PROYECTO 1

Astrid Gabriela Martínez Castillo.
26 de Febrero 2021

Centro Universitario de Occidente.
División de Ciencias de la Ingeniería.
Sistemas Operativos 1

INTRODUCCION

Este documento describe cada una de las herramientas que se utilizaron para el desarrollo de un programa orientado a una pagina web dedicado a la lectura y la publicación de revistas, destacando tanto aspectos relacionados con el hardware como el software esperando sirva de referencia para especificar la creación de la práctica, así como también los la documentación necesaria.

2. OBJETIVOS

2.1. Objetivos Específicos

- Construcción de semaforos para control de procesos.
- Controlar y producir procesos en un sistema operativo.

2.2. Objetivos Generales

- Implementación de algoritmos de semáforos binarios
- Implementación de algoritmos de semáforos sinarios
- Implementación de interfaz grafica en lenguaje c++

4.1. Requerimientos Mínimos de Software y Hardware

- **Memoria RAM:** 1GB.
- **Espacio en disco:** 124 MB para JRE; 2 MB para Java Update.
- **Procesador Mínimo:** Pentium 2 a 266 MHz
- **Linux**
  Oracle Linux 5.5+1
  Oracle Linux 6.x (32 bits), 6.x (64 bits)2
  Oracle Linux 7.x (64 bits)2 (8u20 y superiores)
  Ubuntu Linux 12.04 LTS, 13.x
  Ubuntu Linux 14.x (8u25 y superiores)
  Ubuntu Linux 15.04 (8u45 y superiores)
  Ubuntu Linux 15.10 (8u65 y superiores)
  Exploradores: Firefox

3. HERRAMIENTAS UTILIZADAS PARA EL DESARROLLO

- Glade para facilitar creación de interfaz grafica
- GDK herramienta para interfaz grafica
- Visual studio code como IDE
- Compilador g++

MARCO TEORICO

Proceso:

Para comenzar me gustaría hacer una explicación acerca de los procesos. Comenzando por la definición de un proceso: **"Un proceso es un programa en ejecución"**. El proceso está formado por el código del programa y el conjunto de datos asociados a la ejecución del programa. El proceso además posee una imagen de memoria, esto es el espacio de memoria en el que está autorizado. La imagen de memoria puede estar referida en memoria virtual o memoria física. Además, en cuanto al ciclo de vida hablaré acerca del que se tiene en cuenta en la planificación a corto plazo (ciclo de vida simple). Esta planificación es en la que se decide el siguiente proceso a ejecutar (FIFO, Round Robin, SJF). Este ciclo de vida posee 4 estados: Listo para ejecutarse, en ejecución, bloqueado y fin.

Ejecución de Procesos: fork()

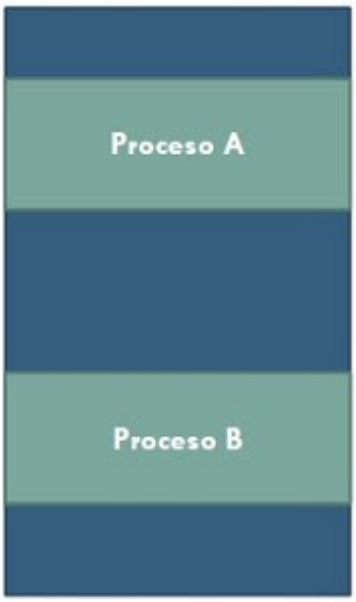
La llamada al sistema que empleamos para crear un nuevo proceso se denomina **fork().** La llamada **fork() crea una copia casi identica del proceso padre** (se copia todo el código) y continúan ejecutándose en paralelo. El proceso **padre recibe de fork() el pid del hijo, mientras que el hijo recibe un 0.** El hijo además hereda recursos del padre (ficheros, abiertos, estado de las variables, etc...) , sin embargo hay otros recursos que no se heredan como por ejemplo las señales pendientes, devuelve -1 en caso de error. La función está declarada tal que así: **size_t fork(void);**

Ejecución de Procesos: Exec()

Esta función **cambia la imagen del proceso actual**. Lo que realiza es **sustituir la imagen de memoria del programa por la de un programa diferente**. Esta función normalmente la invocaremos en un proceso hijo previamente generado por fork(). Existen diferentes funciones para exec, sus declaraciones son:

- **int execl(const char *path, const char *arg, ...);**
- **int execv(const char* path, char* const argv[]);**
- **int execve(const char* path, char* const argv[], char* const envp[]);**
- **int execvp(const char *file, char *const argv[]);**

En **path** debemos pasar la ruta del ejecutable, **file**: Busca el archivo ejecutable en todos los directorios especificados por PATH. Esta función retorna -1 en caso de error, en caso contrario no retorna. No retorna debido a que hemos sustituido la imagen del programa actual por la de un nuevo programa. Debemos pasar los argumentos para el nuevo programa a ejecutarse en ***arg.** Además, se heredan los descriptores de ficheros abiertos y todas las señales pasaran a la acción por defecto.
Cabe mencionar que lo que se debe hacer es dedicar al padre a crear hijos y estos que realizan trabajo por él. Además el padre puede crear mas hijos o esperar a que termine a que termine el hijo. Esta esperar se realiza con la función **wait()**. Esta funciona pasa al padre al estado bloqueado hasta que acabe el hijo.



Sémaforos
Los semáforos son una herramienta de sincronización que ofrece una solución al problema de la sección crítica (porción de código de un programa de computador en la cual se accede a un recurso compartido que no debe ser accedido por mas de un proceso o hilo en ejecución). Un semáforo provee una simple pero útil abstracción para controlar el acceso de múltiples procesos a un recurso común en programación paralela, o entornos multiusuarios. El concepto de semáforo fue inventando por el holandés Esdger W. Dijkstra.

Los semáforos sólo pueden ser manipulados usando las siguientes operaciones (éste es el código con espera activa):

```
Inicia(Semáforo s, Entero v)
{
  s = v;
}
```

En el que se iniciará la variable semáforo s a un valor entero v.

```
P(Semáforo s)
{
  if(s>0)
     s = s-1;
  else
     wait();
}
```
La cual mantendrá en espera activa al regido por el semáforo si éste tiene un valor inferior o igual al nulo.

```
V(Semáforo s)
{
   if(!procesos_bloqueados)
        s = s+1;
   else
        signal();
}
```

Problemas que pueden surgir con mala Sincronización

- DeadLock: Dos o más procesos están esperando por una condición que solo puede ser causada por un proceso que también esta esperando.
- Starvation o Espera Indefinida: Un proceso en la lista de espera de un semáforo de la cual están entrando y saliendo continuamente procesos y listas de espera de semáforo.

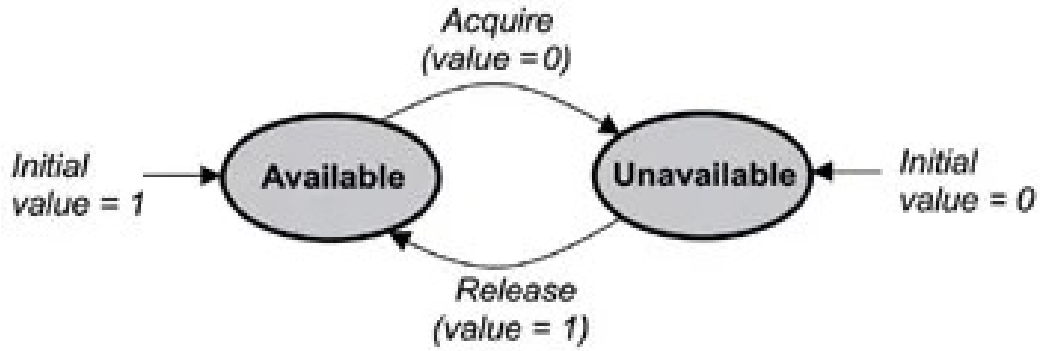
Aplicaciones de los semáforos

Se va a usar la instrucción concurrente COBEGIN-COEND para lanzar de forma concurrente la ejecución de todos los programas que se encuentran dentro de dicha instrucción.

Semáforos binarios

Dijkstra dio en 1968 una solución elegante y sencilla al problema de la exclusión mutua con la introducción del concepto de semáforo binario. Esta técnica permite resolver la mayoría de los problemas de sincronización entre procesos y forma parte del diseño de muchos sistemas operativos y de lenguajes de programación concurrentes.

Un semáforo binario es un indicador de condición (S) que registra si un recurso está disponible o no. Un semáforo binario sólo puede tomar dos valores: O y 1. Si, para un semáforo binario. S=1 entonces el recurso está disponible y la tarea lo puede utilizar; si S=0 el recurso no está disponible y el proceso debe esperar.



Semáforos con múltiples variables
En este caso el semáforo se inicializa con el número total de recursos disponibles (n) y las operaciones de WAIT y SIGNAL se diseñan de modo que se impida el acceso al recurso protegido por el semáforo cuando el valor de éste es menor o igual que cero.

Cada vez que se solicita y obtiene un recurso, el semáforo se decrementa y se incrementa cuando se libera uno de ellos. Si la operación de espera se ejecuta cuando el semáforo tiene un valor menor que uno, el proceso debe quedar en espera de que la ejecución de una operación señal libere alguno de los recursos.

Al igual que en los semáforos binarios, la ejecución de las operaciones son indivisibles, esto es, una vez que se ha empezado la ejecución de uno de estos procedimientos se continuará hasta que la operación se haya completado.