CC4302 Sistemas Operativos

Examen– Semestre Primavera 2012

Prof.: Luis Mateu

Pregunta 1

Ud. dispone de 8 máquinas, enumeradas de 1 a 8, para ejecutar comandos. Por ejemplo para ejecutar el comando "`gcc fib.c`" en la máquina 4 se debe invocar:

```
int rc= ejecutar("gcc fib.c", 4);
```

La función `ejecutar` es dada y es síncrona, lo que significa que solo retorna una vez que se haya completado la ejecución del comando.

Programe en nSystem la siguiente API que puede ser usada concurrentemente desde múltiples tareas:

`Orden *ejecutar_asincrono(char *cmd)` Ordena ejecutar asincrónicamente `cmd` en alguna de las 8 máquinas disponibles. Es decir esta función retorna de inmediato, sin esperar la ejecución del comando. Retorna un objeto identificador de la orden.
`int esperar_comando(Orden *orden)` Espera que termine la ejecución del comando asociado a `orden`. Entrega el valor retornado por la función `ejecutar`.

Por ejemplo el código de una tarea que usa esta API podría ser:

```
Orden *orden= ejecutar_asincrono("make");
hacer_algo_distinto();
rc= esperar_comando(orden);
if (rc!=0)
  error(...);
```

Concretamente se pide definir el tipo Orden, programar una función de inicialización, las funciones `ejecutar_asincrono` y `esperar_comando`, más otras funciones que Ud. requiera.

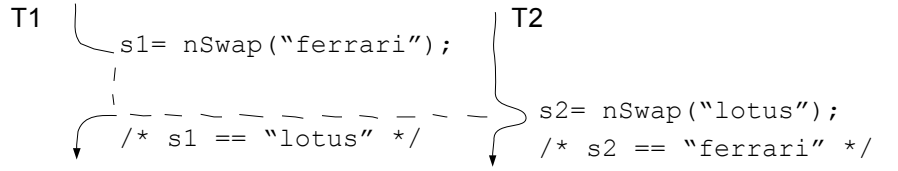
Restricciones: Su implementación debe aprovechar al máximo las 8 máquinas disponibles, pero *una máquina puede ejecutar un solo comando a la vez*. Resuelva el problema de la siguiente manera. Cuando una tarea invoca `ejecutar_asincrono`, ésta solo encola la orden de ejecución. Por cada una de las 8 máquinas disponibles, levante una tarea auxiliar que se encarga de extraer órdenes de la cola y atenderlas invocando `ejecutar`. Utilice la máquina asociada a esa tarea auxiliar. Para la sincronización Ud. debe usar los monitores de nSystem.

Pregunta 2

Parte a.- Se desea agregar a la API de nSystem la siguiente función:

```
char  *nSwap(char *s);
```

Esta función es invocada concurrentemente desde múltiples tareas y varias veces. Sirve para intercambiar punteros a caracteres con cualquier otra tarea que invoque `nSwap`. La siguiente figura explica el funcionamiento de `nSwap`. T1 invoca `nSwap` pero no hay otra tarea pendiente en un `nSwap`, entonces T1 queda pendiente. Más tarde T2 invoca `nSwap` y encuentra que T1 está pendiente en un `nSwap`, entonces T1 y T2 intercambian los punteros. Es decir en T1 `nSwap` retorna el puntero a "lotus" y en T2 `nSwap` entrega el puntero a "ferrari".



Programe `nSwap` usando los procedimientos de bajo nivel de nSystem (START_CRITICAL, Resume, PutTask, etc.). Ud. *no puede usar* otros mecanismos de sincronización ya disponibles en nSystem, como semáforos, monitores, mensajes, etc.

Parte b.- Considere una máquina con 8 cores físicos sin un núcleo de sistema operativo, y por lo tanto no hay *scheduler* de procesos y tampoco threads. Los *spin-locks* son la única herramienta de sincronización disponible. Los 8 cores ejecutan código en paralelo y comparten la memoria. Programe la función `swap` equivalente a `nSwap` de la parte a. Dado que no hay una cola de procesos *ready*, para esperar no le queda otra que hacer *busy-waiting*.

Pregunta 3

i. ¿Cuantos *spink-locks* se necesitan en un núcleo clásico de Unix para una máquina multi-core? ¿Y si es mono-core?

ii. Haga una tabla comparando las 2 estrategias de *paginamiento en demanda* vistas en el curso. En las filas considere (a) sobrecosto en tiempo de ejecución cuando la memoria física sobra, (b) *page faults* cuando hay penuria de memoria y hay muchos procesos en ejecución, (c) *page faults* cuando hay penuria de memoria, pero hay un solo proceso en ejecución.

iii. Haga una tabla comparando el uso de bloques de 1 KB versus bloques de 4 KB en una partición de disco que usa el sistema de archivos Unix. En las filas considere (a) fragmentación interna, (b) número de bloques de indirección requeridos, (c) velocidad de transferencia para archivos secuenciales, y (d) tiempo de lectura durante un acceso directo.

iv. Suponga que el disco se encuentra ocupado leyendo el bloque 1000 pedido por el proceso P1. Entonces los procesos P2, P3, P4, P5 y P6 piden leer los bloques 500, 6000, 3000, 7000 y 2000 respectivamente. ¿En qué orden atendería las peticiones de los procesos y por qué?