Laboratorio Sistemas Operativos 2024

Comision 26

Juan Román Brugnoni

Matias Diomedi

Actividad 1

Estrategia que se utilizó para la resolución del problema de los atletas:

Primero, identificamos que el problema en cuestión era similar al clásico problema de lectores y escritores, con prioridad para los lectores. Para resolverlo, utilizamos dos semáforos: uno contador y otro binario. El semáforo "semCorredores" cuenta la cantidad de corredores que están utilizando las instalaciones, mientras que el semáforo "semJabalinaOMartillo" controla el acceso de los atletas de lanzamiento de jabalina o martillo. Además, implementamos un mutex para gestionar las secciones de entrada y salida de los diferentes atletas.

El semáforo "semCorredores" se inicializa en 0, mientras que "semJabalinaOMartillo" se inicializa en 1. La elección de estos valores se debe a que "semCorredores" cuenta cuántos corredores están dentro de las instalaciones. Al utilizar try-wait, podemos identificar si un corredor es el primero o el último en entrar o salir, lo que nos permite bloquear o liberar el acceso a los atletas de lanzamiento. Por otro lado, decidimos que "semJabalinaOMartillo" inicie en 1 porque solo puede haber un atleta lanzador (ya sea de martillo o de jabalina) en las instalaciones al mismo tiempo.

Instrucciones para compilar: utilizar gcc -o atletas atletas.c -lpthread

Actividad 2

En este problema particular, los semáforos permiten un control más directo y declarativo sobre el acceso a las instalaciones. Sin embargo, esto puede complicar la solución, ya que requiere un mayor control por parte del programador sobre el flujo de ejecución, así como sobre las condiciones de carrera y la inanición.

Por otro lado, la utilización de colas de mensajes simplifica el manejo de las condiciones de carrera, ya que se puede acceder a la cola y leer de acuerdo con el tipo de mensaje que se necesita, estableciendo un orden basado en el envío de mensajes. Esta herramienta garantiza la exclusión mutua. Además, la escalabilidad se simplifica: si se desea leer otro tipo de mensaje, basta con agregar más tipos. Sin embargo, esto puede repercutir en la heterogeneidad de la cola de mensajes, ya que tener muchos tipos puede complicar el código.

En cuanto al rendimiento, la latencia entre el envío y la recepción de mensajes puede ser mayor que con hilos y semáforos. Estos últimos maximizan la concurrencia, lo que puede resultar en un mejor rendimiento en situaciones donde la velocidad de acceso a los recursos compartidos es crucial.

Instrucciones para compilar y ejecutar
Compilación:
gcc -o atletaIniciador atletaIniciador.c -lpthread
gcc -o Corredor Corredor.c -lpthread
gcc -o Jabalina Jabalina.c -lpthread
gcc -o Martillo.c -lpthread

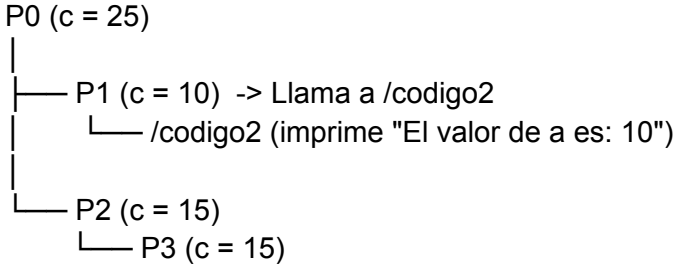
Para ejecutar:
.\atletaIniciador

1. **Proceso Inicial (P0)**:
    - c inicial es 5.
    - Se llama a fork(), creando Proceso 1 (P1) como hijo (copia del proceso inicial
2. **Proceso 1 (P1)**: (Hijo del Proceso Inicial)
    - pid == 0, por lo que entra en el bloque if.
    - c se convierte en 10 (5 + 5).
    - Llama a execl() para ejecutar /codigo2 con c (10) como argumento.
    - codigo2 imprime "El valor de a es: 10".
3. **Proceso Inicial (P0)** continúa:
    - Llama a fork() nuevamente, creando Proceso 2 (P2).
    - c se convierte en 15 (5 + 10).
    - En Proceso 0:
        - pid de fork() es mayor que 0, por lo que entra en el bloque if.
        - c se convierte en 25 (15 + 10).
4. **Proceso 2 (P2)**: (Hijo de P0)
    - c es 15 (hereda la variable de P0).
    - fork() se ejecuta en todos los procesos, lo que crea Proceso 3 (P3).
5. **Proceso 3 (P3)**: (Hijo de P2)
    - c es 15 (hereda de P2).
    - Al ejecutar fork(), también tendrá el mismo valor de c.

Impresiones finales:

- P1 imprime 10 antes de finalizar.
- P0 imprime 25.
- P2 y P3 imprimirán 15.

Jerarquía de Procesos



Si quisiera que todos los procesos “vean” el mismo valor de C usaría segmento de memoria compartida pero generaría condiciones de carrera y la ejecución e impresión del valor depende del orden en el que se atienden los procesos.