CC41B Sistemas Operativos

Control 2 – Semestre Primavera 2005

Prof.: Luis Mateu

Pregunta 1 (60%)

La siguiente especificación describe la API de un *buffer* de tamaño 1, que será usado por múltiples productores y consumidores. Un productor que deposita el primer ítem no se queda bloqueado, pero si se deposita un segundo ítem, sin haber extraído el primero, el productor sí se queda bloqueado hasta que un consumidor extraiga el primer ítem. Esta es la API:

`Box *pbox= makeBox();`	Construye un buffer de tamaño 1.
`putBox(pBox, 5);`	Deposita un entero. Si el buffer está lleno, se bloquea hasta que se extraiga un ítem.
`int item= getBox(pBox);`	Extrae un entero. Si el buffer está vacío, se bloquea hasta que se deposite un ítem.

Esta API se ha implementado de la siguiente manera usando *spin-locks*:

```
typedef struct {
  int fullSpinLock;
  int emptySpinLock;
  int val;
}
  Box;

Box *makeBox() {
  Box *pb= (Box*)
        nMalloc(sizeof(*pb));
  pb->fullSpinLock= OPEN;
  pb->emptySpinLock= OPEN;
  return pb;
}
```

```
void PutBox(Box *pb, val) {
  spinLock(pb->emptySpinLock);
  spinUnlock(pb->fullSpinLock);
  pb->val= val;
}
int GetBox(Box *pb) {
  int rc;
  spinLock(pb->fullSpinLock);
  rc= pb->val;
  spinUnlock(pb->emptySpinLock);
  return rc;
}
```

(a) Indique si esta solución es correcta, es decir si las operaciones hacen lo que la especificación dice que tiene que hacer. Si encuentra algún error, indique qué hay que corregir. (0.5 puntos)

(b) Indique si esta solución es eficiente. Explique. (0.5 puntos)

(c) Incorpore esta API como mecanismo de sincronización *nativo* de nSystem. Es decir, Ud. debe implementar esta API 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. (4 puntos)

(d) Ahora se le pide una implementación *eficiente* de esta API para un núcleo multi-threaded de Unix (i.e. para multiprocesadores). *Describa* qué cambios hay que hacer a su implementación de la parte (c) para esta nueva implementación y *explique* por qué es más eficiente que la implementación que usa spin-locks presentada en este enunciado. Suponga que dentro de este núcleo Ud. dispone de los procedimientos spinLock, spinUnlock y de procedimientos similares a los de nSystem como kResume, kPutTask, kPushTask, etc. pero no START_CRITICAL ni END_CRITICAL. (1 punto)

Pregunta 2 (40%)

(a) Un sistema Unix ejecuta los procesos A y B. Ambos procesos ocupan un código de 8 KB y una pila de 4 KB. Considere los casos en que A y B son:

i. procesos pesados con datos de 8 KB y 16 KB respectivamente.
ii. procesos semi-ligeros con datos de 8 KB.
iii. procesos semi-pesados con datos de 4 KB y 8 KB respectivamente y comparten 4 KB.

Indique para cada caso el contenido de las tablas de páginas de ambos procesos, para una posible asignación de memoria en un computador que implementa *paginamiento en demanda* y dispone de 10 páginas para los procesos (y suficientes páginas para el núcleo), c/u de 4 KB. (3 puntos)

(b) En un sistema Unix se desea agregar las llamadas a sistema *saveProcess* y *restoreProcess*. La llamada `saveProcess(char *filename)` almacena en el archivo `filename` una foto del proceso en el instante de la invocación (su código, datos y pila). La llamada `restoreProcess(char *filename)` descarta el espacio de direcciones del proceso que hace la invocación y restaura el espacio del proceso grabado en `filename`, retomando su ejecución en el punto en donde este último había invocado `saveProcess`.

i. Explique brevemente como implementaría estas llamadas en un sistema paginado. (1.5 puntos)
ii. Explique por qué no es posible implementar estas llamadas en un sistema que no posee una MMU (*Memory Management Unit*), aún cuando este sistema implemente múltiples procesos, como es el caso de los antiguos computadores Commodore Amiga. (1.5 puntos)