# Examen Sistemas Operativos Avanzados

## VIERNES 25 OCTUBRE 2019

## Nombre y FIRMA:

### Preguntas (6 PUNTO) (Usar Otro folio para responder)-mid-09)

1. ¿Qué abstracciones emplea el SO para virtualizar la CPU y la memoria?
2. ¿Por qué es necesario tener un *stack* de usuario y un *stack* de kernel diferentes por proceso? ¿Qué ocurre si se intenta acceder directamente al contenido del *stack* de *kernel* desde el código del usuario?
3. ¿Cómo regresa el procesador el estado “*usuario*” después de ejecutar una llamada al sistema?
4. ¿Por que CFS usa un *red-black-tree*?
5. Corregir el problema de la siguiente implementación de planificación basada en *stride*:

current = remove\_min(queue); // pick client with minimum pass

schedule(current); // use resource for quantum

current->pass -= current->stride; // compute next pass using stride

insert(queue, current); // put back into the queue

1. ¿Qué significa que SQMS “no escala”?

### Problema 1 (2 puntos)

Los siguientes diagramas temporales el tiempo de llegada a la cola de planificación y el tiempo en que el planificador les cede el uso de la CPU (única). ¿Cuál es el tiempo de *turn-around* del *job* A? Suponer que A llega al comienzo del tiempo. Empieza a ejecutarse cuando “\*” aparece en la segunda línea. Finaliza su ejecución cuando “x” aparece en la segunda línea.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Ejemplo  ABABA  \* x | | | A llega en t=0, empieza a ejecutarse inmediatamente y finaliza en t=5. B se ejecuta entre t=1-2 y t=3-4 | |
|  | | | | |
| ABABABABAB  \* x | | BBABABABABA  \* x | | ABCBABCABCA  \* x |
| AAAAABBBBB  \* x | | BCABBBBBABA  \* x | | BBBBBBAAAAA  \* x |
| Repetir el mismo proceso, pero para el tiempo de respuesta (del Job A) | | | | |
| ABABABABAB  \* x | BBBBBBAAAAAA  \* x | ABCABCCCBA  \* x | BBABABABABA  \* x | AAAAABBBBBA  \* X |

### Problema 2 (2 PUNTOS) (spring-16)

Algunos pasos de la gestión con LDE son llevados a cabo por software del sistema operativo (SO), gestionados por el Hardware (HW), y algunos en el propio programa de usuario (USR). A continuación, se listan los pasos llevados a cabo para ejecutar un programa que finaliza justo después de hacer una llamada al sistema. ¿Cuál es el responsable en cada caso?:

* Crear una entrada en la lista de procesos (OS) (HW) (USR)
* Reservar la memoria para ejecutar el programa (OS) (HW) (USR)
* Cargar el programa en memoria (OS) (HW) (USR)
* Preparar el *stack* de usuario con *argv* (OS) (HW) (USR)
* Ejecutar la instrucción de retorno de *trap* (OS) (HW) (USR)
* Conmutar a modo usuario (OS) (HW) (USR)
* Colocar el PC apuntando a *main()* (OS) (HW) (USR)
* Empezar a ejecutar *main()* (OS) (HW) (USR)
* Llamar a la llamada al sistema (OS) (HW) (USR)
* Ejecutar la instrucción *trap* (OS) (HW) (USR)
* Salvar los registros en el *stack* de *kernel* (OS) (HW) (USR)
* Conmutar a modo *kernel* (OS) (HW) (USR)
* Manejar la *trap* (OS) (HW) (USR)
* Realizar el trabajo de la *syscall* (OS) (HW) (USR)
* Ejecutar la instrucción de retorno de *trap* (OS) (HW) (USR)
* Restaurar los registros desde el *stack* de *kernel* (OS) (HW) (USR)
* Conmutar a modo usuario (OS) (HW) (USR)
* Colocar el pc apuntando a la instrucción posterior al *trap* (OS) (HW) (USR)
* Llamar a la llamada al sistema *exit()* (OS) (HW) (USR)

# PRácticas

### Previo

Crear un nuevo repositorio personal, llamado Examen1, en gitlab.com. El repositorio deberá adr acceso *Developer* al profesor. Deben existir 3 directorios P0, P1 y P2, incluyendo cada uno de ellos las implementaciones realizadas. “*Commitear*” todos los ficheros de partida con id:

### Lab 0 (3.5 puntoS)

Extender la implementación para que permita hacer la ordenación en orden inverso si se agrega el *switch* “–r”. Introducir los cambios en el repositorio en un solo “*commit*” con id:

### Lab 1 (3.5 puntoS)

Extender la implementación de la llamada al sistema, de modo que si se le pasa el argumento “0” siga devolviendo todos los procesos y si se le pasa el argumento “1” devuelva sólo los bloqueados. Introducir los cambios en el repositorio en un solo “*commit*” con id:

### Lab 2 (3.0 puntoS)

Extender la implementación, de modo que al hacer *fork()* la prioridad del hijo sea la del padre. Introducir los cambios de en el repositorio en un solo “*commit*” con id: