Sistemas Operativos
1° Parcial 1C2024 - TT - Resolución

Aclaración: La mayoría de las preguntas o ejercicios no tienen una única solución. Por lo tanto, si una solución particular no es similar a la expuesta aquí, no significa necesariamente que la misma sea incorrecta. Ante cualquier duda, consultar con el/la docente del curso.

Teoría

1) El modo kernel puede ejecutar tanto las instrucciones privilegiadas como las no privilegiadas. El modo usuario solamente puede ejecutar instrucciones no privilegiadas.
2) Usar procesos ofrece un aislamiento que permite evitar la propagación de errores (intencionales o no) entre ellos, mientras que los hilos al encontrarse altamente acoplados (compartiendo varios recursos) pueden verse afectados. Además, al no compartir recursos entre ellos por default, los procesos proveen mayor seguridad. Una de las desventajas de utilizar procesos es que su creación es más lenta, ya que crear un KLT requiere de estructuras más livianas que crear procesos. Utilizar hilos de usuario permitiría realizar menos cambios de modo, ya que se gestionan en modo de usuario.
3) La planificación sin desalojo implica que el proceso libera la cpu solamente cuando se bloquea o finaliza su ejecución. La planificación con desalojo agrega la posibilidad de expulsar intempestivamente al proceso de la cpu en la ocurrencia de interrupciones o syscalls, cuando otro proceso tiene más prioridad según indique el planificador. Sería necesario usar planificadores con desalojo para evitar monopolizar la cpu.
4) Verdadero o falso:
    a) Entendiendo por competencia al esquema bajo el cual el sistema operativo es el único quien tiene la responsabilidad de administrar los recursos, sería FALSO, por otra parte, si se entiende por competencia a la utilización de los recursos por parte de los procesos de forma concurrente, es VERDADERA sí y sólo sí al menos uno de los procesos/hilos involucrados fuese a modificar los recursos.
    b) Falso. Si bien es cierto que las soluciones de software incurren en espera activa ya que su implementación requiere de utilizar bucles al no poder usar bloqueo, su performance dependerá de varios factores. Una sección crítica con poca probabilidad de concurrencia, donde el código dentro de la misma se ejecuta rápidamente y teniendo en cuenta un sistema con varios núcleos, esta alternativa podría ser más efectiva que la utilización de soluciones con bloqueo.
5) Considera mirar las siguientes métricas:
    a) Listar los procesos y ver su estado. Si los mismos no progresan y se encuentran en estado BLOQUEADO, sería un indicio de que podrían estar interbloqueados mientras que si todos los procesos se encuentran ejecutando sería un indicio de un posible livelock.
    b) Listar el consumo de la cpu de la computadora y de los procesos involucrados. En caso de que el mismo tuviera valores que indicaran un consumo frecuente, y

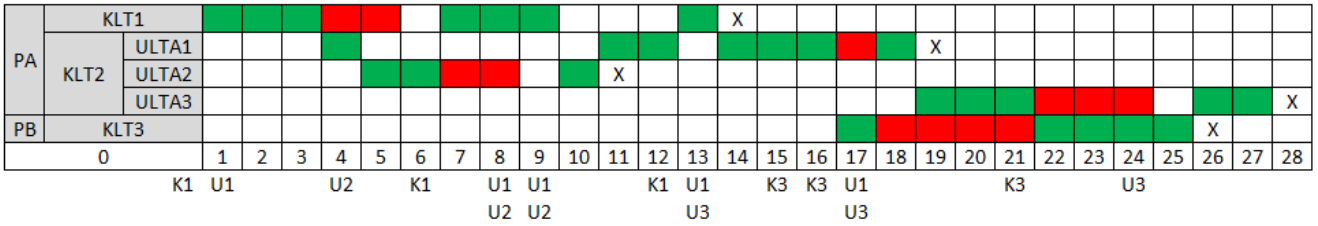
si observáramos que su estado cambia con frecuencia (READY/RUNNING/BLOQUEADO) podríamos estar ante la presencia de un livelock.

Práctica

1)

a) Gantt

- Ráfagas de CPU representadas en verde.
- Ráfagas de IO representadas en rojo.



b) Interrupciones

- t = 5 -> Fin de IO para KLT1.
- t = 8 -> Fin de IO para ULTA2.
- t = 9 -> Fin de quantum para KLT1.
- t = 12 -> Fin de quantum para KLT2.
- t = 17 -> Fin de IO para ULTA1.
- t = 20 -> Fin de quantum KLT2.
- t = 21 -> Fin de IO para KLT3.
- t = 24 -> Fin de quantum para KLT3 y fin de IO para ULTA3.

c) No cambiaría porque cada vez que uno de los hilos ULT pertenecientes al hilo KLT2 solicita realizar una operación de entrada y salida, justo sonaría el clock indicando el fin de quantum. Por lo tanto, si otro ULT quisiera ejecutar, no podría, ya que le toca a otro hilo KLT u otro proceso.

2)
```
sem_artefactos_limite = 50;
sem_seccion_limite[5] = {10, 10, 10, 10, 10};
sem_encargo_disponible[5] = {0, 0, 0, 0, 0};
mutex_seccion[5] = {1, 1, 1, 1, 1};
sem_encargo_entregado = 0;
mutex_cola_sistema = 1;
```

Cliente (N instancias)

```
while(1) {
    artefacto_roto = obtener_artefacto_roto(lista_artefactos_del_cliente);
    encargo = llevar_a_reparar(artefacto_roto);
    wait(sem_artefactos_limite);
    wait(mutex_cola_sistema);
    entregar_encargo(encargo, cola_sistema);
    signal(mutex_cola_sistema);
    signal(sem_encargo_entregado);
}
```

Administrador (1 instancia)

```
while(1) {
    wait(sem_encargo_entregado);
    wait(mutex_cola_sistema);
    encargo = retirar_encargo(cola_sistema);
    signal(mutex_cola_sistema);
    signal(sem_artefactos_limite);
    id_seccion = obtener_seccion_de(encargo);
    wait(sem_seccion_limite[id_seccion]);
    wait(mutex_seccion[id_seccion]);
    depositar(encargo, cola_encargos[id_seccion]);
    signal(mutex_seccion[id_seccion]);
    signal(sem_encargo_disponible[id_seccion]);
}
```

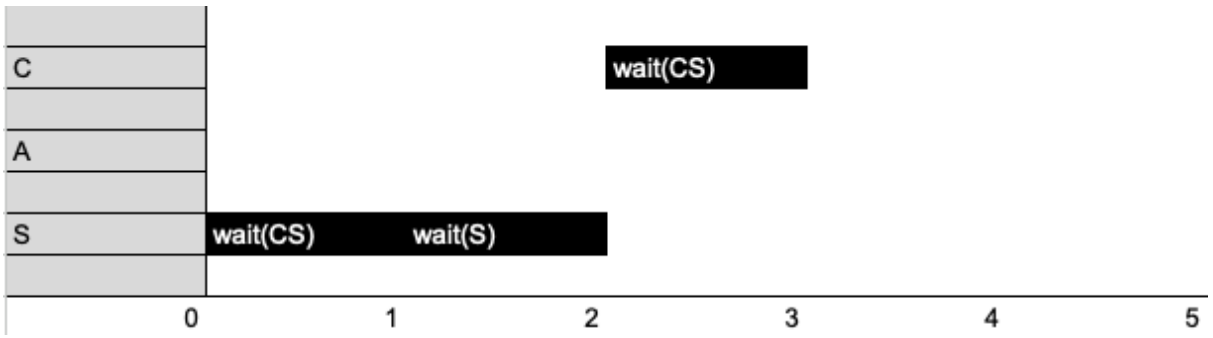
Técnico (5 instancias)

```
while(1) {
    id_seccion = seccion_perteneciente(id_tecnico);
    wait(sem_encargo_disponible[id_seccion]);
    wait(mutex_seccion[id_seccion]);
    encargo = tomar_encargo(cola_encargos[id_seccion]);
    signal(mutex_seccion[id_seccion]);
    signal(sem_seccion_limite[id_seccion]);
    reparar_artefacto(encargo);
}
```

3)

a)

Es posible que los procesos queden en Deadlock. Se pueden encontrar varias secuencias, pero la más sencilla de identificar sería que comience el proceso S y luego el proceso C.



Si bien en el GANTT no se observa al proceso C obteniendo directamente el recurso S, al observar el código se puede ver que el proceso C retiene lógicamente el recurso S, ya que es ese proceso quien habilita al proceso S a ejecutar.

b) Se podría solucionar el problema invirtiendo el orden de los wait iniciales tanto en el proceso S como en el proceso C. Primero deberían realizar wait(C) y wait(S) respectivamente, para luego realizar el wait(CS). Conceptualmente los primeros semáforos controlan el orden y el segundo el acceso efectivo, por lo cuál si los invertimos podemos caer en Deadlock y poniéndolos de forma correcta lo estamos evitando.