Proyectos de Taller

Sistemas Operativos

2018

Entrega: para realizar la entrega pueden seguir los pasos que indica el archivo DiffPatch.txt, que se encuentra en moodle, en la subcarpeta de xv6 (sección 3 - taller).

- **Proyecto 0**: Xv6 hace que un proceso use una CPU en ráfagas de duración de 1 tick. Modificar el kernel de xv6 para que permita que los procesos puedan usar la cpu en ráfagas (timeslices o QUANTUM) de más 1 tick.

  Para poder probar que un proceso usa una CPU por mas de un tick, imprimir un cartel cada vez que un proceso abandona la CPU por vencimiento del quantum y hacer que algún proceso de usuario haga uso intensivo de CPU.

  Fecha límite de entrega: Miércoles 4 de Abril.

  Modalidad de entrega: El desarrollo del código y la entrega se deben realizar de manera individual o grupal. Para ello tienen habilitada la opción de subida de archivos en moodle. Deberán subir el patch generado o el código comprimido.

- **Proyecto 1**: Extender a xv6 creando una llamada al sistema `int procstat(void)`, la cual deberá listar los procesos existentes en el sistema y su estado.

  Consejos: la implementación de la llamada al sistema debería hacerse en el módulo `sysproc.c`.

  Ayuda: al definir una nueva llamada al sistema, ésta deberá incluirse en la tabla de llamadas al sistema del kernel (ver `syscall.c` y `syscall.h`). También se deberá definir la nueva función (`int procstat(void)`) en espacio de usuario (ver los archivos `usys.S` y `user.h`).

Con esta nueva llamada al sistema, modificar el programa de usuario `forktest.c` el cual deberá realizar la llamada al sistema `procstat()` luego del quinto fork().

Se deberá ejecutar el comando `forktest` en el shell de xv6 para verificar que la llamada al sistema se realiza.

Fecha límite de entrega: Miércoles 11 de Abril.

Modalidad de entrega: La entrega la deberá realizar uno de los integrantes del grupo. Para ello tienen habilitada la opción de subida de archivos en moodle. Deberán subir el patch generado o el código comprimido. Además, deberán indicar quienes son los integrantes del grupo.

- **Proyecto 2**: Implementar un planificador multinivel con retroalimentación (4 niveles). Incluir una nueva llamada al sistema `void setpriority(int priority)` que cambie la prioridad de un proceso de usuario. En la creación cada proceso de usuario deberá tener una prioridad inicial 0 (la más alta).

  Además, implementar una política de envejecimiento (*aging*), para garantizar que todos los procesos sean eventualmente planificados. Realice un programa de usuario que permita verificar el funcionamiento del la política definida.

  Fecha límite de entrega: Jueves 19 de abril.

  Modalidad de entrega: La entrega la deberá realizar uno de los integrantes del grupo. Para ello tienen habilitada la opción de subida de archivos en moodle. Deberán subir el patch generado o el código comprimido. Además, deberán indicar quienes son los integrantes del grupo.

- **Proyecto 3**: La única primitiva de sincronización que posee xv6 son los *spinlocks*. En este proyecto deberán agregar semáforos n-arios. Para eso deberán realizar las siguientes acciones:

  Implementar en xv6 un módulo semaphore que defina las siguientes primitivas de sincronización de semáforos

  - `int semget(int semid, int initvalue)`
    Descripción: Crea u obtiene un descriptor de un semáforo.
    Parámetros:
    - `semid` identificador del semáforo, o -1 si se desea crear uno nuevo.

- `initvalue` valor inicial, solo utilizado cuando `sem_id` es -1.

Retorno: Identificador del semáforo obtenido o creado, caso contrario retornar un valor negativo indicando error. Los posibles valores de error pueden ser:

- -1: `semid`>=0 y el semáforo `semid` no está en uso.
- -2: `semid`=-1 y el proceso ya obtuvo el número máximo de semáforos.
- -3: `semid`=-1 y no hay más semáforos disponibles en el sistema.

- `int semfree(int semid)`
  Descripción: Libera el semáforo.
  Parámetros: `semid` es el identificador (descriptor) del semáforo.
  Retorno: -1 en caso de error (semáforo no obtenido por el proceso). Cero en otro caso.
- `int semdown(int semid)`
  Descripción: Decrementa una unidad el valor del semáforo y bloquea al proceso invocante en caso que el valor del semáforo sea <0.
  Parámetros: `semid` es el identificador (descriptor) del semáforo.
  Retorno: -1 en caso de error (semáforo no obtenido previamente por el proceso), cero en otro caso.
- `int semup(int semid)`
  Descripción: Incrementa una unidad el valor del semáforo y eventualmente despierta algún proceso en la cola de espera por ese semáforo.
  Parámetros: `semid` es el identificador (descriptor) del semáforo.
  Retorno: -1 en caso de error (semáforo no obtenido previamente por el proceso), cero en otro caso.

1. Detalles de implementación:
   a) El sistema deberá contener hasta `MAXSEM` semáforos.
   b) Cada proceso podrá usar hasta `MAXPROCSEM` (`MAXPROCSEM` $\leq$ `MAXSEM`) semáforos.
   c) Cada semáforo debería mantener un contador de referencias para que se destruya efectivamente cuando ser realicen tantos `semfree()` como `semget()`.
   d) Cuando un proceso finalice su ejecución, deberá eliminar todos los semáforos que le pertenecen no liberados explícitamente.

   e) Pude incluir nuevos códigos de errores en las funciones, si lo considera necesario.

2. Definir las llamadas al sistema en espacio de usuario correspondientes para cada operación.
3. Escribir un programa de usuario que implemente el problema del productor-consumidor usando semáforos.

Fecha límite de entrega: Jueves 10 de Mayo.

Modalidad de entrega: La entrega la deberá realizar uno de los integrantes del grupo. Para ello tienen habilitada la opción de subida de archivos en moodle. Deberán subir el patch generado o el código comprimido. Además, deberán indicar quienes son los integrantes del grupo.