SISTEMAS OPERATIVOS AVANZADOS (TEORÍA)[1]

LUNES 17 ENERO 2022

PROCESOS (3 PUNTOS)

a) ¿Por qué un proceso ha de tener un *stack de kernel* independiente del *stack* de usuario? ¿En qué condiciones el código de usuario del proceso puede acceder al *stack* de *kernel*?
b) ¿Qué utilidad tiene que el *timer* sea programado con un lapso más pequeño que el *quantum* de planificación de la cola más prioritaria (i.e., la que tiene el quantum más pequeño) en un planificador MLFQ?
c) ¿Qué ventaja tiene un planificador basado en *stride* sobre uno basado en lotería?
d) ¿Por que no es siempre posible mantener la afinidad de cache en un planificador multicola en un sistema multiprocesador (MQMS)?
e) En el planificador CFS, ¿Cuál es la interpretación del parámetro *niceness*?
f) ¿Por qué, en el contexto de la planificación de procesos, un planificador tipo SJF (*Shortest Job First*) o STCF (Shortest Time-to-completion first) no es implementable? ¿Cuál es su utilidad?

MEMORIA (3 PUNTOS)

a) Suponer un *kernel* con mitigación *Meltdown* (es decir, la tabla de paginas del proceso no incluye una copa de la tabla de páginas del *kernel*). ¿Qué implicaciones tiene a la hora de hacer una llamada al sistema?
b) ¿Cuál es la utilidad de las tablas de páginas multinivel? ¿Qué restricciones tiene su diseño (i.e, que determina cuantos bits de la dirección se emplean en cada nivel)?
c) En sistema de memoria que emplee segmentación (con código, *heap*, *stack*, etc…) ¿Quién y cuando decide el número de segmento que aparece en cada dirección virtual?
d) ¿Cuántos accesos a memoria implica ejecutar la instrucción `movl $0x0,(%edi,%eax,4)` en un procesador x86, suponiendo que el sistema operativo emplea paginación en dos niveles y todas las entradas del TLB son inválidas?
e) ¿Por qué es innecesario especificar el tamaño datos a liberar cuando se emplea la función *free()*?
f) ¿Cómo se previene que se produzca un hit en el TLB si el marco que contiene la página implicada ha sido *swapeado* al disco?

CONCURRENCIA (2.5 PUNTOS)

a) ¿Por qué varios *threads* no pueden compartir el mismo *stack*?
b) ¿Qué implicaciones tienen para el diseño de la librería de gestión del HEAP que tenga soporte multithread?
c) Explicar brevemente en que consiste la concurrencia *lock-free* (o también denominada *wait-free*).
d) ¿Por qué una llamada *pthread_cond_broadcast()* es capaz de "resolver" bugs derivados de un uso inadecuado de las variables condicionales?
e) ¿En que condiciones una llamada tipo *adquire()* puede implicar una llamada al sistema?

PERSISTENCIA (1.5 PUNTOS)

a) ¿Por qué no se emplea una *freelist* para conocer los bloques de datos libres en un disco? ¿Qué se usa?
b) ¿Qué implicaciones, desde el punto de vista de fiabilidad, puede tener que un disco ignore los *write_barriers*?
c) ¿El DWPD de un disco SSD con celdas SLC es más bajo que un disco con celdas QLC? ¿Qué tipo de disco emplearías en un servidor?
d) En un sistema de ficheros LFS, ¿Cómo sabemos la posición del *imap* en el disco para un i-nodo determinado?

[1] Todas las respuestas han de ser justificadas adecuadamente

SISTEMAS OPERATIVOS AVANZADOS (PRÁCTICAS)

LUNES 17 ENERO 2022

Para cada una de las prácticas hay que explicar que comprueba el test indicado y explicar de que modo lo resuelve **tú implementación** de la práctica (fichero/s y líneas de código). Indicar el *commit* que incluye el cambio (8 primeras cifras del hash). Ser recomienda resolver el examen empleado exclusivamente el interfaz de ***gitlab***.

La solución de las prácticas 1, 2 y 3 son opcionales (Si se desea conservar la nota del examen previo se pueden dejar en blanco).

PRACTICA 1

test/P1/ctest/procs5.c

PRÁCTICA 2

test/P1/ctest/sleep_high_priority.c
test/P1/ctest/getpinfo.c

PRÁCTICA 3

tests/P3/ctests/shmem_access_persistent.c
tests/P3/ctests/shmem_count_fork.c

PRÁCTICA 4

tests/P4/ctests/badclone.c
tests/P4/ctests/thread2.c

PRÁCTICA 5

tests/P5/ctests/write4.c