# Sistemas Operativos
# 1° Parcial 1C2017 - TM - 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 profesor del curso.*

**TEORIA:**

1) Porque utilizan instrucciones que requieren que el bit de modo esté seteado en modo kernel, ya que acceden a hardware o bien a secciones de memoria que un proceso no puede (ni debe) acceder en modo usuario. Si la misma se mantuviera en modo usuario, se producirá un error (en forma de interrupción) a la hora de ejecutar alguna instrucción privilegiada

2) Puede estar suspendido listo, lo cual significa que estuvo esperando un evento y mientras eso ocurría fue suspendido. Luego dicho evento se produjo, y por lo tanto el proceso quedó en estado suspendido/listo.

3) En principio alguna variante de colas multinivel. Idealmente, los procesos financieros deben estar en una cola de mayor prioridad, y las atenciones virtuales en una cola de menor. La primera cola probablemente utilice un FIFO, la segunda podría usar RR, para poder distribuir la atención de los clientes. No hay transiciones entre las colas, ya que la idea es separarlos. Dos criterios que podrían usarse para evaluar el funcionamiento son: Deadlines (sobre todo para la cola de mayor prioridad) y quizás tasa de procesamiento (para todos, cuantas más consultas se contesten, mejor será el algoritmo).

4) El semáforo bloqueante decrementa la variable semáforo con wait, y si el valor es < a cero bloquea al proceso. Lleva una cola de procesos bloqueados. La función signal la incrementa, y si corresponde, desbloquea un proceso de la cola. El sistema operativo puede implementar mutua exclusión, sobre el semáforo en sí, deshabilitando las interrupciones o bien con instrucciones de hw (por ej, test and set)

5) La estrategia consiste en no otorgar límite alguno en el inicio y solicitud de recursos de parte de los procesos. Agregado a eso, se ejecuta periódicamente un algoritmo de detección de deadlock sobre el estado actual, y en caso de encontrar alguno se intenta eliminar el mismo a través de alguna estrategia de recupero (finalización de proceso, expropiación de recurso, etc).
Desventajas podrían ser el overhead que genera correr el algoritmo cada cierto periodo de tiempo, y también el hecho de que los procesos podrían verse finalizados abruptamente si fueron parte de un deadlock.

## PRACTICA

## Ejercicio 1a:

a) El sistema tiene **2 procesadores.**
   CPU1: Ejecutan P1 y P2
   CPU2: Ejecutan P2 y P4
   En el instante T=1 podemos observar que ejecutan A y F

b) Ambos procesadores usan el mismo algoritmo **FIFO**. Dado que todos sus hilos ejecutan hasta tener un evento (I/O).
   Se puede observar:
   CPU1: T=3 se bloquea P1 el hilo A y comienza a ejecutar P2. En T=7 se bloque a P2 el hilo C y vuelve a ejecutar P1.
   CPU2: T=1 se bloquea P4 el hilo F y comienza a ejecutar P4 hilo G. En T=4 se bloque a P4 el hilo G y vuelve a ejecutar P4 con el hilo F y una vez que finaliza, comienza a ejecutar P3 con el hilo D.

c) **P1: hilos A y B usuarios**, dado que cuando uno se bloquea,bloquea el proceso ejemplo en T= 3.
   **P2: hilo C podria ser usuario o de Kernel**
   **P3: Hilos D y E** de usuario, dado que cuando uno se bloquea,bloquea el proceso ejemplo en T= 8.
   **P4: Hilos F y G** de Kernel, dado que cuando un hilo se bloque puede seguir el otro usando la CPU. Ejemplo T=2.

d) **P1: en la CPU1 utiliza FIFO**, porque ejecuta hasta que llegue un evento . ejemplo Hilo A hasta T= 3 y Hilo B T=8 pero comienza su ejecución en T =11 porque estaba usando la I/O el hilo C de P2.
   **P3: en la CPU2 utiliza FIFO**, porque ejecuta hasta que llegue un evento . ejemplo Hilo D hasta T= 8 y Hilo E T=16 .
   **P3: en la CP2 tambien podria ser SJF,** dado que siempre ejecuta primero el hilo más corto**.En este caso las I/O podría ser manejadas por S.O o por Biblioteca**

e) **En P1 las I/O son manejadas por la biblioteca,** dado que siempre vuelve el hilo contrario, ejemplo en T=8
   **En P3 si es FIFO las I/O son manejada por el S.O** dado que siempre vuelve el mismo hilo. El S.O no conoce a los hilos de usuario. Ejemplo T = 12
   **En P3 si es SJF está contestado en el ítems D.**

f) **El sistema tiene dos I/O** una en cada CPU lo puedo observar en T=5 que ejecutan al mismo tiempo I/O de P4 y I/O de de P1 y otra vez en T=9 P2 y P3.

g) **Se aplica la Regla en T =3** entre P3 que llega nuevo y Hilo F que vuelve de I/O

## Ejercicio 2:

**mut_transacciones = 1; cant_transacciones = 0; roles_cargados = 0; usuario_cargado = 0; pedido_usuario = 0; pedido_roles = 0; usuario_completo = 0;**

**No se necesitan mutex para usuarios ni roles, porque el orden en la sincronización hace imposible que se pisen.**

```
int main() {
  while(true) {
    crear_hilo(f_roles);
    crear_hilo(f_usuarios);
    wait(usuario_completo);
    if(usuario.rol == rol_autorizado) {
      transaccion = crear_transaccion();
      wait(mut_transacciones);
      agregar(transacciones,
transaccion);
      signal(mut_transacciones);
      signal(cant_transacciones);
    }
    roles = null;
    usuario = null;
  }
}
function void f_roles() {
  signal(pedido_roles);
  wait(roles_cargados);
  rol_autorizado = buscar(roles);
}
function void f_usuarios() {
  signal(pedido_usuario);
  wait(roles_cargados);
  wait(usuario_cargado);
  usuario.rol = cargar_rol(roles);
  signal(usuario_completo);
}
```

**Servicio de Usuarios:**

```
while(true) {
  wait(pedido_usuario);
  usuario = obtener_de_la_DB('usuario');
  signal(usuario_cargado);
}
```

**Servicio de Roles:**

```
while(true) {
  wait(pedido_roles);
  roles = obtener_de_la_DB('roles');
  signal(roles_cargados); // Para f_usuario
  signal(roles_cargados); // Para f_roles
}
```

**Servicio de Transacciones:**

```
while(true) {
  wait(cant_transacciones);
  wait(mut_transacciones);
  tx = obtener(transacciones);
  signal(mut_transacciones);
  procesar(tx);
}
```

**Ejercicio 3**

1)
Primero calculemos la matriz de Necesidad

| | R1 | R2 | R3 | R4 |
|---|---|---|---|---|
| P1 | 1 -> 0 | 2 -> 1 | 0 | 0 |
| P2 | 0 | 0 | 1 | 4 |
| P3 | 1 | 2 | 4 | 0 |
| P4 | 0 | 0 | 1 | 0 |

P1 pide 1 1 0 0 , como es asignado, en principio esos recursos deben estar disponibles. También sabemos que las peticiones son válidas por lo que debe haber como mín 2, 2, 4, 4 (que son los máx que piden los procesos).
Como lo asignado es (1, 0 , 3,  2) => actualmente disponible tiene que haber mínimo 1, 2, 1, 2

Ahora veamos si con ese vector, al asignar 1 1 0 0 el sistema queda en estado seguro:

Inicial: 1 2 1 2
Asignó petición P1 -> 0 1 1 2

Término de atender a P1 -> 2 2 1 3
Atiendo a P4 -> 2 2 2 3
No puedo atender ni a P2 ni a P3, como mínimo, necesitaría agregar una instancia de R4 -> 2 2 2 4
Atiendo P2 -> 2 2 4 4
Atiendo a P3 -> 2 2 4 5 -> este sería el mínimo vector de recursos totales para que se cumpla la condición

2) Partiendo de recursos totales = 2 2 4 5 => rec disponibles = 1 2 1 3
P3 pide 1 R1  -> 0 2 1 3
P4 -> 0 2 2 3
No puedo atender a ninguno más, dejaría el sistema en estado inseguro, no se asigna