EXAMEN SISTEMAS OPERATIVOS AVANZADOS

TEORIA
FEBRERO 2016

PREGUNTA 1 PROCESOS (2.5 PUNTOS)

a) ¿Cuál es la utilidad del *stack* en modo supervisor de los procesos de usuario?
b) ¿Que componente hardware se requiere para implementar el protocolo de ejecución directa limitada no cooperativo?
c) ¿Qué componente del sistema operativo es el encargado de realizar los cambios de contexto?
d) ¿Qué ayuda por parte del hardware (del procesador) requieren los cambios de contexto?
e) ¿Cual es objetivo fundamental de los algoritmos de planificación de múltiples colas con *feedback*?
f) ¿Qué problemas tienen los algoritmos de planificación de sistemas multiprocesador basados en una sola cola (como por ejemplo SQMS)?

PREGUNTA 2 MEMORIA (2.5 PUNTOS)

a) ¿Que representan A, B, C y D en el siguiente código?

```
#include <stdio.h>
#include <stdlib.h>
int y=1;
int main(int argc, char *argv[])
{
    int x = 3*y;
    printf("location of A  : %p\n", (void *) &y);
    printf("location of B  : %p\n", (void *) main);
    printf("location of C  : %p\n", (void *) &x);
    printf("location of D  : %p\n", (void *) malloc(1));
    return x;
}
```

b) ¿Por qué el siguiente código es incorrecto? Proponer como corregir el problema.

```
char *src = “hola”;
char *dst;
strcpy(dst, src)
```

c) El siguiente código para determinar la dirección física de en un sistema con segmentación. ¿Cuantos segmentos tiene el proceso, cuanto es el tamaño de los espacios de direcciones virtual y físico?

```
SEG_MASK= 0x3000
SEG_SHIFT OFFSET_MASK =12
OFFSET_MASK= 0xFFFF
Segment = (VirtualAddress & SEG_MASK) >> SEG_SHIFT
Offset  = VirtualAddress & OFFSET_MASK
if (Offset >= Bounds[Segment])
          RaiseException(PROTECTION_FAULT)
else
          PhysAddr = Base[Segment] + Offset
          Register = AccessMemory(PhysAddr)
```

d) ¿En qué condiciones hay un cambio de modo del procesador al ejecutar la llamada *malloc()* en *linux*?
e) ¿Cómo se evita que el TLB permita compartir traducciones de múltiples procesos? (i.e. entradas de diferentes tablas de páginas)
f) ¿En qué condiciones un algoritmo de reemplazo de páginas RANDOM puede ser más efectivo que uno LRU?

PREGUNTA 3 CONCURRENCIA (2.5 PUNTOS)

a) El siguiente código, correspondiente a la suma de dos vectores, no es *thread-safe* (pues pueda dar lugar a *deadlock*). Proponer una implementación alternativa que si lo sea.

```
void vector_add(vector_t *v_dst, vector_t *v_src) {
    Pthread_mutex_lock(&v_dst->lock);
    Pthread_mutex_lock(&v_src->lock);
    int i;
    for (i = 0; i < VECTOR_SIZE; i++) {
        v_dst->values[i] = v_dst->values[i] + v_src->values[i];
    }
    Pthread_mutex_unlock(&v_dst->lock);
    Pthread_mutex_unlock(&v_src->lock);
}
```

b) La siguiente función corresponde a la ejecución de un *thread*. El resultado de dicho thread no se retorna correctamente. Proponer una corrección al problema/s.

```
void *mythread(void *arg) {
     myarg_t *m = (myarg_t *) arg;
     myret_t r;
     r.x = 1;
     r.y = 2;
     return (void *) &r;
 }
```

c) ¿Por qué se requiere soporte hardware para implementar los *locks*? ¿Que aproximación usa x86?

d) En la práctica los spinning *locks* se suelen evitar mediante colas de espera. Sin embargo, esta aproximación las carreras entre *wait/wakeup* pueden bloquear el sistema ¿Cómo se puede evitar dicho problema?

e) ¿Qué utilidad tiene la llamada *pthread_cond_broadcast()*?

f) ¿Cómo se implementa un *lock* basado en semáforos? ¿Y una variable condicional?

PREGUNTA 4 PERSISTENCIA (2.5 PUNTOS)

a) ¿En qué condiciones es conveniente emplear la I/O basada en *pooling* (encuesta) en lugar de interrupciones?

b) ¿Cuál es el factor dominante a la latencia de acceso de los discos mecánicos? ¿Cómo se tiene en cuenta en FFS?

c) Describir los accesos a disco que requerimos para crear un fichero (que requiere un solo bloque de datos) en el directorio raíz en VSFS. Cuáles son los pasos para hacerlo en el directorio */usr/share/doc*.

d) ¿Qué papel juega el grupo de bloques (o también denominado grupo de cilindros) en un sistema de ficheros moderno?

e) ¿Cómo se garantiza la atomicidad de las transacciones en un sistema ficheros con *journaling* de datos?

f) ¿De qué tamaño tiene que ser el segmento en un sistema LFS para lograr al menos una tasa de utilización del 95% de al ancho de banda de un disco sata2 (300MB/s), si el tiempo de posicionamiento de las cabezas es de 10ms?

EXAMEN SISTEMAS OPERATIVOS AVANZADOS

LABORATORIO
FEBRERO 2016

NOMBRE Y FIRMA: USUARIO GITLAB:

a) (1 punto) Crear un repositorio llamado *examenFebrero* en tu cuenta de gitlab. Inicializar el contenido del repositorio con la implementación "xv6-wisc" de https://gitlab.com/AOSUC/Lab/. El repositorio tiene que ser privado y con acceso de desarrollador a vpuente@gmail.com

b) (3 puntos) Se desea realizar una nueva implementación del planificador de xv6 que un similar a MLFQ. El planificador solo tendrá dos niveles (colas). Los procesos inicialmente se alojan en la cola más prioritaria. Una vez se consuman completamente dos quantums de tiempo en esa posición, el proceso pasara a la cola menos prioritaria. Cuando tenemos más de dos procesos en una cola, se debe usar *Round-Robin*. Cuando no tenemos ningún proceso en la cola prioritaria, debemos hacer que todos los procesos del sistema promocionen a ella.

c) (2 puntos) Además, se desea que exista la posibilidad de que un proceso nunca use la cola más prioritaria. Para ello, es necesario una llamada al sistema llamada ***int demoteFather().*** Una vez se realice esta llamada, el proceso padre del que realiza la llamada no puede usar la cola más prioritaria. Deber retornar 0 cuando su llamada sea exitosa y -1 en caso contrario.

d) (2 puntos) Un proceso "marcado" por la llamada anterior, no podrán emplear más de 16KB de memoria. Si en el momento de ser marcados ya están por encima de ese límite, deben ser matados.

e) (2 puntos) Cuando un proceso "marcado" finaliza, los hijos en ejecución, deben heredar la marca del padre.

Los puntos b), c), d) y e) deben estar en *commits* independientes y deben estar realizados antes de la hora de finalización del examen.