CC41B Sistemas Operativos

Examen – Semestre Primavera 2006

Prof.: Luis Mateu

Pregunta 1

El procedimiento make sirve para compilar un programa respetando el orden dado por un conjunto de dependencias. Por ejemplo para compilar A se requiere compilar previamente B y C, para compilar B se require compilar primero D y E y para C se requiere compilar primero D. Observe que tanto B como C dependen de D. En general las dependencias forman un *grafo dirigido acíclico*.

Se dispone de una implementación secuencial de make:

```
typedef struct SrcFile {
  char *name;
  int ndep;
  struct SrcFile **deps; /* arreglo de dependencias */
} SrcFile;
void make(SrcFile *file) { seqMake(makeSet(), file); }
void seqMake(Set readySet, SrcFile *file) {
  if (!contains(readySet, file->name)) {
    int i;                        /* No ha sido compilado aun. */
    for (i= 0; i<file->ndep; i++)      /* Primero compilar  */
      seqMake(readySet,file->deps[i]); /* las dependencias. */
    compile(file->name);          /* Compilar y agregar a      */
    add(readySet, file->name); /* readySet para que no se   */
  }                            /* recompile.                */
}
```

`Make` recibe como parámetro el grafo de dependencias y retorna cuando cada nombre de archivo en el grafo fue compilado exactamente una y solo una vez. `Compile` es la única operación que requiere mucho tiempo de CPU.

Reprograme make de modo que compile los archivos en paralelo cuando sea posible. Ud. debe respetar las dependencias: no debe iniciar la compilación de X antes de que termine la compilación de Y, si X depende de Y. Cada archivo debe ser compilado exactamente una vez. Para resolver este probleme Ud. debe usar las tareas y monitores de nSystem. El tipo Set no admite llamadas concurrentes.

Indicaciones: Por cada dependencia cree una tarea encargada de hacer su `make`. Además de readySet utilice un conjunto adicional para almacenar los nombres de los archivos para los cuales ya se creó una tarea. Utilice un solo monitor para sincronizar el acceso a ambos conjuntos. No olvide liberar el monitor cuando va a compilar, de otro modo la compilación será secuencial o en el peor caso producir un *deadlock*.

Pregunta 2

(a) Explique por qué la estrategia del *working set*, como se vio en clases, se vuelve ineficiente cuando dos o más procesos comparten sus páginas de código. Dé un ejemplo de lo que podría ocurrir. Si se admite que las páginas compartidas no pueden ir a disco, explique cómo usaría los bits de atributos en la tabla de páginas para corregir la implementación de esta estrategia.

(b) En una aplicación se requiere implementar un diccionario. Se consideran las siguientes implementaciones:

```
typedef struct { char key[8]; char data[24]; } Entry;
```

`Entry dict[1000];`	`Entry *dict[1000];`

En la primera implementación toda la información se encuentra contigua en memoria. En la segunda, las entradas del diccionario pueden quedar muy dispersas y desordenadas en la memoria según la fragmentación del heap manejado por malloc. Suponga que las búsquedas en el diccionario son secuenciales. Evalúe ambas implementaciones desde la perspectiva de la TLB (*Translation Lookaside Buffer*). Explique estimando para cada implementación el peor caso del número de fallas en la TLB al hacer una búsqueda.

(c) La siguiente tabla muestra los espacios de direcciones de los procesos que ejecuta un computador Unix con páginas de 4 KB y que posee en total 16 páginas disponibles para procesos:

proceso	***código***	***datos***	***pila***
A	[4K, 12K[	[12K, 24K[	[1M-4K,1M[
B	[4K, 16K[	[16K, 32K[	[1M-8K, 1M[

Suponga que el proceso A invoca sbrk pidiendo 16 KB adicionales para datos y luego B invoca *fork*. Indique los contenidos posibles de las tablas de páginas de todos los procesos justo después que B invoca *fork*.

(d) Explique por qué en Unix cuando se accede al contenido de un puntero nulo se produce un *segmentation fault*.

Pregunta 3

i. Se tiene un archivo que no requiere bloques de indirección doble en una partición Unix con bloques de 2 KB. Se agrega un byte a este archivo y se crea el bloque de indirección doble. Haga un diagrama mostrando inodo, bloques de datos y de indirección. ¿De qué tamaño es el archivo?

ii. Un programador usuario de un computador Linux descubre que la implementación de `read(fd, n, buf)` en el núcleo no valida que `buf` apunte a un área de memoria del proceso. Explique cómo este programador podría lograr correr su propio código en modo sistema (¡sin ser el usuario root!).

iii. Suponga que un proceso lee el bloque 14. Mientras se lee este bloque en el disco distintos procesos solicitan leer los siguientes bloques de un disco: 15, 13, 18, 7. En qué orden se leerán estos 4 bloques si la estrategia de scheduling de disco es (A) shortest seek first y (B) método del ascensor. Explique.