CC41B : Sistemas Operativos
Control 2–Semestre Primavera'98
Prof.: Luis Mateu.
Sin apuntes, 1 hora 45 minutos

Pregunta 1 (50%)

En nSystem la operación `nSleep(dt)` duerme la tarea que lo invoca por `dt` milisegundos. Se desea agregar la operación `nWakeup` que despierta una tarea antes del tiempo estipulado. La semántica del nuevo `nSleep` y de `nWakeup` será la siguiente :

- `int nSleep(int dt)` : Si lo invoca la tarea A, hace que A duerma hasta que se cumplan `dt` milisegundos o hasta que una segunda tarea B despierte la tarea A con `nWakeup`. El valor retornado es 0 si se cumple el tiempo o 1 si la tarea es despertada.
- `int nWakeup(nTask task)` : Despierta la tarea `task`. Si `task` no estaba durmiendo, no se hace nada y se retorna 0. Si `task` estaba durmiendo, se despierta y se retorna 1.

Implemente ambos procedimientos (`nSleep` y `nWakeup`) para nSystem. Su implementación no puede basarse en otras primitivas de sincronización como semáforos, mensajes, etc. Si lo requiere, Ud. puede agregar variables al descriptor de tarea o nuevos estados. Le serán de utilidad los siguientes recursos ya disponibles en nSystem :

- `Queue ready_queue` : La cola de tareas "ready".
- `void Resume()` : Retoma la primera tarea presente en la cola de tareas "ready".
- `void START_CRITICAL()` : Deshabilita las interrupciones.
- `void END_CRITICAL()` : Habilita las interrupciones.
- `void PutTask(Queue queue, nTask task)` : Agrega una tarea al final de una cola.
- `void ProgramTask(int dt)` : Se invoca antes de suspender la tarea actual para hacer que ésta se active automáticamente después de `dt` milisegundos.
- `void CancelTask(nTask task)` : Cancela una orden previa de auto-activación de la tarea `task`.

Pregunta 2 (25%)

Un sistema operativo ofrece como parte de su API un mecanismo de envío/recepción y respuesta de mensajes para *procesos pesados*. Este mecanismo consiste en las siguientes llamadas al sistema :

- `void send(int pid, char* pbuff, int size)` : envía un mensaje al proceso con identificador `pid`. El mensaje contiene `size` bytes a partir de la dirección `pbuff` en el espacio de direcciones del emisor. Esta llamada se bloquea hasta que el receptor responda con `reply`.

- `char* receive(int* ppid, int* psize)`: recibe un mensaje proveniente de cualquier otro proceso. El identificador del emisor queda en `*ppid` y el tamaño del mensaje en `*psize`. El núcleo entrega al receptor el mensaje emitido en un área del espacio de direcciones del receptor no asignada previamente. La dirección de esta área es el valor de retorno de `receive` (que en general no coincide con la dirección en el emisor).

  El proceso receptor puede modificar el mensaje recibido, y tales modificaciones serán vistas por el emisor al ser respondido.

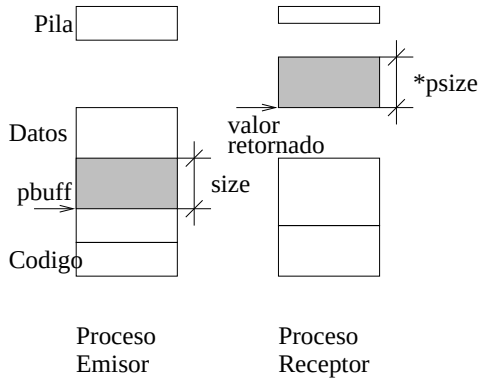
- `void reply(int pid)`: responde el mensaje recibido de `pid`. Desde ese instante el receptor no puede seguir accesando el mensaje recibido.

i.- Explique en palabras como implementar estas operaciones en una arquitectura que admite paginamiento (con páginas de tamaño 4KB). Indique qué hacer al recibir el mensaje y qué hacer al responderlo.

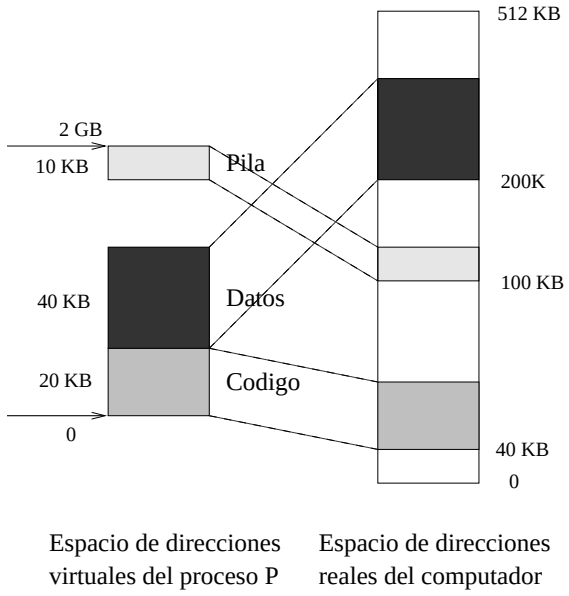
ii.- Ejemplique su explicación para el caso en que un proceso A envía un mensaje de 10 KB a un proceso B. Haga un diagrama que muestre la asignación de páginas de ambos procesos a la memoria real justo después de la recepción.

iii.- Suponga que el contenido del mensaje incluye punteros al mismo mensaje. Discuta acerca de la validez de tales punteros en el receptor.

Pregunta 2



Pregunta 3



Pregunta 3 (25%)

La figura muestra la asignación de memoria para un proceso cualquiera.

i.- Haga la tabla de segmentos del proceso. Indique los valores de la base virtual, límite virtual, desplazamiento y atributos para los 4 segmentos del proceso.

ii.- Suponga que el proceso hace fork. Modifique la figura para mostrar una posible asignación de la memoria, justo después del fork.