Departamento de Arquitectura

Examen de Sistemas Operativos

12 de agosto de 2020

Lea detenidamente las siguientes instrucciones. No cumplir los requerimientos puede implicar la pérdida del examen.

Formato

- Indique su nombre completo y número de cédula en cada hoja (no se corregirán las hojas sin nombre). Numere todas las hojas e indique la cantidad total de hojas en la primera.
- Escriba las hojas de un solo lado y empiece cada problema en una hoja nueva.
- Si se entregan varias versiones de un problema solo se corregirá la primera de ellas.

Dudas

- Sólo se contestarán dudas de letra.
- No se aceptarán dudas en los últimos 30 minutos del examen.

Material

- El examen es SIN material (no puede utilizarse ningún apunte, libro ni calculadora). Sólo puede tenerse las hojas del examen, lápiz, goma y lapicera en su banco. Todas sus demás pertenencias debe colocarlas en el piso debajo de su asiento.

Aprobación

- Para aprobar el examen se debe tener un mínimo de 60 puntos.

Finalización

- El examen dura **3 horas**.
- Al momento de finalizar el examen no se podrá escribir absolutamente nada en las hojas, debiéndose parar e ir a la fila de entrega. Identificar cada una de las hojas con nombre, cédula y numeración forma parte de la duración del examen.

Problema 1 (30 pts)

Conteste justificando cada una de sus respuestas.

(a) I. (3 pts) Explique las diferencias entre los monitores Hoare y Mesa. ¿En qué condiciones se puede asegurar que un código funciona siempre para cualquiera de los dos casos?

II. (5 pts) ¿Es posible implementar semáforos con monitores? Justifique aportando un contra-ejemplo o la implementación correspondiente.

(b) Describa brevemente

I. (4 pts) ¿Qué es y qué funciones tiene el planificador?

II. (2 pts) ¿Cuándo se dice que un planificador es expropiativo?

(c) Describa brevemente

I. (4 pts) ¿En qué consiste la técnica de memoria virtual llamada paginación?

II. (3 pts) ¿Cómo se implementa la memoria compartida cuando se usa paginación?

III. (2 pts) ¿Qué es la hiperpaginación y en qué ocasiones se presenta? ¿Cómo afecta al Sistema Operativo?

(d) Describe brevemente

I. (3 pts) Las estructuras que el sistema Operativo mantiene en memoria para manejar el sistema de archivos.

II. (2 pts) Tres atributos de un archivo

III. (2 pts) El método de asignación indexada para la disposición de los datos de los archivos en disco

Solución:

(a) I. Cuando un procedimiento invoca a un signal(), hay 2 procesos que pueden seguir activos:
1) El proceso que está invocando el signal()
2) El proceso que se despierta con el wait()
En la variante de Hoare, el proceso que invoca el signal() devuelve el monitor y queda en una cola, esperando (al menos) a que termine el proceso despertado. El proceso que es despertado comienza a trabajar en forma inmediata.
En la variante de Mesa, el proceso que invoca el signal() pone al proceso que estaba esperando en el wait() en una cola de listos y sigue ejecutando. Cuando termina el proceso comienza a ejecutar el primero de la cola de listos.
Para asegurarnos que un código funciona de la misma manera para ambas variantes, deben cumplirse las siguientes condiciones:
- Solo debe existir una ejecución de signal() en el código del monitor
- La ejecución de signal() debe ser la última instrucción del monitor.

II. Si, es posible ya que se puede implementar semáforos con monitores.

```
Monitor Semaforo
var cont: Integer
var bloq: Condition

Procedure Init (N)
   cont:= N
end Procedure

Procedure P()
   if cont=0 then
      bloq.wait ();
   end if
   cont := cont - 1;
en procedure

procedure V()
   cont := cont+1;
   bloq.signal();
end procedure

begin
end
end Monitor
```

(b) I. El planificador es un componente de software del sistema. Si bien la función principal del mismo es la elección del próximo proceso a ejecutar (corto plazo), existen distintos tipos de planificadores que tienen distintas funciones:
Largo plazo
Determinar qué programas son admitidos al sistema para ejecución
Controlar el grado de multiprogramación
Mediano plazo
Determinar si agregar más programas a los que ya están parcial o totalmente en memoria principal

Corto plazo

Determinar el próximo proceso que será ejecutado por el procesador

II. El planificador puede ser invocado en varios momentos.

Se dice que el planificador es expropiativo si éste es invocado en:

- Cuando un proceso cambia del estado ejecutando al estado pronto (Pej al ocurrir una interrupción)
- Cuando ocurre una interrupción de E/S y el proceso pasa del estado bloqueado al estado pronto
- Cuando se crea un proceso

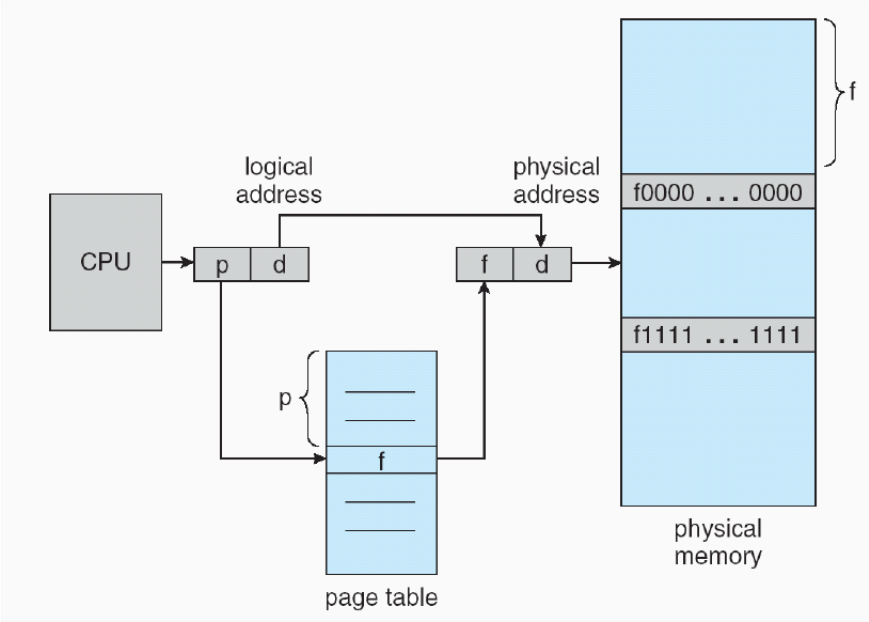
Esto significa que se le puede quitar el procesador al proceso que estaba en ejecución.

(c) Describa brevemente

I. La paginación es una técnica que divide a la memoria física en particiones de tamaño fijo llamados frames. El espacio de direcciones virtuales se divide en particiones de tamaño fijo del mismo tamaño que los frames, denominadas páginas.

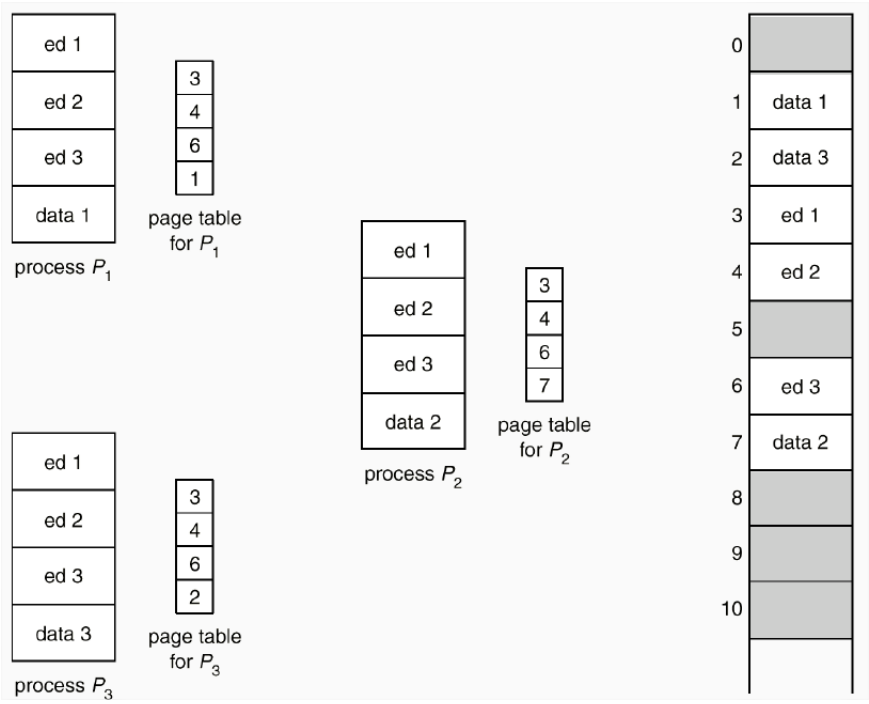
Cuando un proceso ejecuta sus páginas son cargadas en los frames de memoria principal y en disco (swap). La transferencia entre memoria principal y disco es siempre en unidad de página (ya que tienen el mismo tamaño).

Las direcciones virtuales en paginación se componen de un número de página (page number) y un desplazamiento (offset). El número de página es un índice sobre una tabla de página (page table) y el desplazamiento es el desplazamiento dentro del frame. La mayoría de los Sistemas Operativos asignan una tabla de páginas por proceso con el siguiente esquema.



II. Para implementar memoria compartida con paginación, se referencian los frames compartidos desde las tablas de paginas de los distintos procesos.

Ejemplo:



En este ejemplo, los frames 3,4,6 son compartidos por los procesos P1, P2, P3

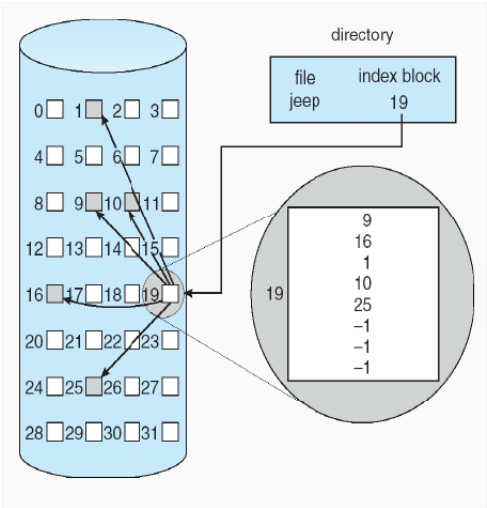
III. La hiperpaginación es un estado del Sistema Operativos en el que alguno de sus procesos puede pasar más tiempo paginando que ejecutando. Esta situación puede ocurrir cuando un proceso tiene un uso activo de un mayor número de frames que los asignados por el sistema. En esta situación tendrá un alto porcentaje de page fault ya que sus frames están activos. Debido a esta condición, el SO estará ejecutando de manera continua la rutina de atención a un fallo de página (page fault) intercambiando información de la memoria con el almacenamiento secundario con velocidades de acceso varios órdenes de magnitud mayor. Debido a todo esto, la hiperpaginación puede llevar el sistema a un estado de degradación.

(d) I. El SO mantiene en memoria las siguientes estructuras:

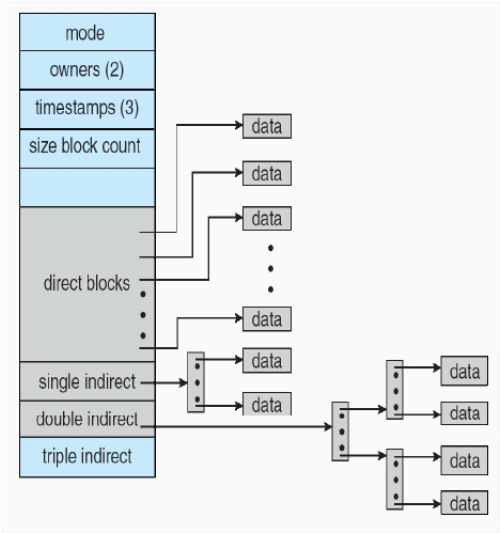
- La tabla de partición con los sistemas de archivos cargados que contiene información de cada Sistema de archivos montado
- La estructura de directorios de los accedidos últimamente
- La tabla de descriptores de archivos abiertos a nivel global del sistema
- La tabla de descriptores de archivo abiertos por proceso del sistema

II. Ubicación: Puntero al dispositivo y lugar donde reside el archivo.
Protección: Información de control para el acceso al archivo. Ej: Qué usuarios pueden acceder y en qué forma.
Nombre: un nombre simbólico que permite identificar el archivo a los usuarios. Pueden existir más de un archivo con el mismo nombre en un sistema de archivos jerárquico (directorios).

III. En la asignación indexada se mantiene una tabla en donde cada entrada referencia a un bloque de datos.
Los bloques son accedidos directamente a través del bloque de indexación (index block). Este bloque ocupa espacio y se intenta que se lo más pequeño posible, pero esto limita la cantidad de bloques y por lo tanto el tamaño máximo de archivo.



Una posible alternativa para poder solucionar esta limitación es la indexación en varios niveles, donde algunos índices hacen referencia a bloques directos y otros a bloques de indexación. En UNIX los bloques de control de archivos pueden tener hasta tres nivel de indexación lo que permite representar archivos muy grandes.



Problema 2 (33 pts)

Se tiene un sistema de archivos que utiliza una estrategia indexada simple como muestra la siguiente estructura de datos:

```
1   const MAX_BLOQUES = 16777216; // 2 ^ 24
2   type bloque = array [0..4095] of byte;
3   type entradaDir = Record
4       usado: bit; // entrada usada o no (1 bit)
5       nombre: array [0..22] of char; // nombre del elemento (23 bytes)
6       inicio: integer; // direccion de comienzo (4 bytes)
7       tipo: {ARCHIVO, DIRECTORIO}; // tipo de elemento (1 bit)
8       tam: integer; // tamanio del archivo en bytes (4 bytes)
9       reservado: array[0..5] of bit;// espacio reservado x el sistema (6 bits)
10  end; // 32 bytes
11
12  type fat = Array [0..(MAX_BLOQUES - 1)] of -2..(MAX_BLOQUES - 1);
13  type disco = Array [0..(MAX_BLOQUES-1)] of sector;
14  type mapaBits = Array [0..MAX_BLOQUES-1] of bit;
15
16  type ubicacionEntrada = Record
17      entrada: entradaDir;
18      bloque: integer;
19      indice: integer; // Indice de la entrada de directorio dentro del bloque
20  end;
21
22  var F: fat;
23      D: disk;
24      MB: mapaBits;
```

Notas generales:

- Las variables F, D y MB son globales.
- La variable F siempre se encuentra cargada en memoria con el contenido de la FAT. No se debe modelar la persistencia en disco de la FAT que además se sabe que se encuentra fuera del rango 0..MAX_BLOQUES.
- Archivos y directorios son alojados en D usando la estructura de la FAT. El directorio raíz comienza en el sector número 0 de D.
- En la FAT el valor -1 representa “fin de archivodirectorio” y el -2 “bloque libre”.

Se dispone de los siguientes procedimientos:

- `Procedure leerBlq(numBloque: int, var buff: bloque) : boolean`
  Lee de disco el bloque `numBloque`, pasado como parámetro, y carga el contenido leído en el parámetro de salida `buff`. Retorna `true` en caso de que la operación se ejecute con éxito y `false` en caso contrario.
- `Procedure escrBlq(numBloque: int, buff: bloque) : boolean`
  Escribe en el bloque `numBloque`, pasado como parámetro, la información que se encuentra en el parámetro `buff`. Retorna `true` en caso de que la operación se ejecute con éxito y `false` en caso contrario.

Se pide:

(a) (5 pts) Calcule la cantidad de archivos máxima que soporta un sistema de archivos con estas estructuras de datos.

Solución: Como los archivos vacios no ocupan un bloque de disco, vamos a maximizar la cantidad de bloques que puede tener un directorio y que en cada bloque tenga todas sus entradas archivos.

Tenemos que la estuctura entradaDir ocupa 32 bytes, y un sector de disco es de 4096 bytes, por lo que por sector tenemos: 4096/32 = 128 entradas.

Por lo que por cada bloque de disco podemos tener 128 archivos, y un directorio puede tener (si es el raiz) un maximo de MAX_BLOQUES, por lo que entonces la cantidad maxima de archivos que es de: MAX_BLOQUES * 128.

(b) (9 pts) Implementar una función que busque un archivo en un directorio. Retorna la ubicacionEntrada del archivo correspondiente en caso que exista, y null en caso contrario.
`function buscarArchivo(entradaDir directorio,char* nomArchivo):ubicacionEntrada`

Solución:

```
  function buscarArchivo(entradaDir directorio, char* nombreArchivo)
var buff = [0..127] of entradaDir;
if (!directorio.usado || directorio.tipo != DIRECTORIO
    || nombreArchivo == "") {
  return null;
}

var encontre = false;
var iterBloque = directorio.inicio;
var ultimoSector = false;
var i = 0;
while (!encontre and !ultimoSector) {
  if (!leerBlq(iterBloque, buff)){
    return null;
  }
  i = 0;
  while (i < 128 AND !encontre) {
    encontre = buff[i].usado && buff[i].tipo == ARCHIVO
        && buff[i].nombre == nombreArchivo;
    if (!encontre){
      i++;
    }
  }
  if (!encontre){
    if (F[iterBloque] == -1) {
      ultimoSector = true;
    } else {
      iterBloque = F[iterBloque];
    }
```

```
    }
  }
  if (encontre){
    var ubic:ubicacionEntrada;
    ubic.entrada = buff[i];
    ubic.indice = i;
    ubic.bloque = iterBloque;
    return ubic;
  } else {
    return null;
  }
}
```

(c) (8 pts) Implementar una función que agregue un bloque a un directorio pasado como parámetro. Retorna `true` en caso de éxito y `false` en caso contrario.
`function addBlock(entradaDir directorio): boolean`

Solución:

```
function addBlock(entradaDir directorio)
  if (!directorio.usado || directorio.tipo != DIRECTORIO) {
    return null;
  }

  var bloqueLibre = -1;
  var i = 0;
  while ((bloqueLibre == -1) AND (i < MAX_BLOQUES -1)) {
    if (!mapaBits[i]) { // 0 esta libre, 1 esta usado
      bloqueLibre = i;
    } else {
      i++;
    }
  }
  if (bloqueLibre == -1) { // No hay mas lugar en disco
    return false;
  }
  // Itero en la fat hasta encontrar el final,
  // se asume consistencia de file system
  var iter = directorio.inicio;
  while (!F[iter] == -1) {
    iter = F[iter];
  }
  // formateo el bloque
  var buff = [0..127] of entradaDir;
  if (!leerBlq(bloqueLibre, buff)){
    return false;
  }
  for (i = 0; i < 128 ; i++) {
```

```
      buff[i].usado = false;
    }
    if (!escrBlq(bloqueLibre, buff)){
      return false;
    }
    F[iter] = bloqueLibre; // Agrego el bloque al final
    F[bloqueLibre] = -1;   // de la lista de sectores
    mapaBits[bloqueLibre] = 1; // Marco como utilizado el bloque
    return true;
  }
```

(d) (11 pts) Implementar una función que mueva un archivo de un directorio a otro directorio, retorna `true` si la operacion fue exitosa y `false` en caso contrario.
`function moverArchivo(char* nomArchivo, entradaDir dirOrigen, entradaDir dirDestino): boolean`

Solución:

```
function moverArchivo(char* nombreArchivo, entradaDir dirOrigen,
entradaDir dirDestino)
  var archivoOrigen = buscarArchivo(dirOrigen, nombreArchivo);
  if (archivoOrigen == null) {
      // no encuentro la entrada directorio del archivo origen
    return false;
  }
  var archivoDestino = buscarArchivo(dirDestino, nombreArchivo);
  if (archivoDestino != null) {
      // Valido que el archivo NO exista en el destino
    return false;
  }

  var entradaLibre = null;
  var buff = array[0..127] of entradaDir;
  var encontre = false;
  var bloque = dirDestino.inicio;
  var prevBloque = dirDestino.inicio;
  var iter = 0;
  // busco lugar en el directorio destino
  while (!encontre AND (bloque != -1)) {
    if (!leerBlq(bloque, buff)) {
      return false;
    }
    iter = 0;
    while (buff[iter].usado AND iter < 128) { // Busco entrada
      iter++;
    }
    if (iter >= 128) {
      prevBloque = bloque;
```

```
          bloque = F[bloque]; // Itero en la fat
        } else {
          encontre = true;
        }
      }

      // Busco un nuevo bloque para mi archivo
      if (!encontre) {
        if (!addBlock(dirDestino)){
          return false;
        }
        bloque = F[prevBloque]; // obtengo el indice del nuevo bloque
        if (!leerBlq(bloque, buff)) {
          return false;
        }
        iter = 0;
      }

      // Escribo los cambios en disco,
      // agregando el archivo destino
      buff[iter].tipo = entradaArchivo.entrada.tipo;
      buff[iter].nombre = entradaArchivo.entrada.nombre;
      buff[iter].inicio = entradaArchivo.entrada.inicio;
      buff[iter].tam = entradaArchivo.entrada.tam;
      buff[iter].reservado = entradaArchivo.entrada.reservado;
      buff[iter].usado = true;
      if (!escrBlq(dirDestino.inicio, buff)) {
        return false;
      }

      // Escribo los cambios en disco, liberando
     la entrada directorio que usaba el archivo
      if (!leerBlq(archivoOrigen.bloque, buff)) {
        return false;
      }
      buff[archivoOrigen.indice].usado = false;

      if (!escrBlq(archivoOrigen.bloque, buff)) {
        return false;
      }
    }
```

Problema 3 (37 pts)

En un museo quieren instaurar un sistema mediante mailbox para organizar las visitas. El museo permite dos tipos diferentes de visitas: autónomas con audio-guías o guiadas. El visitante espcifíca en la recepción que tipo de visita realizará.

Para las visitas guiadas el museo cuenta con 3 guías, que harán los recorridos con al menos 5 visitantes. También se dispone de 30 audio-guías.

Tanto los visitantes que están esperando que quede un guía disponible o llegar a ser 5 visitantes para

recorrido guiado como los que están esperando por una audio-guía esperarán en una sala de espera.

El uso de las audio-guías será en orden de llegada, pero en caso de espera tendrán prioridades los visitantes que no han estado antes en el museo.

Al finalizar el recorrido los visitantes con audio-guías devolverán las mismas en la recepción y los otros pagarían una propina al guía que les hizo la recorrida.

La atención en recepción será por orden de llegada, sin importar la acción que se vaya a realizar.

Se dispone de los siguientes procedimientos auxiliares:

- `Datos(bool R_G, bool nuevo)` : Procedimiento de los visitantes para saber si quiere hacer la recorrida guiada o no y si es la primera visita al museo o no.
- `Tomar_audio_guia()` : Procedimiento de los visitantes para tomar un audio-guía.
- `Devolver_audio_guia()` : Procedimiento de los visitantes para devolver un audio-guía.
- `Calcular_propina()` : Procedimiento de los visitantes para determinar el monto que pagará de propina.
- `Recorrer_con_audio_guia()` : Procedimiento de los visitantes para realizar el recorrido con audio-guía.
- `Recorrer_con_guía()` : Procedimiento de los guías para realizar el recorrido.

Implemente utilizando mailbox el sistema incluyendo los procedimientos **visitante**, **guía** y **recepción**. No se pueden utilizar otros procedimientos o funciones.

Solución:

```
recepcion: mailbox of int
mutex_rec: mailbox of None
soy_nuevo_mb: mailbox of bool
audio_prio: mailbox of None
audio_no_prio: mailbox of None
rec_cli: mailbox of None
guia_id: mailbox of int
cant_guia: mailbox of int
turno_guia: mailbox of None
propinas[3]: array of mailbox of int
termino_guia[3]: array of mailbox of None
listo_para_recorrido[3]: array of mailbox of None

PIDO_AUDIO = 0
DEVUELVO_AUDIO = 1
PIDO_GUIA = 2

def recepcion:
    cant_audio = 30
    cant_prio = 0
    cant_no_prio = 0
    while (True):
        tramite = receive(recepcion)
        switch (tramite):
            case PIDO_AUDIO:
                es_nuevo = receive(soy_nuevo_mb)
```

```
                if cant_audio == 0:
                    if es_nuevo:
                        cant_prio++
                    else:
                        cant_no_prio++
                else:
                    cant_audio--
                    if es_nuevo:
                        send (audio_prio, None)
                    else:
                        send (audio_no_prio, None)
                break

            case DEVUELVO_AUDIO:
                if cant_prio > 0:
                    cant_prio--
                    send (audio_prio, None)
                elif cant_no_prio > 0:
                    cant_no_prio--
                    send (audio_no_prio, None)
                else:
                    cant_audio++
                break

            case PIDO_GUIA:
                cant = receive (cant_guia)
                cant2 = cant + 1
                send (cant_guia, cant2)
                if (cant2 == 5):
                    send (turno_guia, None)
                break
            send (rec_cli, None)


def visitante:
    Datos ( hago_recorrido, soy_nuevo)
    if hago_recorrido:
        receive (mutex_rec)
        send (recepcion, PIDO_GUIA)
        receive (cli_rec)
        send (mutex_rec, None)
        id_guia = receive (espero_guia)
        send (listo_para_recorrido[id_guia], None)
        receive (termino_guia[id_guia])
        propina = calcular_propina()
        send (propinas[id_guia], propina)

    else:
        receive (mutex_rec)
```

```
        send (recepcion, PIDO_AUDIO)
        send (soy_nuevo_mb, soy_nuevo)
        receive (rec_cli)
        send (mutex_rec, None)

        if soy_nuevo:
            receive (audio_prio)
        else:
            receive (audio_no_prio)

        tomar_audio_guia ()
        recorrer_con_audio ()
        receive (mutex_rec)
        send (recepcion, DEVUELVO_AUDIO)
        devolver_audio_guia ()
        receive (rec_cli)
        send (mutex_rec, None)

def guia:
    id = receive (guia_id)
    while (True):
        receive (turno_guia)
        cant = receive (cant_guia)
        send (cant_guia, 0)
        for i in range(cant):
            send (espero_guia, id)
        for i in range(cant):
            receive (listo_para_recorrido[id])
        recorrer_con_guia()
        for i in range(cant):
            send (termino_guia[id], None)
        propina = 0
        for i in range(cant):
            propina += receive (propinas[id])


def main:
    send (mutex_rec, None)
    for i in range(3):
        send (guia_id, i)
```