Sistemas Operativos

Trabajo Práctico 2019

Trenes

Profesor: Ing. Mostovoi Alejandro

Integrantes del Grupo:
Fernandez Gonzalo
Palacios Facundo
Pignatta Mauro
Pirrota Ezequiel
Ugobono Alejandro
Wiñar Mariano

Índice

ESTACIONES

TRENES

INTERFAZ ESTACIONES

INTERFAZ TRENES

Funciones de Estación

Encontraremos aquí todas aquellas funciones necesarias para la gestión, proceso y administración de todas las estaciones, en sus facetas tanto como tipo servidor hacia los clientes, es decir la comunicaciones entre servidor/cliente (proceso estación/proceso tren), como así también todas aquellas que tienen que ver con al operatoria y administración de cada estación

1- Estructuras de Datos:

1.1- Estructura Estaciones

```
typedef struct
{
        int ID;
        char nombre[max_nombre_est];
        int distancia;
        int online;
        int nCliente;
        TREN tren[MAX_TREN];
}ESTACION;
```

1.2- Estructura Nodo Trenes

```
typedef struct nodo
{
        struct nodo * sig;
        TREN * tren;
        int prioridad;
}ST_NODO_TRENES;
```

1.3- Variables global de Punteros a Tren, Nodo Cola Prioridad Menor y Mayor

```
pthread_mutex_t lock;
TREN * anden ;
ST_NODO_TRENES * ColaPrioridadMenor ;
ST_NODO_TRENES * ColaPrioridadMayor ;
```

1.4- Variable Global Puntero tipo File

```
pthread_mutex_t log_lock;
FILE * logEstacion;
```

1.5- Variable global del servidor de estaciones con su máximo de estaciones

```
int serverEst[MAX_ESTACION];
```

1.6- Variable global estaciones con el tipo de estructura de estaciones y su cantidad máxima definida:

```
ESTACION estaciones[MAX_ESTACION];
```

1.7- Variable global de un entero que se utiliza para las posiciones
int miPos;
1.8- Variable global de un entero para el tren viajando
int trenEnViaje;

2- Funciones y Aplicaciones:

2.1 int ObtenerDatosMiEstacion(char * nomArchivo, ESTACION est[]):
Aplicación:
Abre el archivo de configuración pasado como argumento y lo guarda, en la posición del vector de estaciones que corresponda,
Parámetros:
* nomArchivo es el nombre del archivo.
ESTACION est es el vector de estaciones,
Retorna:
Devuelve la posición en la que se encuentra tu estación.

2.2 void ObtenerOtrasEstaciones(ESTACION est[],int miPos):
Aplicación:
Abre el resto de los archivos de configuración, y obtiene los datos de las demás estaciones.
Parámetros:
ESTACION est es el vector de estaciones,
int miPos guarda la posición de la estación en el vector.

2.3 int registrarTren(ESTACION *estacion, char * mensaje):
Aplicación:
Copia los datos del tren en la estación en caso que haya lugar disponible,
Parámetros:
ESTACION * estacion puntero a la estación,
char * mensaje contiene los datos para registrar el tren,
Retorna:
Devuelve 1 si el tren se registro correctamente y 0 en caso de que no se haya registrado.

2.4 int buscarTrenes(TREN trenes[] ,int posTrenes[]):
Aplicación:
Devuelve un vector con las posiciones del vector de trenes en las que se encuentran,
Parámetros:
TREN trenes es el vector de trenes,
int posTrenes indica la posición de los trenes, encontrados,
Retorna:
Devuelve la cantidad.

2.5 int BuscarTrenPorID(ESTACION estacion, int idTren):
Aplicación:
Busca un Tren en el vector de trenes,
Parámetros:
ESTACION estacion variable del tipo ESTACION,
int idTren es el numero de tren a buscar,
Retorna:
Devuelve la posición en la que se encuentra el tren en el vector de trenes de la estación, o -1 si no se encuentra.

2.6 void ConexionServer():
Aplicación:
Para el hilo que se encarga de la conexión servidor-cliente.

2.7 int buscarEstacionPorNombre(char * mensaje):
Aplicación:
Busca a la estación por el nombre,
Parámetros:
char * mensaje donde tiene el nombre de la estación a buscar,
Retorna:
Devuelve la pos si la encuentra o -1 si no la encuentra.

2.8 int mensajeListadoEstDisp(char * mensaje):
Aplicación:
Copia las estaciones disponibles para viajar,
Parámetros:
Mensaje puntero a char donde se van a copiar las estaciones disponibles,
Retorna:
Devuelve la cantidad de estaciones que están disponibles para viajar.

2.9 int mostrarTrenesMigrados(char * mensaje):
Aplicación:
Copia los trenes que migraron al mensaje*,
Parámetros:
Mensaje puntero a char donde se van a copiar los trenes,
Retorna:
Devuelve la cantidad de trenes migrados.

2.10 int elegirTren():
Aplicación:
Pide al usuario que ingrese el tren que quiere que viaje,
Retorna:
Posición del tren elegido en el vector o -1 en caso de que el tren elegido no sea valido.

2.11 int elegirEstDestino():
Aplicación:
Pide al usuario que ingrese la estación donde quiere viajar*,
Retorna:
Posición la estación elegida o -1 en caso de que no sea valida.

2.12 int calcularTiempoDeViaje(int posEstacionDestino):
Aplicación:
Calcula el tiempo basándose en la distancia entre una estación y otra,
Parámetros:
posEstacionDestino La posición en el vector de estación a la cual se quiere dirigir el tren,
Retorna:
Tiempo de viaje.

2.13 void prepararEnvioTren(char *mensaje , TREN * tren):
Aplicación:
Prepara el mensaje para enviar un tren de una estación a otra,
Parámetros:
* mensaje puntero a char copia el mensaje a enviar,
* tren Para saber que tren hay que enviar.

2.14 ST_NODO_TRENES * crearNuevoNodo(TREN * tren):
Aplicación:
Crea nodo de tren,
Parámetros:
* puntero tren de estructura TREN,
Retorna:
Devuelve el nodo del nuevo nodo creado.

2.15 void encolarTren(TREN * tren, ST_NODO_TRENES ** cola):
Aplicación:
Suma trenes a la cola de trenes,
Parámetros:
* puntero tren de estructura TREN,
** cola puntero a puntero del Nodo Trenes.

2.16 TREN * asignarAnden(ST_NODO_TRENES ** cola):
Aplicación:
Asigna Anden al tren,
Parámetros:
** cola puntero a puntero del Nodo Trenes,
Retorna:
Devuelve el puntero del tren afectado.

2.17 TREN * eliminarNodoPrioridad(ST_NODO_TRENES ** cola):
Aplicación:
Elimina Nodos según la prioridades,
Parámetros:
** cola puntero a puntero del Nodo Trenes,
Retorna:
Devuelve el puntero del tren afectado.

2.18 TREN * eliminarNodoTrenSegunID(int IDTren, ST_NODO_TRENES ** cola):
Aplicación:
Elimina Nodo del tren por ID,
Parámetros:
int IDTren para identificar al tren afectado,
** cola puntero a puntero del Nodo Trenes,
Retorna:
Devuelve el puntero del tren afectado.

2.19 int subirPrioridadTrenes(ST_NODO_TRENES * cola):
Aplicación:
Sube la prioridad a los trenes por tiempo de su espera en la cola,
Parámetros:
* cola puntero al Nodo Trenes,
Retorna:
Devuelve el entero con la sumatoria que le corresponda según la vuelta de prioridades.

2.20 void CambiarDeColaTrenes(ST_NODO_TRENES ** cola_Menor, ST_NODO_TRENES ** cola_Mayor, int cantNodos):
Aplicación:
Cambia de cola menor prioridad a la Cola Mayor prioridad,
Parámetros:
** cola Menor puntero a la Colas de Menor prioridad,
** cola Mayor puntero a la Colas de Mayor prioridad,
int cantNodos lleva la cantidad de nodos tipo entero.

2.21 void NuevoTrenAnden(TREN ** anden, ST_NODO_TRENES ** cola_Menor, ST_NODO_TRENES ** cola_Mayor):
Aplicación:
Nuevo tren para anden,
Parámetros:
TREN ** anden puntero de puntero anden de trenes,
** cola Menor puntero a la Colas de Menor prioridad,
** cola Mayor puntero a la Colas de Mayor prioridad.

2.22 FILE * crearLogEstacion(char * nombreEstacion):

Aplicación:

Crea el Archivo tipo txt según nombre de la estación,

Parámetros:

char * nombreEstacion puntero del nombre de la estación,

Retorna:

Puntero tipo FILE al archivo txt de la estación.

2.23 void * llenarLog(TREN * tren , FILE * logEstacion):

Aplicación:

Carga los datos en el archivo txt de la estación afectada,

Parámetros:

TREN * tren puntero estructura tipo TREN con los datos del tren afectado,
FILE * logEstacion puntero tipo FILE al archivo txt de la estación afectada.

2.24 void avisarEstaciones(int posEstacionDestino, int tipoAviso);

Aplicación:

Avisa a las estaciones que hay un tren en viaje o que llego al destino.

Parámetros:

int posEstacionDestino posición de la estación a la que viaja el tren
int tipoAviso es 1 para avisar que viaja, 2 para avisar que llego.

Funciones de Tren

Realizan el recorrido de cada proceso llamado Tren a lo largo de todo el programa, pasa por toda la rama de comunicaciones entre el proceso tren y los procesos estación, con la característica de servidor/cliente, como así también las operaciones del mismo y los informes de su estado y condición.

1- Estructuras de Datos:

1.1- Estructura Trenes

```
typedef struct{
    int ID;
    int combustible;
    char modelo[10];
    char estOrigen[max_nombre_est];
    char estDestino[max_nombre_est];
    int tiempoRestante;
    int migrado;
    int nCliente;
}TREN;
```

2- Funciones y Aplicaciones:

2.1- TREN inicializarTren(char *arch):

Aplicación:
Inicializa el Tren con todas las variables,
Parámetros:
char * arch puntero a
Retorna:
Devuelve la variable tipo Tren con toda la info cargada.
int tipoAviso es 1 para avisar que viaja, 2 para avisar que llego.

2,2- void partir(TREN tren):

Aplicación:
Dibuja el Tren y resta el tiempo,
Parámetros:
TREN tren variable tipo tren.

2.3- void cargarCombustible(int *combustible):

Aplicación:
Carga Combustible al tren que lo solicite,
Parámetros:
int * combustible variable para indicar cantidad.

2.4- void armarMensajeRegistrarse(TREN tren,char *mensaje):
Aplicación:
Arma en una variable el mensaje con los datos del tren,
Parámetros:
TREN tren una variable del tipo TREN,
char * mensaje la variable del mensaje a enviar.

2.5- void armarMensajeEstadoDelTren(TREN t, char * mensaje):
Aplicación:
Arma el mensaje con el estado del tren,
Parámetros:
TREN tren una variable del tipo TREN,
char * mensaje la variable del mensaje a enviar.

2.6- void armarMensajeExit(TREN tren, char * mensaje):
Aplicación:
Arma el mensaje para salir del programa,
Parámetros:
TREN tren una variable del tipo TREN,
char * mensaje la variable del mensaje a enviar.

2.7- void armarMensajePartir(TREN tren ,char * mensaje):
Aplicación:
Arma el mensaje para indicarle a la estación que quiere partir el tren,
Parámetros:
TREN tren una variable del tipo TREN,
char * mensaje la variable del mensaje a enviar.

Interfaz de Estación

Es la conexión funcional entre los dos procesos, en este caso los Trenes como Cliente y las Estaciones como Servidores de esos Clientes, es decir una relación servidor/cliente, proporcionándonos así la comunicación entre ambos, el envío de datos e información necesario para el funcionamiento total del programa, además de las funciones relacionadas al dibujo en pantalla a través de la interfaz gráfica con ncurses

1- Estructuras de Datos:

1.1- Interfaz gráfica

```
typedef struct {
   WINDOW *pAppFrame;
   WINDOW *pLogFrame;
   WINDOW *pLogWindow;
   WINDOW *pCmdFrame;
   WINDOW *pCmdWindow;
   WINDOW *pRegFrame;
   WINDOW *pRegWindow;
} ST_APP_WINDOW;
```

1.2- Variable global para la interfaz grafica

```
ST_APP_WINDOW pWin;
```

1.3- Estructura del tipo enum para los colores a utilizar

```
typedef enum {RED=1, GREEN, BLUE, WHITE, YELLOW, CYAN,} COLOUR;
```

2- Funciones y Aplicaciones:

2.1- void initUserInterface(ST_APP_WINDOW *):

Aplicación:

Crea las ventanas de la app. Asocia colores con las ventanas, Crea en pantalla un grupo de ventanas con la siguiente estructura:
Log: muestra mensajes,
Cmd: permite el ingreso de comandos al usuario.

Parámetros:

ST_APP_WINDOW * puntero a estructura que contiene las ventanas.

2.2- void drawUserInterface(ST_APP_WINDOW *):

Aplicación:

Dibuja el Marco de la App, Ventana Log, Ventana Cmd, Ventana Reg,

Parámetros:

ST_APP_WINDOW * puntero a estructura que contiene las ventanas.

2.3- void printWindowTitle(WINDOW *pWin, const char * message):
Aplicación:
Toma el Titulo a imprimir,
Parámetros:
* pWin Variable global para la interfaz grafica,
* message es la Estación y su numero como titulo.

2.4- void printLog(ST_APP_WINDOW *pWindow, const char * message, COLOUR colour):
Aplicación:
Imprime en la pantalla tipo Log en la parte superior,
Parámetros:
* pWindow puntero a la estructura que contiene las ventanas,
* message contiene el mensaje a imprimir,
colour de la estructura tipo enum con los colores a utilizar.

2.5- void printRegistro(ST_APP_WINDOW *pWindow, const char *message, COLOUR colour):
Aplicación:
Imprime en la parte derecha de la ventana si se conecto el tren, se utiliza en funcEstaciones.c,
Parámetros:
* pWindow puntero a la estructura que contiene las ventanas,
* message contiene el mensaje a imprimir,
COLOUR colour es de la estructura tipo enum con los colores a utilizar.

2.6- int printEstadoTrenes(ST_APP_WINDOW *pWin , TREN trenes[]):
Aplicación:
Imprime el Estado completo del Tren con todas sus variables,
Parámetros:
* pWin Variable global para la interfaz grafica,
TREN trenes es el vector de trenes,
Retorna:
Un entero.

2.7- void unInitUserInterface(ST_APP_WINDOW *):
Aplicación:
Inicia el usuario de la interfaz grafica,
Parámetros:
* variable global para la interfaz grafica.

2.8- void printEstadoEstaciones(ST_APP_WINDOW *pWin, ESTACION est[]):
Aplicación:
Imprime el Estado completo de la Estación,
Parámetros:
* pWin Variable global para la interfaz grafica,
ESTACION est es el vector de estaciones.

2.9- void printHelp(ST_APP_WINDOW *pAppWin):
Aplicación:
Imprime la Ayuda del programa,
Parámetros:
* pAppWin representa la Variable global para la interfaz grafica.

2.10- void clearWindow(WINDOW *pWin):
Aplicación:
Limpia la ventana de comandos y hace un refresco de pantalla,
Parámetros:
* pWin variable global, ventana de comandos,
Retorna:
ERR_OK.

2.11- void clearCmdWindow(WINDOW *pWin):
Aplicación:
Limpia la ventana de comandos,
Parámetros:
* pWin variable global, ventana de comandos,
Retorna:
ERR_OK.

2.12- void InterfazGrafica():
Aplicación:
Para el hilo que se encarga de la interfaz grafica.

Interfaz de Tren

Es la conexión funcional entre los dos procesos, en este caso los Trenes como Cliente y las Estaciones como Servidores de esos Clientes, es decir una relación servidor/cliente, proporcionándonos así la comunicación entre ambos, el envío de datos e información necesario para el funcionamiento total del programa, además de las funciones relacionadas al dibujo en pantalla a través de la interfaz gráfica con ncurses

1- Estructuras de Datos:

1.1- Interfaz Gráfica

```
typedef struct {
   WINDOW *pAppFrame;
   WINDOW *pLogFrame;
   WINDOW *pLogWindow;
   WINDOW *pCmdFrame;
   WINDOW *pCmdWindow;
} ST_APP_WINDOW;
```

1.2 Estructura del tipo enum para los colores a utilizar

```
typedef enum {RED = 1, GREEN, BLUE, WHITE, YELLOW, CYAN,} COLOUR;
```

2- Funciones y Aplicaciones:

2.1- void initUserInterface(ST_APP_WINDOW *)

Aplicación:

Crea las ventanas de la app. Asocia colores con las ventanas, Crea en pantalla un grupo de ventanas con la siguiente estructura:

Log: muestra mensajes,
Cmd: permite el ingreso de comandos al usuario.

Parámetros:

ST_APP_WINDOW * puntero a estructura que contiene las ventanas.

2.2 void drawUserInterface(ST_APP_WINDOW *):

Aplicación:

Dibuja el Marco de la App, Ventana Log, Ventana Cmd, Ventana Reg,

Parámetros:

ST_APP_WINDOW * puntero a estructura que contiene las ventanas.

2.3 void printWindowTitle(WINDOW *pWin, const char * message):
Aplicación:
Imprime en la pantalla tipo Log en la parte superior,
Parámetros:
* pWindow puntero a la estructura que contiene las ventanas,
* message contiene el mensaje a imprimir,

2.4 void printMessage(ST_APP_WINDOW *pWindow, const char * message, COLOUR colour):
Aplicación:
Imprime mensaje, param * pWindow puntero a la estructura que contiene las ventanas,
Parámetros:
* message contiene el mensaje a imprimir,
const char * colour de la estructura tipo enum con los colores a utilizar.

2.5 void unInitUserInterface(ST_APP_WINDOW *):
Aplicación:
Inicia el usuario de la interfaz grafica,
Parámetros:
* variable global para la interfaz grafica.

2.6 void clearCmdWindow(WINDOW *pWin):
Aplicación:
Limpia la ventana de comandos,
Parámetros:
* pWin variable global, ventana de comandos,
Retorna:
ERR_OK.

2.7 void clearLogWindow(WINDOW *pWin):
Aplicación:
Limpia la ventana de comandos y hace un refresco de pantalla,
Limpia la pantalla del Log,
Parámetros:
* pWin variable global, ventana de comandos,
Retorna:
ERR_OK.

2.8 void printHelp(ST_APP_WINDOW *pAppWin):
Aplicación:
Imprime la Ayuda del programa,
Parámetros:
* pAppWin representa la Variable global para la interfaz grafica.

2.9 void imprimirAndenAsignado(ST_APP_WINDOW *pWin):
Aplicación:
Imprime un mensaje que se le a asignado el anden al tren,
Parámetros:
* pAppWin representa la Variable global para la interfaz grafica.

2,10 void DibujarTrenViajando(WINDOW *pLogWindow, int * tiempoRestante):
Aplicación:
Muestra el Tren viajando,
Parámetros:
* pLogWindow representa la Variable global,
* puntero al tiempo restante del tren para actualizarlo.

2.11 void salirDelPrograma(TREN tren, int client, ST_APP_WINDOW * pWin):
Aplicación:
Salir del programa,
Parámetros:
TREN tren estructura de datos tren,
int client numero de cliente servidor,
* pAppWin representa la Variable global para la interfaz grafica.

2.12 void InterfazGrafica(void * argumentos):
Aplicación:
Para el hilo que se encarga de la interfaz grafica,
Parámetros:
* argumentos.

Conexiones

Crea y conecta a todos los participantes del programa, en este caso los procesos tren y estación.

1- Funciones y Aplicaciones:

1.1 char * FormatearNombre(char * Palabra):
Aplicación:
Organiza la palabra ingresada
Parámetros:
* palabra es la palabra ingresada en el comando,
Retorna:
El nombre formateado, primera letra en Mayúscula y las demás en minúsculas

1.2 void obtenerDatosRed(char* IP, int *Puerto, char * confRed):
Aplicación:
Obtiene IP y Puerto de la red de servicios,
Parámetros:
char * IP es la dirección de IP que identifica la interfaz de red,
char * Puerto es el puerto de comunicación,
char * confRed el archivo de configuracion de red,
Retorna:
* red puntero al archivo de configuración de red.

1.3 int CrearSocketServer(char * confRed):
Aplicación:
Crea el Socket de comunicación tipo Servidor,
Parámetros:
* confRed es el puntero del archivo de configuración del Servidor,
Retorna:
Un int server formateado para la conexión.

1.4 int CrearSocketCliente(char * confRed):
Aplicación:
Crea el Socket de comunicación tipo Cliente,
Parámetros:
* confRed es el puntero al archivo de configuración del Cliente,
Retorna:
Un int cliente formateado para la conexión.

1.5 int conectarEstacion(char * confRed):

Aplicación:

Crea la Conexión de la Estación,

Parámetros:

* confRed es el puntero del archivo de configuración del Servidor,

Retorna:

Un int cliente formateado para la conexión.

1.6 void obtenerConfRed(char * nombreEstacion , char * archConfigRed):

Aplicación:

Obtiene el nombre del archivo de conf de Red según el nombre de la estación a conectarse,

Parámetros:

* nombreEstacion puntero al argumento con el nombre de la estación a conectarse,
* archConfigRed puntero a char donde se va a guardar el nombre del archivo de conf de red.

Casos de Prueba

Presentamos situaciones o casos para analizar, comprobar y probar el programa, su funcionamiento y la capacidad de analizar y plantear soluciones a cada instancia

1- Registro en Estación

DESCRIPCIÓN	Tren inicia y se registra en la estación.
PASOS	1 - Iniciar el proceso Tren. 2 - Ingresar el comando «registrarse»
RESULTADO ESPERADO	Mensaje en la ventana «Log» del tren: “Te has registrado correctamente”. Mensaje en la ventana «Registro» de la estación: “Un Tren se ha registrado”.

2- Registro Duplicado

DESCRIPCIÓN	Tren que intenta registrarse nuevamente cuando ya estaba registrado.
PASOS	1- Iniciar el proceso Tren. 2- Ingresar el comando «registrarse» 3- Repetir paso 2.
RESULTADO ESPERADO	Mensaje en la ventana «Log» : “Ya te has registrado”.

3- Estado del Tren

DESCRIPCIÓN	Informa el estado del tren completo con todas sus características
PASOS	1- Iniciar el proceso Tren. 2- Ingresar el comando «estado»
RESULTADO ESPERADO	Mensaje en la ventana «Log» mostrando: - ID. - Combustible. - Modelo. - Estación Actual. - Estación Destino . - Tiempo de viaje restante.

4- Partir sin haberse registrado

DESCRIPCIÓN	Solicitar anden cuando esta libre desde el tren
PASOS	1- Iniciar el proceso Tren. 2- Ingresar el comando «registrarse» 3- Ingresar el comando «anden» 4- Ingresar la estacion donde quiere partir
RESULTADO ESPERADO	Mensaje en la ventana de «Log»: “Se te asigno el anden, usa partir para viajar.”

5- Solicitar anden cuando esta libre desde el tren

DESCRIPCIÓN	Solicitar el anden a la estacion cuando se encuentra desocupado.
PASOS	1- Iniciar el proceso Tren. 2- Ingresar el comando «registrarse». 3- Ingresar el comando «anden». 4- Ingresar la estación donde quiere partir.
RESULTADO ESPERADO	Mensaje en la ventana de «Log»: “Se te asigno el anden, usa partir para viajar.”

6- Solicitar anden cuando esta ocupado desde el tren

DESCRIPCIÓN	Solicitar el anden a la estacion cuando se encuentra ocupado.
PASOS	1- Iniciar el proceso Tren. 2- Ingresar el comando «registrarse». 3- Ingresar el comando «anden». 4- Ingresar la estacion donde quiere partir.
RESULTADO ESPERADO	Mensaje en la ventana de «Log»: “Has entrado en la cola. Esperando a que se desocupe el anden..”

7- Buscar estaciones

DESCRIPCIÓN	Uso del comando «buscar est» para que las estaciones se puedan comunicarse entre ellas.
PASOS	1 Iniciar el proceso Estacion. 2 Ingresar el comando «buscar est».
RESULTADO ESPERADO	Mensaje en la ventana «Log» con un listado de las estaciones que se encontraron conectadas.

8- Estado de las Estaciones

DESCRIPCIÓN	Informa el estado de las estaciones.
PASOS	1 Iniciar el proceso Estacion. 2 Ingresar el comando «estado est»
RESULTADO ESPERADO	Mensaje en la ventana «Log» mostrando: -Nombre. -ID. -Distancia. -Si se encuentra conectada o no.

9- Estado de los trenes desde la estación

DESCRIPCIÓN	Informa el estado de todos los trenes que se encuentran registrados en la estacion.
PASOS	1 Iniciar el proceso Estación. 2 Iniciar al menos un proceso Tren. 3 Registrar los trenes iniciados en la estación. 4 Ingresar el comando «estado tren» desde la estación.
RESULTADO ESPERADO	Mensaje en la ventana «Log» mostrando los datos de los trenes registrados.

10- Proceso Tren Migrado

DESCRIPCIÓN	El tren al partir de una estación finaliza como proceso, y al llegar a su estación destino comienza un nuevo proceso tren.
PASOS	1. Ejecutar 2 procesos Estación 2. Ingresar, en ambas estaciones, el comando «buscar est». 3. Ejecutar proceso Tren. 4. Desde el Tren, Ingresar el comando «registrarse». 5. Desde el Tren, Ingresar el comando «anden». 6. Ingresar la estacion donde dirigirse. 7. Ingresar el comando «partir». 8. Esperar a que el tren migre a la estacion destino. 9. En una nueva Terminal, Ejecutar «htop».
RESULTADO ESPERADO	Dentro de la tabla de procesos de htop, debe aparecer el proceso ./tren <ID del tren> <nombre de la estacion>.

11- Solicitar anden para un tren migrado desde la estación

DESCRIPCIÓN	Ya con un tren migrado en una estacion, hacer que solicite anden para poder partir nuevamente.
PASOS	1 Migrar un tren a una estacion. 2 Desde la estacion, Ingresar el comando «anden». 3 Ingresar el ID del tren que quiera que solicite anden. 4 Ingresar la estacion al cual quiera partir.
RESULTADO ESPERADO	Mensaje en la ventana «Log» indicando si el tren encuentra en el anden o en la cola de espera.

12- Hacer que un tren migrado parta

DESCRIPCIÓN	Ya con un tren migrado en una estacion que haya solicitado anden previamente, hacer que parta.
PASOS	1 Migrar un tren a una estacion. (Caso de prueba 10) 2 Solicitar anden para el tren migrado. (Caso de prueba 11) 3 Ingresar el comando «partir».
RESULTADO ESPERADO	Mensaje en la ventana «Log» mostrando el dibujo del tren viajando.

Diagrama UML

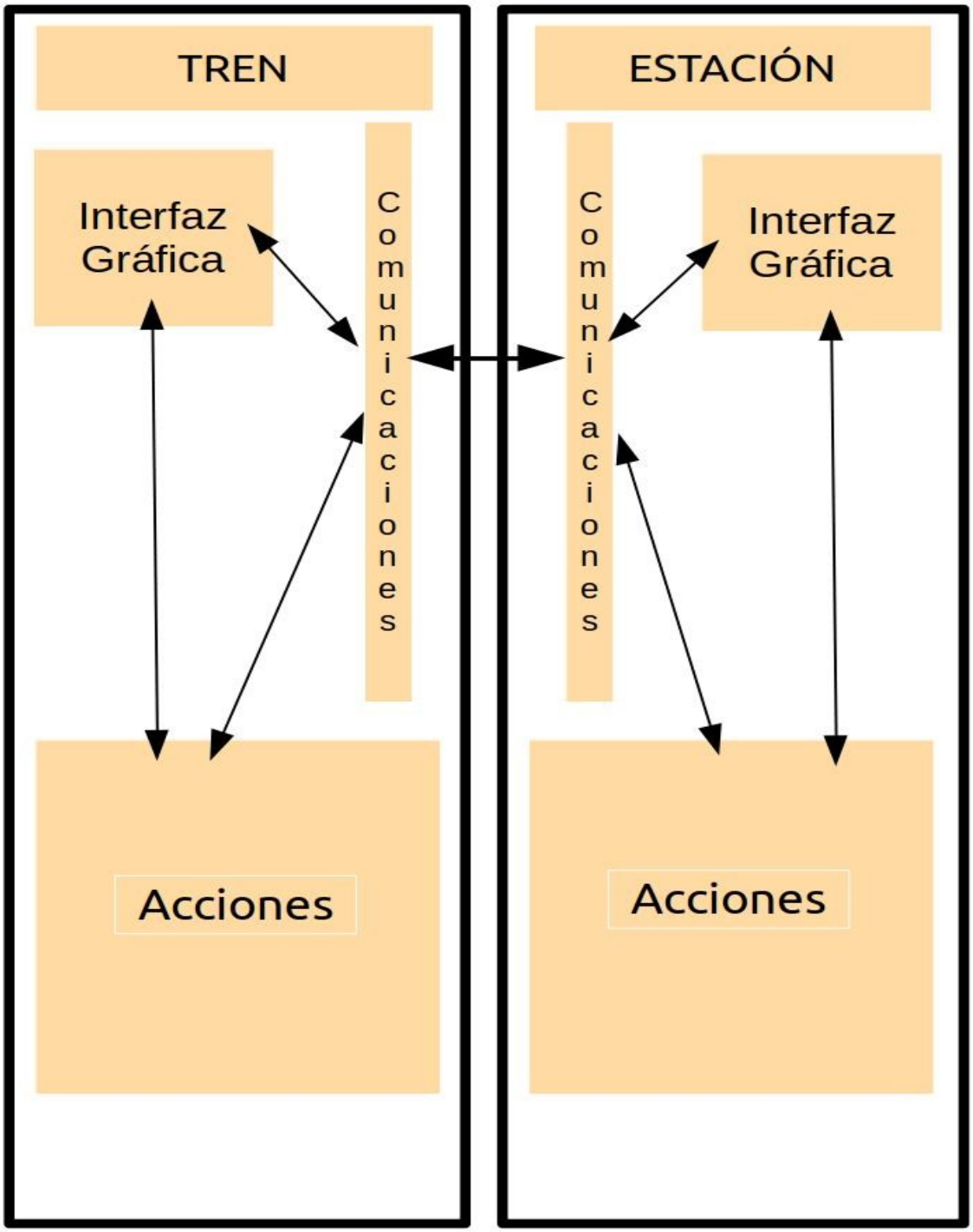

Diagrama PBS