Sistemas Operativos

Entrega 2 – Programación Concurrente

- ### Objetivo

Se debe implementar en *C* un sistema en el cual un proceso *servidor* permita acceder a información requerida por distintos procesos *clientes*. Esta información estará almacenada en distintos archivos de texto desde los cuales el servidor la levantará y se la brindara a los clientes. Se la brindará por un tiempo definido, no permitiendo a otros clientes acceder a dichos datos en este periodo.
Para la comunicación entre los procesos *clientes* y el proceso *servidor* se deberá utilizar *memoria compartida* y *semáforos*.
El proceso *servidor* debe almacenar información de a que proceso le está brindando información, y por cuanto tiempo más la seguirá brindando. Esta información podrá ser consultada en tiempo de ejecución.
La comunicación entre procesos se realizara mediante un(os) nodo(s) de *memoria compartida*, y para manejar la concurrencia sobre esta usaremos *semáforos*.

Para la implementación se deben usar las bibliotecas *ipc* de *C*.

```
<unistd.h>
<sys/types.h>
<sys/wait.h>
<sys/stat.h>
<sys/ipc.h>
<sys/shm.h>
```

Se recomienda estudiar estas bibliotecas. Para facilitar también se dispondrá de una implementación más sencilla del manejo de memoria compartida y semáforos.

```
<shmem.h>
<semaforos.h>
```

Se dispondrá también de `semaforos.c` y `shmem.c` para que se pueda estudiar su contenido.

Se deberán implementar 2 programas:

- ### Servidor:

El mismo será invocado de la siguiente manera:

```
./servidor cantArch cantSeg
```

donde `cantArch` es la cantidad de archivos que levantará el programa y `cantSeg` es el tiempo de barrido de recursos, es decir el máximo tiempo que un cliente tiene acceso a un archivo.
Para realizar el barrido y alguna otra subtarea se debe bifurcar la línea del programa, esta se debe hacer con el método *ipc* `fork()` que fue explicado en clase.
El servidor deberá poder atender a varios clientes al mismo tiempo, al cliente solicitar un archivo, el servidor deberá dar su contenido al cliente mediante un nodo de memoria compartida.
El servidor deberá soportar el ingreso del comando "`estado`" este imprimirá en la salida estándar el estado de los recursos, a quien le fueron asignados, y cuanto tiempo resta.

Por ejemplo si el servidor fue llamado para manejar 5 archivos y 20 segundos., un ejemplo de salida seria:

```
Servidor de Archivos - Sistemas Operativos 2011
FIng- CETP

> estado

Archivo        pid       Tiempo
----------------------------------------
001.txt        2270      4 segundos
002.txt        n/a
003.txt        2289      17 segundos
004.txt        n/a
005.txt        n/a

>
```

· De esta salida se puede ver los requisitos de información que debe llevar el servidor. Se recomienda estudiar bien en cuantos procesos bifurcar el servidor.
· Se deben leer los archivos con `fscanf` u otro método
· Para simplificar se puede suponer que los archivos se llamarán como en el ejemplo es decir *nnn.txt* donde *nnn* es un número entre `1` y `cantArch`.
· El largo máximo del archivo estará dado por la constante `MAX_LARGO_ARCHIVO`:

```
#define MAX_LARGO_ARCHIVO 255
```

- ***Cliente:***

Es un proceso que debe obtener desde el servidor el contenido de un archivo e imprimirlo en pantalla, y quedar esperando hasta que se termine el préstamo, para recibir otra orden por la entrada estándar. Recibirá el nombre por la entrada estándar. En caso de estar el archivo en uso, deberá esperar a que quede libre.
Se pueden lanzar muchas instancias de este programa, cada instancia representa a un cliente distinto.
En caso de un *cliente* solicitar el mismo archivo que tiene otro cliente, el mismo deberá esperar a que se libere, y una vez liberado accederá al archivo, es decir, si hay mas de un cliente, el archivo se le brindará al primero que lo pida, al terminar se liberara el recurso que se asignara al siguiente que lo pidió. Esto se realizara utilizando semáforos.

```
Cliente de Archivos - Sistemas Operativos 2011
FIng- CETP

> 026
ERROR: Archivo no encontrado

> 001
Este es el texto contenido en el archivo

>
```

• Consideraciones Generales

- Los programas deberán atender las señales del sistema, en particular **CTRL + C (*SIGINT*)** para terminar su ejecución.
- Se recomienda leer los siguientes links sobre señales.
  http://www.chuidiang.com/clinux/senhales/senhales.php
  http://es.wikipedia.org/wiki/Se%C3%B1al_%28inform%C3%A1tica%29
  http://www2.dis.ulpgc.es/~itis-ps/signal/index.html

- Se recomienda también familiarizarse con el comando `ipcs` de ***UNIX*** para ver los semáforos y segmentos de memoria compartidos, pues si no son destruidos "sobreviven" a la terminación del programa.
- Se debe entregar un `makefile` que compile todo el proyecto
- Se publica un archivo llamado `ejemploipc.tar.gz`, el mismo contiene un ejemplo con el uso de las bibliotecas y manejo de señales, el mismo puede ser utilizado como base del proyecto.

• Entrega

- El trabajo se realizará en grupos de hasta 3 integrantes
- Se deberá entregar un correo electrónico a <mzabalza@fing.edu.uy>
- El correo deberá tener el siguiente formato:

- **Subject: so2011e1-1234567-1234568-1234569**

*(Donde **1234567, 1234568, 1234569** son las cédulas de los integrantes del grupo sin dígito de verificación)*

- **Body:**
  1234567 Nombre1 APELLIDO1
  1234568 Nombre2 APELLIDO2
  1234569 Nombre3 APELLIDO3

(Corresponden a los datos de cada integrante del grupo)

- **El correo electrónico deberá tener adjunto un archivo llamado `tarea2.tar.gz`. (deberá contener todos los archivos necesarios para la solución, así como un `makefile` para compilar el proyecto, y opcionalmente un archivo `leame.txt` con aclaraciones sobre la solución).**
- **El correo electrónico deberá ser de texto sin formato.**

- La entrega deberá realizarse antes del día Viernes 11 de Noviembre a las 23:59.
(L*os trabajos que no respeten el formato de entrega no serán considerados*)