Ejercicio 1 de procesos y memoria (febrero 2000)

Enunciado

En un sistema monolítico se desea incluir un servicio de comunicación por paso de mensajes que ofrezca los servicios `Enviar(puerto, mensaje)` y `Recibir(puerto, mensaje)` con la funcionalidad que indica su nombre. Se pretenden estudiar los aspectos de sincronización asociados a estas primitivas.

En el sistema existen las dos siguientes funciones internas para la gestión del bloqueo de un proceso a la espera de un evento y su posterior desbloqueo:

- `Bloquear(proceso, lista)`: Incluye el proceso en la lista de espera especificada correspondiente a un determinado evento, lo pone en estado bloqueado y realiza un cambio de contexto al proceso seleccionado por el planificador. Todo proceso bloqueado está incluido en alguna lista de espera.
- `Desbloquear(lista)`: Elimina el primer proceso bloqueado de la lista de espera especificada poniéndolo en estado listo.

Se consideran dos alternativas a la hora de diseñar este servicio: comunicación de tipo asíncrona (o sea, envío no bloqueante y uso de *buffering*) o de tipo síncrona (o sea, envío bloqueante y sin *buffering*). En ambos casos la recepción es bloqueante. Por simplicidad, se considera que todos los mensajes son del mismo tamaño y que, en el caso de comunicación asíncrona, el tamaño del *buffer* es ilimitado.

Se pide:

1. Para el caso de un modo asíncrono, especificar cómo se implementan los servicios `Enviar` y `Recibir` mostrando qué llamadas a `Bloquear` o a `Desbloquear` se realizan y bajo qué condiciones, así como en qué momento se realizan las copias del mensaje. Previamente se debe determinar qué lista o listas de espera estarán asociadas con cada puerto.
2. Lo mismo que en el apartado anterior pero para una comunicación de tipo síncrona.
3. Suponiendo que en el sistema se usa un algoritmo de planificación expulsivo basado en prioridades, analizar en qué puntos de los servicios `Enviar` y `Recibir` especificados en los dos apartados anteriores se pueden producir cambios de contexto y de qué tipo son los mismos (voluntarios o involuntarios). ¿Qué diferencia habría dependiendo de si se trata de un sistema operativo que no permita que haya concurrencia entre las llamadas al sistema (como, por ejemplo, Linux o el minikernel) y uno que sí lo permita?
4. Para conseguir que la eficiencia en la comunicación de mensajes grandes en modo asíncrono sea similar a la del síncrono, se plantea un mecanismo alternativo de transferencia asíncrona con las siguientes características:
   - Un proceso que quiere enviar un mensaje, usa una primitiva denominada `m_malloc` que reserva una zona de memoria para el mensaje tal que esté alineada al principio de una página y ocupe un número entero de páginas. Un programa que envía un mensaje tendría la siguiente estructura:

```
int main() {
mensaje *m;

m= m_malloc(TAM);
Preparar el mensaje
Enviar(puertoX, m);
.................
El proceso puede volver a usar la misma zona reservada para preparar
un segundo mensaje
.................
}
```

   - Un proceso que quiere recibir un mensaje no va a reservar espacio para el mismo, sino que el sistema operativo lo hace internamente por él. Un programa que quiera recibir un mensaje tendría la siguiente estructura:

```
int main() {
mensaje *m;

Recibir(puertoX, m);
.................
}
```

Diseñe un mecanismo de transferencia optimizado utilizando las técnicas de gestión de memoria que considere oportunas. Debe especificar qué operaciones de gestión de memoria se llevarían a cabo en el servicio `Enviar` y cuáles en `Recibir`.

Solución

1. Comunicación asíncrona. Asociado con cada puerto existirá una lista (a la que denominaremos `recibiendo`) donde se enlazarán los procesos bloqueados esperando recibir un mensaje por ese puerto. Además, habrá una lista de los mensajes pendientes de recibir por ese puerto (`mensajes`). Por simplicidad se supone que puede haber un número ilimitado de mensajes en dicha lista.

   El *pseudocódigo* de `Enviar` sería el siguiente:

```
Enviar(puerto, mensaje)
{
        ppuerto=&tabla_puertos[puerto];

        /* copia el mensaje al final de la lista de mensajes */
        Copiar_ultimo(mensaje, ppuerto->mensajes);

        Si (ppuerto->recibiendo!=NULL)
                Desbloquear(ppuerto->recibiendo);
}
```

   Y el *pseudocódigo* de `Recibir` el siguiente:

```
Recibir(puerto, mensaje)
{
        ppuerto=&tabla_puertos[puerto];

        Mientras (ppuerto->mensajes==NULL)
                Bloquear(proc_actual, ppuerto->recibiendo);

        /* copia el primer mensaje de la lista de mensajes a la
        dirección especificada en la llamada */
        Copiar_primero(ppuerto->mensajes, mensaje);
}
```

   Nótese el uso de un bucle (`mientras`) para asegurarse de que cuando ejecuta un proceso después de haber estado bloqueado en `Recibir`, todavía queda algún mensaje disponible.

2. Comunicación síncrona. Asociado con cada puerto existirán dos listas. Una similar a la del apartado anterior (`recibiendo`) donde se enlazarán los procesos bloqueados esperando recibir un mensaje por ese puerto. Otra lista que contendrá los procesos que han mandado un mensaje pero están bloqueados esperando que un proceso lo lea (`mandando`). En este caso no habrá una lista de mensajes ya que la comunicación síncrona no usa *buffering*. Sin embargo, será necesario que en el BCP de cada proceso haya un campo (`buffer`) que contenga la dirección del *buffer* donde el proceso quiere recibir el mensaje, en el caso de un proceso bloqueado al recibir, o del *buffer* que contiene el mensaje, si se trata de un proceso bloqueado al mandar.

   El *pseudocódigo* de `Enviar` sería el siguiente:

```
Enviar(puerto, mensaje)
{
        ppuerto=&tabla_puertos[puerto];
        Si (ppuerto->recibiendo==NULL) {
                /* guardo la dirección donde se almacena el mensaje */
```

```
                proc_actual->buffer=mensaje;
                Bloquear(proc_actual, ppuerto->mandando);
        }
        Si no {
                /* obtiene el BCP del primer proceso en la lista recibiendo */
                pproc=ppuerto->recibiendo;
                /* copia mensaje al buffer donde dicho proceso lo espera */
                Copiar(mensaje, pproc->buffer);
                Desbloquear(ppuerto->recibiendo);
        }
}
```

Y el *pseudocódigo* de `Recibir` el siguiente:

```
Recibir(puerto, mensaje)
{
        ppuerto=&tabla_puertos[puerto];
        Si (ppuerto->mandando==NULL) {
                /* guardo la dirección donde se almacena el mensaje */
                proc_actual->buffer=mensaje;
                Bloquear(proc_actual, ppuerto->recibiendo);
        }
        Si no {
                /* obtiene el BCP del primer proceso en la lista mandando */
                pproc=ppuerto->mandando;
                /* copia mensaje desde el buffer del proceso remitente */
                Copiar(pproc->buffer, mensaje);
                Desbloquear(ppuerto->mandando);
        }
}
```

3. Al tratarse de un esquema de planificación expulsivo, se podrán producir cambios de contexto en dos situaciones:
   - Cambios de contexto voluntarios cuando un proceso se bloquea.
   - Cambios de contexto involuntarios cuando un proceso se desbloquea y tiene más prioridad que el proceso actual.

   Por lo tanto, en el caso de una comunicación asíncrona se podrían dar los siguientes cambios de contexto:

   - En `Enviar` se puede producir un cambio de contexto involuntario si al desbloquear a un proceso receptor, éste tiene más prioridad que el proceso actual.
   - En `Recibir` se puede producir un cambio de contexto voluntario si se bloquea el proceso por no existir ningún mensaje pendiente.

   Por lo que se refiere al caso de una comunicación síncrona se podrían dar los siguientes cambios de contexto:

   - Como ocurría en el caso asíncrono, en `Enviar` se puede producir un cambio de contexto involuntario si al desbloquear a un proceso receptor, éste tiene más prioridad que el proceso actual. Además, podrá haber un cambio de contexto voluntario si el proceso se bloquea al no haber un proceso esperando el mensaje.
   - Como ocurría en el caso asíncrono, en `Recibir` se puede producir un cambio de contexto voluntario si se bloquea el proceso por no existir ningún proceso intentado enviar un mensaje. Asimismo, se podría dar un cambio de contexto involuntario si al desbloquear a un proceso remitente, éste tiene más prioridad que el proceso actual.

   El que se trate de un sistema operativo que permita llamadas concurrentes o no influiría en los cambios de contexto involuntarios. En el caso de un sistema que no lo permita, cuando se desbloquea un proceso con más prioridad que el actual, no se haría un cambio de contexto en ese instante sino que se retardaría hasta que el proceso actual termine la llamada al sistema en curso. Si se trata de un sistema que permita la ejecución concurrente de llamadas, el cambio de contexto se produciría justo en el momento de desbloquear al proceso.

4. En este apartado se pueden plantear múltiples soluciones para optimizar la transferencia asíncrona. A continuación se muestra una posible estrategia basada en el uso del "copy-on-write".

   En la llamada `Enviar`, en vez de copiar el mensaje a un *buffer* del sistema, se puede reservar una zona del mapa de memoria del sistema operativo de manera que apunte a las páginas que contienen el mensaje a enviar. Estas páginas se marcarán como COW tanto en el mapa del proceso remitente como en el mapa del sistema operativo. Así, cualquier modificación que haga el proceso remitente sobre una página de esa zona creará una copia propia de dicha página.

   Mientras no se produce la recepción, sólo aquellas páginas que no hayan sido modificadas por el proceso remitente seguirán marcadas como COW. Las modificadas sólo estarán referenciadas desde el mapa del sistema operativo puesto que el proceso tendrá su propia copia.

   En la llamada `Recibir`, el sistema operativo reservará una zona del mapa del proceso receptor haciendo que referencie a las páginas que contienen los datos del mensaje accesibles a través del mapa de memoria del propio sistema operativo. Sólo las páginas con COW en el mapa del sistema operativo se marcarán como COW en el mapa del proceso receptor. Por último, el sistema operativo eliminará de su mapa de memoria las referencias a esas páginas.

   Así, al completarse la transferencia, el proceso receptor tiene los datos del mensaje en su zona estando marcadas como COW sólo las páginas que todavía no ha modificado el proceso receptor. El resultado de todo este proceso es que se realiza una transferencia asíncrona pero realizando, como máximo, sólo una copia como ocurre en el caso síncrono. Esta única copia de cada página del mensaje se produciría al tratar la excepción correspondiente a la escritura sobre una página marcada como COW.