Facultad de Ciencias Exactas, Ingeniería y Agrimensura
Escuela de Ciencias Exactas y Naturales
Departamento de Ciencias de la Computación
R-322 — Sistemas Operativos I

Nombre y Apellido:

Examen Parcial

Los mecanismos de sincronización sin espera activa son habitualmente implementados en el núcleo de los sistemas operativos, estos tienen mayor facilidad y capacidad para cambiar el estado de los procesos. Sin embargo, las llamadas de sistemas toman un tiempo considerable debido a la interrupción de software y posterior cambio de contexto a ejecución del núcleo.

Se puede mejorar el rendimiento si se invoca al núcleo únicamente en caso de competencia entre procesos. Si solo uno desea entrar a una región crítica, puede resolverse directamente en el espacio de usuario con ayuda de las instrucciones atómicas como `get&add`, `test&set (TAS)`, `swap`, `compare&swap (CAS)`.

El objetivo de la interfaz FUTEX (Fast User space muTEXes) de Linux fue facilitar la implementación de técnicas (que usan sincronización por hardware) para primitivas bloqueantes eficientes sin necesidad de programarlas en el núcleo.

Las librerías Native POSIX Thread Libraries (NPTL, implementan el estándar POSIX Threads en Linux) y las de semáforos POSIX (ambas incluidas en las GLibc) usan FUTEX con varias optimizaciones a nivel de usuario.

La base de la interfaz es una variable entera llamada futex. Los procesos deben especificar la dirección de dicha variable, puede ser cualquiera que sea válida y accesible, no hay requisitos especiales.

Ej. 1. (40 puntos) Se pide implementar una librería de semáforos usando futex.

Para hacerlo supongamos que cuentan con las siguientes 2 funciones disponibles en el archivo `linux/futex.h`[1]:

`futex_wait(futex, value)` : Si el valor de `futex` es igual al valor `value` suspende al proceso y lo agrega a la cola de bloqueados del futex. Retorna 0 si fue desbloqueado por el `futex_wake`, -1 si no pudo bloquear o hubo error.

`futex_wake(futex, processes)`: Desbloquea a uno o más procesos según lo indicado por `processes` y retorna el número de procesos desbloqueados.

Puede hacer uso de las funciones atómicas que necesite del estilo[2]:

```
atomic_add_fetch (type *ptr, type val);
atomic_sub_fetch (type *ptr, type val); // substracción
atomic_and_fetch (type *ptr, type val);
atomic_xor_fetch (type *ptr, type val);
atomic_or_fetch (type *ptr, type val);
```

donde `type` es un tipo válido en `C` y esas funciones realizan las operaciones sugeridas por el nombre y retornan el resultado de la operación. Esto es:

```
{ *ptr op= val; return *ptr; }
```

[1] Lo que sigue a continuación no es exactamente así, sino que se ha simplificado para los propósitos del ejercicio.
[2] Ayuda: no todas son necesarias para resolver el ejercicio.

Ej. 2. (20 puntos) Utilice la librería de semáforos implementada anteriormente para sincronizar 4 threads que desean actualizar un contador de forma concurrente.

Ej. 3. (10 puntos) ¿Qué son y cómo se relacionan entre sí los conceptos de región crítica, *race condition* y exclusión mutua?

Ej. 4. (15 puntos) Dado un proceso con un archivo abierto y su hijo creado a través de `fork()`; si ambos quisieran escribir en dicho archivo ¿qué problemas podrían surgir? ¿Cómo solucionaría dichos problemas? ¿Sería posible a partir de los mecanismos usados para solucionar dichos problemas que se genere un deadlock en este caso en particular? Justifcar. ¿Y un livelock? Justificar.

Ej. 5. (15 puntos) Completar las líneas en blanco de forma tal que el programa imprima siempre `"Hola Mundo"`. Utilizar variables de condición y asumir que el sistema se comporta con semántica Mesa.

```
int ben = 0;
----------------------;
----------------------;

void *helper(void *arg) {
    ----------------------;
    ben += 1;
    ----------------------;
    ----------------------;
    pthread_exit(NULL);
}

void main() {
    pthread_t thread;
    pthread_create(&thread, NULL, &helper, NULL);
    sched_yield();
    ----------------------;
    ----------------------;
    ----------------------;
    if (ben == 1)
        printf("Hola Mundo\n");
    else
        printf("Hasta luego\n");
    ----------------------;
}
```