CC41B : Sistemas Operativos
Control 2–Semestre Primavera'2001

Prof.: Luis Mateu.

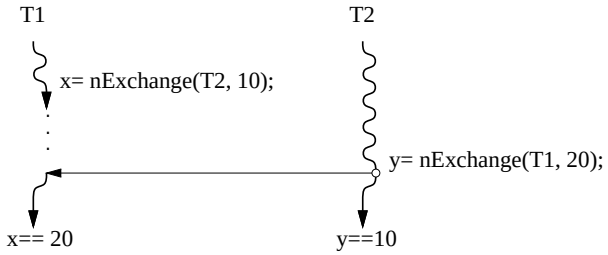
Sin apuntes, 1 hora 30 minutos

Pregunta 1

Con el fin de reducir el tamaño de nSystem se ha eliminado todo el código correspondiente a los semáforos, mensajes y monitores. En su reemplazo se le pide a Ud. que implemente como herramienta de sincronización la siguiente llamada:

```
int nExchange(nTask task, int val);
```

Esta llamada sirve para que dos tareas intercambien un valor entero. Para ello las dos tareas deben invocar este procedimiento pasando como argumento el identificador de la tarea con quien realizarán el intercambio (`task`) y el valor que desean intercambiar (`val`).



Como se muestra la figura, el intercambio de valores entre dos tareas T1 y T2 solo se puede producir cuando T1 invocó nExchange especificando T2 como parámetro en `task` y T2 invocó nExchange especificando T1 como parámetro en `task`. El valor retornado en T1 es el valor (`val`) pasado por T2 y el retornado en T2 es aquel que pasó T1.

Cuando una tarea T1 invoca nExchange y su contraparte T2 aún no lo hace, T1 se queda en espera hasta que T2 invoque nExchange especificando como parámetro T1. Si la contraparte T2 invoca nExchange indicando una tercera tarea T3, T1 debe permanecer en espera.

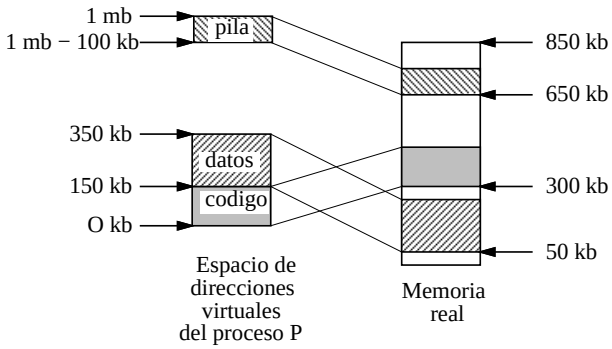
Implemente nExchange usando para ello los procedimientos que internamente usa nSystem para implementar herramientas de sincronización. Si lo necesita puede hacer cambios al descriptor de proceso, agregar nuevos estados de espera, etc.

Torpedo con los procedimientos internos de nSystem (incluye más de lo que necesita) :

```
typedef struct Task {
  int status;
  int rc;
  ...
}
  *nTask;
Queue ready_queue;
nTask current_task;
```

```
void START_CRITICAL();
void END_CRITICAL();
void PutTask(Queue queue, nTask task);
void PushTask(Queue queue, nTask task);
nTask GetTask(Queue queue);
int EmptyQueue(Queue queue);
Queue MakeQueue();
void Resume();
```

Pregunta 2



La figura muestra a la izquierda el espacio de direcciones virtuales de un proceso Unix que ocupa 450 KB de memoria. El proceso corre en un computador de 1 MB con arquitectura segmentada. A la derecha se muestra la ubicación actual de los segmentos en la memoria real. Se dispone en total de 850 KB para segmentos (el resto se ocupa en estructuras de datos del núcleo).

a.- Suponga que el proceso invoca a `sbrk` para hacer crecer su área de datos de 200 KB a 400 KB. Modifique la figura para mostrar una posible asignación de memoria después de la llamada a `sbrk`. Muestre en su figura las direcciones que ocupan ahora los segmentos en el espacio de direcciones virtuales y la memoria real.

b.- Haga la tabla de segmentos del proceso después de invocar `sbrk`. Indique base virtual, límite virtual, desplazamiento y atributos para cada segmento.

c.- Suponga que ahora el proceso invoca a `fork`. Explique cómo logra el núcleo del sistema operativo satisfacer este nuevo requirimiento a pesar que no hay memoria disponible para todos los segmentos del nuevo proceso.