CC41B : Sistemas Operativos
Control 2–Semestre Primavera'96

Prof.: Luis Mateu.
Prof. Aux.: Daniel Verdugo

Pregunta 1

Conteste brevemente:

- Parte a.- (25%)

  Indique por qué es importante que la implementación que se haga en el núcleo de la llamada al sistema `read` valide el puntero que recibe como parámetro (en donde se leen los datos). ¿Qué tipo de validación necesita hacer el núcleo? Argumente en términos de seguridad del núcleo: Cómo podría un usuario malicioso aprovechar esta situación para ejecutar su propio código en modo sistema.

- Parte b.- (75%)

  Se tiene la siguiente implementación de mensajes a partir de candados (locks). Este sistema de mensajes no acepta *time-outs* en la recepción y además la respuesta no recibe un código de retorno.

```
void Send(Task task, void *msg)
{
  Task this_task= CurrentTask();
  task->msg= msg;
  task->send_task= this_task;
  Lock(task->send_lock);
  Unlock(task->receive_lock);
  Lock(this_task->reply_lock);
}

void* Receive(Task *ptask)
{
  Task this_task= CurrentTask();
  Unlock(this_task->send_lock);
  Lock(this_task->receive_lock);
  *ptask= this_task->send_task;
  return this_task->msg;
}

void Reply(Task task)
{
  Unlock(task->reply_lock);
}
```

El descriptor de proceso posee 3 locks: `send_lock`, `reply_lock` y `receive_lock`. Todos estos locks parten cerrados al crear el descriptor de proceso.

1. Invente un escenario de ejecución de tareas que muestre que esta solución es incorrecta. Use diagramas de progreso de tareas que grafiquen las llamadas de procedimientos en función del tiempo.
2. Corrija esta solución sólo moviendo líneas.
3. Critique esta solución en el caso de que los candados estén implementados mediante *spin-locks*. Es decir, indique por qué esta solución no sería satisfactoria.

Pregunta 2

Se desea agregar a la implementación que Ud. vió de nSystem un nuevo mecanismo de sincronización entre procesos. Este mecanismo esta basado en *condiciones*, que se manipulan con los siguientes procedimientos:

- `Condition MakeCondition()`: Construye y retorna una nueva condición.
- `void WaitCondition(Condition cond)`: Bloquea el proceso que lo invoca hasta que otro proceso invoque en el futuro `NofityCondition` sobre `cond`. Observe que este procedimiento siempre se bloquea, sin importar los `NotifyCondition` que se hayan hecho en el pasado.
- `void NotifyCondition(Condition cond)`: Desbloquea *todos* los procesos que esperaban la condición `cond` y retorna de inmediato. El orden en que se retoman los procesos puede ser cualquiera.

Implemente estos procedimientos en nSystem. Suponga que nSystem no posee semáforos, ni mensajes ni monitores y por lo tanto deberá basar su implementación en los procedimientos:

- `void ResumeNextReadyTask() /* O simplemente Resume() */`
- `void START_CRITICAL()`
- `void END_CRITICAL()`
- `Queue MakeQueue()`
- `void PutTask(Queue queue, nTask task)`
- `void PushTask(Queue queue, nTask task)`
- `nTask GetTask(Queue queue)`
- `int EmptyQueue(Queue queue)`