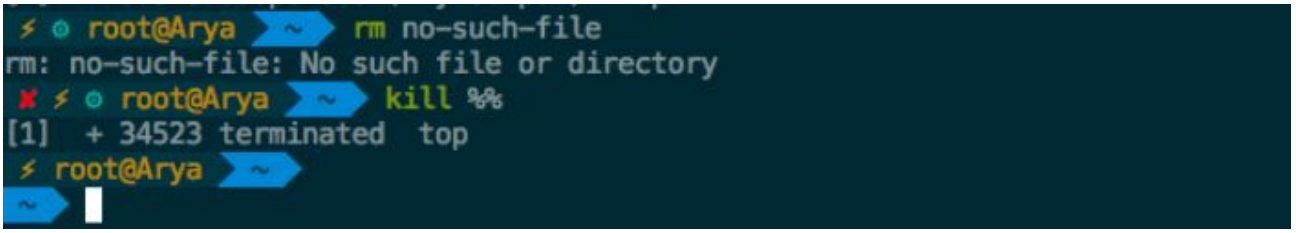
En caso de no aprobar tiene un plazo de dos semanas para entregar el TP con las correcciones indicadas durante la demo (en recuperatorio no se pone A+). La demo del tp recuperatorio es de forma individual.

Introducción

Los objetivos de este trabajo son:

- Familiarizar al alumno con la línea de comandos.
- Familiarizar al alumno con las nociones de Proceso, Thread y Semáforo.
- Fomentar el trabajo en equipo

Enunciado 1ra Parte: Shell



En esta parte se programará un script bash que permite realizar operaciones complicadas de manera fácil y automática. Las opciones deben aparecer en forma de un menú donde el usuario elije una opción y el script realiza todo el trabajo.

Ejercicio 1 : Sistema Operativo Linux

- Instalar una distribución de Linux a su elección (se recomienda debian, y en caso de tener problemas de drivers se recomienda lubuntu, ubuntu y mint). Puede realizarlo desde una imagen live .iso en una máquina, una máquina virtual con virtualbox ó en un usb pero esta última opción debe realizarse con persistencia.

- Dentro de su nuevo sistema crear un usuario diferente de root y agregarlo a sudoers con: adduser nombre_usuario sudo

- Con el nuevo usuario actualizar la lista de repositorios del sistema con: sudo apt-get update
  y actualizar los programas del sistema con: sudo apt-get upgrade

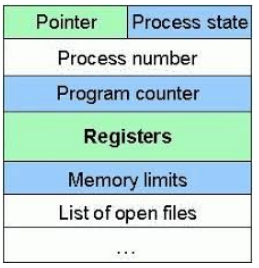
- Instalar los programas gcc, vim, nano, gedit, emacs, git. Por ejemplo con: sudo apt-get install gcc

Ejercicio 2 : Shell y supermenu

- Bajar el ejemplo proporcionado para este tp (ver supermenu en la sección Código Base), dar permisos de ejecución al usuario propietario (grupos y otros solo deben tener acceso de lectura)

- El supermenu proporcionado interactúa con una cuenta git ficticia, Uds. deben adaptarlo para interactuar con su cuenta.
  - Crear una cuenta Gitlab y un proyecto

    - Modificar el supermenu para que funcione con su proyecto



- Agregar al supermenu una nueva función que le permita obtener información de PCB. Esta información se encuentra disponible en texto plano en /proc/PID/status. A partir de un proceso dado por parámetro mostrar solamente:
    - Nombre del proceso, Pid y PPid
    - Estado
    - Información de Context Switch

- El resto de los ejercicios de este TP deben agregarse como opciones del supermenu. Utilizar al menos una vez los operadores
    - pipe | ,
    - redirección > ,
    - redirección concatenando >>
    - y los comandos
        - grep,
        - ls,
        - cat.

Ejercicio 3 : Editores de texto y habilidades básicas

- Usar editor vim. Aprender y practicar los comandos para entrar y salir de modo edición y modo comandos.

- Usar el editor nano. Cómo copiar y pegar una línea, como salir y guardar los cambios.

- Tipear sin mirar y mantener una postura correcta. Completar el nivel Beginner de https://www.typing.com/

Enunciado 2da Parte: Procesos y Semáforos

En los siguientes ejercicios:
- procesos,
- threads
- y semáforos.

Ejercicio 1 : Procesos y Fork

Fork es la llamada al sistema que permite crear nuevos procesos. Por ejemplo, fork es la base de la implementación de un intérprete de comandos. Ante cada comando, el intérprete de comandos crea un nuevo proceso que ejecute el comando solicitado por el usuario.

- Describir y justificar el output del siguiente programa

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
int main()
{
   fork();

   printf("Hola mundo!\n");
   return 0;
}
```

- Crear una bomba fork. Cuántos procesos se pueden crear? Agregar seguridad a su sistema con el comando ulimit.

- Crear un programa que simule ser un intérprete de comandos shell. Su programa debe imprimir un prompt en la terminal y luego debe quedarse esperando a que el usuario ingrese un comando shell. Dicho comando debe ejecutarse y mostrarse en la misma terminal. Puede usar las siguientes instrucciones:

```
//para leer un comando input
#define MAX_COMMAND_LENGTH 100
char cmd[MAX_COMMAND_LENGTH];
cmd=fgets(cmd, sizeof(cmd), stdin)

//ejecutar los parámetros
execvp(params[0], params);

//remover el enter de tu comando
if(cmd[strlen(cmd)-1] == '\n') {
        cmd[strlen(cmd)-1] = '\0';
}

//dividir el comando en un array de parámetros
#define MAX_NUMBER_OF_PARAMS 10
char* params[MAX_NUMBER_OF_PARAMS + 1];
parseCmd(cmd, params);

//donde:
void parseCmd(char* cmd, char** params)
{
  for(int i = 0; i < MAX_NUMBER_OF_PARAMS; i++) {
        params[i] = strsep(&cmd, " ");
        if(params[i] == NULL) break;
  }
}
```

Ejercicio 2 : Threads y Semáforos

Exclusión Mutua

El problema de los lectores y los escritores consiste en dos o más procesos que comparten una base de datos. Uno de los procesos sólo accede a escribir mientras que el otro proceso accede solamente a leer los datos. El problema surge cuando el proceso que escribe no terminó de escribir y al mismo tiempo el proceso lector empieza a leer, en esta situación el proceso lector puede llegar a leer datos incompletos.

- Implementar el problema de los lectores y escritores basándose en el siguiente pseudocódigo.

Thread escritor	**Thread lector**

while (true) {
. . .
/* escribir datos */
. . .
}
	while (true) {
	. . .
	/* leer datos */
	. . .
	}

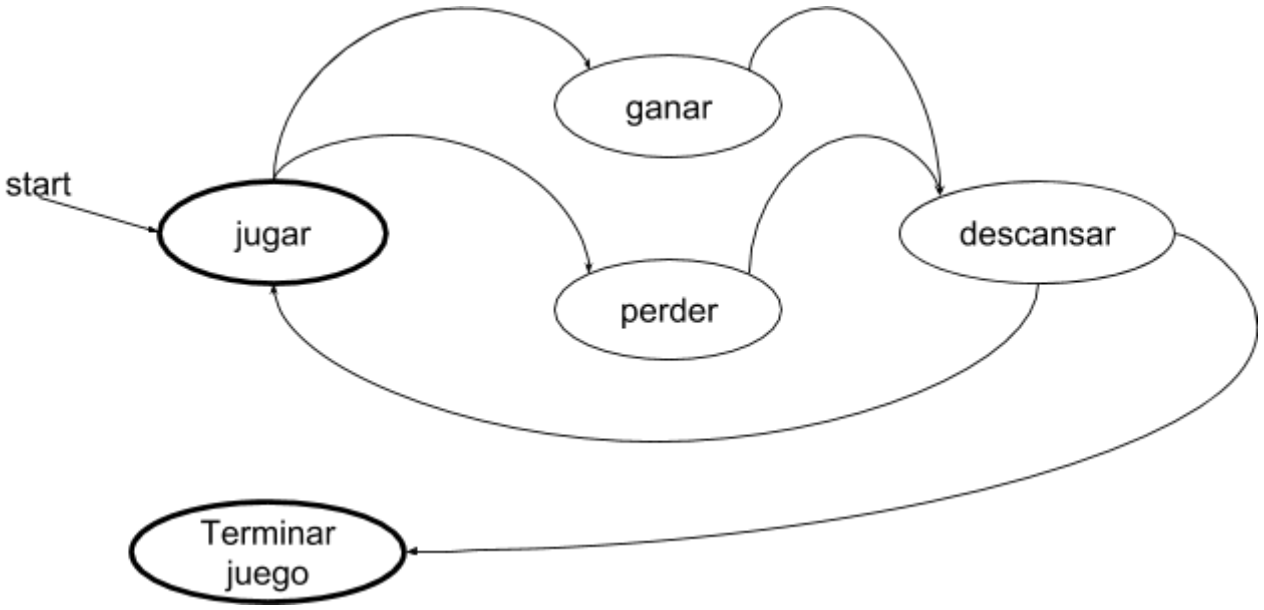
- Realizar los siguientes puntos:
    - Identificar la condición de carrera entre los threads
    - Identificar la sección crítica
    - Resolver el problema usando mutex, hacer pseudocódigo. Es necesario agregar semáforos de sincronización?

Sincronización

Cuando tenemos muchos procesos que se ejecutan en paralelo puede surgir la necesidad de sincronizarlos para obtener un comportamiento general. Por ejemplo, considerar diferentes procesos que disparan eventos de un juego, tales eventos pueden ser:

- jugar,
- ganar,
- perder,
- descansar,
- terminar el juego

Por cada evento existe un proceso que lo realiza y que además imprime por pantalla el nombre del evento. Si los procesos se ejecutan de manera sincronizada, se esperaría obtener las siguientes transiciones de estado:



El diagrama anterior se puede interpretar como una abstracción o resumen de los eventos que suceden en la naturaleza o en la vida. Jugar puede ser buscar alimento, pareja ó trabajo. Siempre se puede ganar o perder, alguno de los dos eventos puede suceder **uno ó el otro**. Después se puede descansar para volver a intentarlo. Luego de un número limitado de intentos llega el fin. Son ciclos que suceden en la naturaleza.

- Suponer que cada evento (jugar, ganar, perder, descansar y terminar) es realizado por un thread como indica el siguiente pseudocódigo:

Thread 1	Thread 2	Thread 3	Thread 4

while(condition){
…….
printf("jugar");
…….
}
	while(condition){
	…….
	printf("ganar");
	…….
	}
		while(condition){
		…….
		printf("perder");
		…….
		}
			while(condition){
			…….
			printf("descarsar");
			…….
			}

El objetivo es sincronizar los threads para que se imprima por pantalla:

jugar (ganar ó perder) descansar
jugar (ganar ó perder) descansar
jugar (ganar ó perder) descansar
…
repetir lo anterior un número de veces y luego imprimir **terminar** (con otro thread)

- Implementar en C el pseudocódigo anterior de tal forma que la secuencia anterior se repita una cantidad de veces que se fija por parámetro.
- Dejar corriendo el programa un tiempo determinado. Guardar el output y contar cuántas veces sucedió el evento ganar y cuantas veces perder. Hay azar? Justificar.
- Se suceden rachas ganadoras o perdedoras? Porque?
- Agregar el modo jugar bien (se le asigna prioridad al evento ganar). Repetir los experimentos anteriores. Qué cambios suceden?

Para asignar prioridad puede basarse en el siguiente ejemplo:

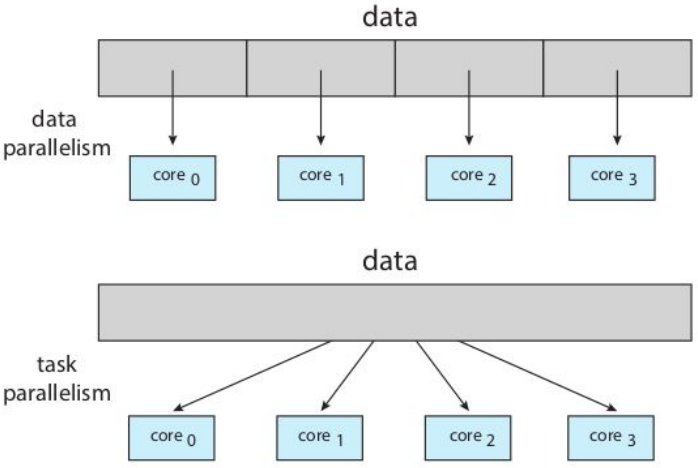
```
pthread_attr_init (&attr);/* setear los atributos por defecto*/
pthread_attr_getschedparam (&attr, &param); /*obtener los parametros de scheduling*/
(param.sched_priority)++;/*aumentar la prioridad*/
pthread_attr_setschedparam (&attr, &param);/*setear el nuevo parametro*/
```

Ejercicio 3 : El Poder del Paralelismo

El paralelismo consiste en ejecutar más de un proceso al mismo tiempo. Esto es posible lograr gracias a la existencia de procesadores con más de un core, es decir, en las arquitecturas multiprocesador-multicore.

El paralelismo se clasifica en dos tipos:

- Paralelismo de datos: se cuenta con una cantidad muy grande de datos, como es tan grande se divide en partes y cada parte se reparte entre procesos para que realicen cierta tarea.
- Paralelismo de tareas: se cuenta con un cuerpo de datos sobre los que hay que realizar diferentes tareas, usualmente son tareas independientes una de otra, por lo tanto se puede asignar una tarea a un proceso diferente.



Se cuenta con un archivo que contiene los primeros 10.000 dígitos del famoso número Pi (link)
También está disponible un código C que carga dichos dígitos en un array listo para usar.

Se pide:

- Implementar un ejemplo de paralelismo de datos. Por un lado calcular la suma total de los primeros dígitos de Pi.
    - Realizar una implementación secuencial con un For
    - Realizar una implementación con paralelismo de datos, es decir, debe dividir el array en subarrays de tamaño igual y asignar cada porción del array a un thread diferente.

- Implementar un ejemplo de paralelismo de tareas. Debe realizar las siguiente tareas
    - Calcular el promedio de entre los primeros 10000 digitos de Pi,
    - Calcular para cada dígito cuantas veces aparece y si hay alguno que aparece más
    - Obtener cuantos primos ocurren y cuanto dá la suma de ellos

  Realizar una implementación secuencial y una implementación con paralelismo de tareas.

- Comparar el tiempo de ejecución de la implementación secuencial y la implementación con paralelismo, usar el comando time para medir el tiempo, por ejemplo,
  time ./ejecutable

Código Base

Script supermenu ejemplo (link) , Digitos de Pi (link)

Bibliografía

[1] Brian W. Kernighan and Rob Pike. 1983. *The UNIX Programming Environment*. Prentice Hall Professional Technical Reference.
[2] Digitos de Pi https://www.angio.net/pi/digits.html