UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Ingeniería en Sistemas de Información

PedidOS Ya!

Te llevamos el TP a casa

Cátedra de Sistemas Operativos

Trabajo práctico Cuatrimestral

-2C2020 -
Versión 1.4

Índice

Versión de Cambios

v1.1 (06/09/2020) Primera Revisión

- *Se modificó la redacción del apartado de la administración de los mensajes del módulo Comanda.*
- *Se eliminó el campo "Esquema memoria" de la configuración de la comanda que estaba demás.*
- *Se agregó el mensaje 'Plato Listo' en el apartado de la administración de los mensajes del módulo App.*
- *Se modificó valores erróneos en el config del módulo App, se agrega la IP y puerto del módulo Restaurante y se arreglo el ejemplo de configuración de la App.*
- *Se agregaron los mensajes Obtener Receta y Terminar Pedido en el apartado de lineamientos del módulo Sindicato.*
- *Se modificó el mensaje Plato Listo dentro del módulo Restaurante.*
- *Se agrego los mensajes Obtener Receta en el Anexo I.*
- *El mensaje Consultar Pedido de la App no devolverá el repartidor.*
- *Se modificó el archivo de configuración del módulo Comanda.*
- *Se agregan campos faltantes en el config del módulo Restaurante*

v1.2 (26/09/2020) Segunda Revisión

- *Agregamos archivo de logs en los resultados de los comandos de la consola del Cliente*
- *Definimos el handshake inicial del módulo Cliente*
- *Modificamos el archivo de configuración del Cliente*
- *Cambiamos las operaciones que debe realizar el mensaje Guardar Plato del módulo Comanda*
- *Explayamos más las especificaciones del Planeamiento e Implementación del módulo Sindicato*
- *Modificamos el retorno de Consultar Pedido en el módulo App*
- *Modificaciones y ajustes del Anexo 1*
    - *Retorno de Obtener Pedido*
    - *Retorno de Obtener Restaurante*
    - *Retorno de Consultar Plato*
    - *Descripción de Añadir Plato*
    - *Parámetro de Confirmar Pedido*
    - *Retorno de Consultar Pedido*

v1.3 (10/10/2020) Tercera Revisión

- *Ajustamos la descripción de cómo debe funcionar el Guardar Pedido en el Sindicato.*
- *Eliminamos la necesidad de que el repartidor tenga que agregar un 5% de recargo al pedido*
- *Modificación de la recomendación respecto a comunicaciones bidireccionales*
- *Se eliminó el puerto el PUERTO_APP del config del cliente por ser un error en la versión previa.*
- *Se decidió colocar opcional el campo Cliente en Seleccionar Restaurante*

v1.4 (31/10/2020) Cuarta Revisión

- *Agregamos una aclaración en Seleccionar Restaurante en el módulo App.*
- *Arreglamos un typo en la descripción de Guardar Plato en Comanda.*
- *Agregamos un retardo de ciclo de CPU en Restaurante.*
- *Agregamos un typo en un path de Sindicato en la sección de Files.*
- *Agregamos Cantidad_Pedidos en la Info del Restaurante en Sindicato.*
- *Aclaramos el caso de Restaurante en el mensaje Consultar Pedido en el Anexo I.*
- *Aclaramos que el alpha también se utiliza para el caso de HRRN.*

Objetivos y Normas de resolución

Objetivos del Trabajo Práctico

Mediante la realización de este trabajo se espera que el alumno:

- Adquiera conceptos prácticos del uso de las distintas herramientas de programación e interfaces (APIs) que brindan los sistemas operativos.
- Entienda aspectos del diseño de un sistema operativo.
- Afirme diversos conceptos teóricos de la materia mediante la implementación práctica de algunos de ellos.
- Se familiarice con técnicas de programación de sistemas, como el empleo de makefiles, archivos de configuración y archivos de log.
- Conozca con grado de detalle la operatoria de Linux mediante la utilización de un lenguaje de programación de relativamente bajo nivel como C.

Características

- Modalidad: grupal (5 integrantes ± 0) y obligatorio
- Tiempo estimado para su desarrollo: 90 días
- Fecha de comienzo: 29/8
- Fecha de primera entrega: 28/11 (a confirmar)
- Fecha de segunda entrega: 5/12 (a confirmar)
- Fecha de tercera entrega: 19/12 (a confirmar)
- Lugar de corrección: Discord y Google Meet

Evaluación del Trabajo Práctico

El trabajo práctico consta de una evaluación en 2 etapas.

La primera etapa consistirá en las pruebas de los programas desarrollados en el laboratorio. Las pruebas del trabajo práctico se subirán oportunamente y con suficiente tiempo para que los alumnos puedan evaluarlas con antelación. Queda aclarado que para que un trabajo práctico sea considerado evaluable, el mismo debe proporcionar registros de su funcionamiento de la forma más clara posible.

La segunda etapa se dará en caso de aprobada la primera y constará de un coloquio, con el objetivo de afianzar los conocimientos adquiridos durante el desarrollo del trabajo práctico y terminar de definir la nota de cada uno de los integrantes del grupo, por lo que se recomienda que la carga de trabajo se distribuya de la manera más equitativa posible.

Cabe aclarar que el trabajo equitativo no asegura la aprobación de la totalidad de los integrantes, sino que cada uno tendrá que defender y explicar tanto teórica como prácticamente lo desarrollado y aprendido a lo largo de la cursada.

La defensa del trabajo práctico (o coloquio) consta de la relación de lo visto durante la teoría con lo implementado. De esta manera, una implementación que contradiga a lo visto en clase o lo escrito en el documento ***es motivo de desaprobación del trabajo práctico***.

Deployment y Testing del Trabajo Práctico

Al tratarse de una plataforma distribuida, los procesos involucrados podrán ser ejecutados en diversas computadoras. La cantidad de computadoras involucradas y la distribución de los diversos procesos en estas será definida en cada uno de los tests de la evaluación y **es posible cambiar la misma en el momento de la evaluación**. Es responsabilidad del grupo automatizar el despliegue de los diversos procesos con sus correspondientes archivos de configuración para cada uno de los diversos tests a evaluar.

Todo esto estará detallado en el documento de pruebas que se publicará cercano a la fecha de Entrega Final. Archivos y programas de ejemplo se pueden encontrar en el repositorio de la cátedra.

Finalmente, recordar la existencia de las Normas del Trabajo Práctico donde se especifican todos los lineamientos de cómo se desarrollará la materia durante el cuatrimestre.

Aclaraciones

Debido al fin académico del trabajo práctico, los conceptos reflejados son, en general, versiones simplificadas o alteradas de los componentes reales de hardware y de sistemas operativos modernos, a fin de resaltar aspectos de diseño.

Invitamos a los alumnos a leer las notas y comentarios al respecto que haya en el enunciado, reflexionar y discutir con sus compañeros, ayudantes y docentes al respecto.

Definición del Trabajo Práctico

Esta sección se compone de una introducción y definición de carácter global sobre el trabajo práctico. Posteriormente se explicarán por separado cada uno de los distintos módulos que lo componen, pudiéndose encontrar los siguientes títulos:

- Abstract 's: Todo contenido que se encuentre dentro de un título llamado “Abstract” tiene como fin la explicación teórica de un concepto que no es afín a la cátedra pero ayudará a la comprensión e implementación del módulo en cuestión. De esta manera, se remarca que dicho contenido ***no se debe implementar ya que es una definición teórica de un tema***.
- Lineamiento e Implementación: Todos los títulos que contengan este nombre representarán la definición de lo que deberá realizar el módulo y cómo deberá ser implementado. La no inclusión de alguno de los puntos especificados en este título puede conllevar a la desaprobación del trabajo práctico.

Cabe destacar que en ciertos puntos de este enunciado se explicaran exactamente como deben ser las funcionalidades a desarrollar mientras que en otros no se definirá específicamente ,quedando su implementación a decisión y definición del equipo.. Se recomienda en estos casos siempre consultar a sus ayudantes o en el foro de github.

¿Qué es el trabajo práctico y cómo empezamos?

El objetivo del trabajo práctico consiste en desarrollar una solución que permita la simulación de un sistema distribuido que utiliza el concepto de pedidos de comida. Para el desarrollo del mismo se decidió la creación de un sistema bajo la metodología Iterativa Incremental donde se solicitarán en una primera instancia la implementación de ciertos módulos para luego poder realizar una integración total con los restantes.

De esta manera, recomendamos seguir el lineamiento de los distintos puntos de control que se detallan al final de este documento para su desarrollo. Estos puntos están planificados y estructurados para que sean desarrollados a medida y en paralelo a los contenidos que se ven en la parte teórica de la cátedra. Cabe aclarar que esto es un lineamiento propuesto por la cátedra y no implica impedimento alguno para el alumno de realizar el desarrollo en otro orden diferente al especificado.

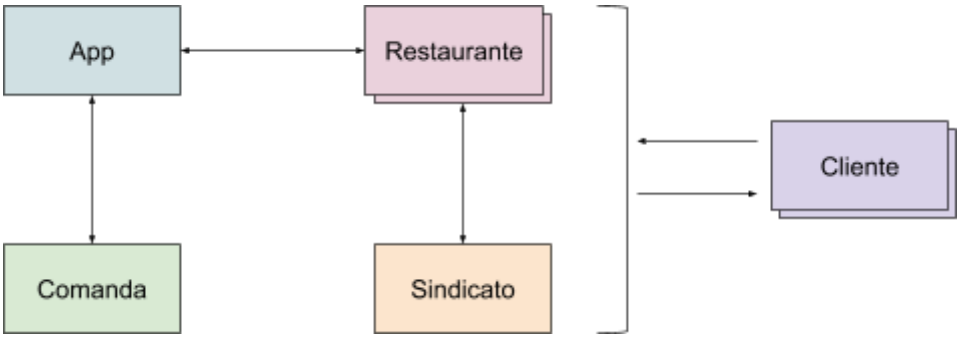
Los componentes incluidos dentro de la arquitectura del sistema deberán trabajar en conjunto para la planificación y ejecución de distintas operaciones, entre las que se encuentran, por ejemplo: leer y escribir valores. Las operaciones que conforman estos mensajes están asociadas y vinculadas a restaurantes y delivery.

Los componentes del sistema serán:

- Un proceso que será el punto de partida del sistema (Cliente).
- Un proceso planificador de repartidores (App).
- Un proceso planificador de platos y pedidos (Restaurante).
- Un proceso filesystem que se encargará de mantener los archivos en tiempo real (Sindicato).
- Un proceso memoria que se encargará de almacenar los pedidos activos (Comanda).

Arquitectura del sistema

El trabajo práctico funcionará bajo la siguiente arquitectura de conexiones entre módulos.



Como dijimos anteriormente, el trabajo práctico funciona bajo la temática de la solicitud y administración de pedidos de delivery. En nuestro trabajo práctico tomaremos los delivery's de dos maneras diferentes: uno a través de una app centralizada de restaurantes y otro solicitándolo directamente a estos últimos. De esta manera, los módulos dibujados en la imagen anterior tomarán la siguiente funcionalidad:

- Módulo Cliente: Módulo encargado de consultar restaurantes y recetas y de realizar, confirmar y consultar pedidos. Cada módulo cliente podrá realizar un pedido y finalizarlo cuando el mismo concluya. Dependiendo la modalidad que se utilice se podrá comunicar con la APP, actuando de intermediario entre él y el Restaurante, o directamente con este último.
- Módulo App: Módulo encargado de centralizar, administrar y planificar distintos pedidos entre los repartidores que haya en el sistema. A su vez, funcionará de vínculo entre los clientes y los restaurantes que estén registrados en la Aplicación.
- Módulo Comanda: Se encargará de mantener y almacenar los distintos pedidos que se encuentren en la aplicación en determinado momento.
- Módulo Restaurante: Se encargará de centralizar, administrar y planificar los distintos pedidos que sean de dicho restaurante entre los distintos empleados/cocineros que disponga.
- Módulo Sindicato: Se encargará de almacenar y contener toda la información de los distintos restaurantes así como de los pedidos que se le hagan a ellos.

Para el desarrollo del trabajo práctico recomendamos la siguiente distribución de tareas:

- Módulo Cliente: 0,5 personas
- Módulo App y Restaurante: 2 personas
- Módulo Comanda: 1,5 personas
- Módulo Sindicato: 1 persona

Cabe aclarar que esta es una distribución de trabajo estimativa y tentativa que creemos que debe tener el trabajo práctico para que todos los alumnos puedan trabajar equitativamente y aprender conceptos de la materia. Dado que el trabajo funciona bajo el concepto iterativo incremental

recomendamos modificar dichos números para que todos los integrantes del grupo participen en varios módulos y no estén sin trabajo hasta la segunda iteración.

Por último, se recuerda que *desarrollar únicamente el módulo cliente y/o temas de conectividad, serialización y sincronización es insuficiente para poder entender y aprender los distintos conceptos de la materia* ***por lo que será un motivo de desaprobación***.

Módulo Cliente

Abstract

Polimorfismo[1]

El polimorfismo dentro de nuestra profesión se refiere a la capacidad por la que distintos módulos, componentes, clases o productos puedan ser tratados indistintamente por otro, comprendiendo los mismo mensajes. De esta manera, los componentes intervinientes definen una interfaz o contrato sobre el cual obtendrán y responderán a los mensajes de manera homogénea.

Lineamiento e Implementacion

Será un proceso Multihilo que se encargará de la administración de uno o varios pedidos de un cliente específico. Este módulo funcionará en base a comandos escritos en consola que ejecutarán distintos mensajes a la APP o Restaurantes.

Por otro lado, funcionará como un intermediario que nos permita verificar (testear) los módulos Comanda y Sindicato. De esta manera, permitirá el envío del 100% de los mensajes de nuestro sistema.

Al momento de su inicialización el módulo se intentará conectar al IP/Puerto definido por archivo de configuración. En caso de que la conexión no sea exitosa, finalizará informando por pantalla dicho error.

Una vez conectado, este módulo aceptará cargar por consola los comandos indicados en el Anexo I. En los subsiguientes módulos se explicarán las funcionalidades que tendrán dichos mensajes en cada uno de ellos.

El módulo cliente podrá recibir la notificación de la actualización de un pedido haciendo que la comunicación funcione bidireccionalmente. Esto quiere decir que mientras un pedido ha sido creado/confirmado puede llegar una actualización de otro. Por esto, queda a libre elección del grupo la utilización de uno o más sockets adicionales a fin de que sea más simple la implementación de los mensajes asincrónicos. .

El objetivo del polimorfismo en nuestro trabajo práctico se debe a que los mensajes que cada módulo use serán polimórficos entre ellos. Esto quiere decir que las interfaces de los mensajes ***deben ser iguales***, es decir, recibirán lo mismo y retornar lo mismo. Por ejemplo, si el módulo A y el módulo B usan el mensaje “Crear pedido”, ambos deben recibir y retornar el/los mismos tipos de datos.

De esta manera, independientemente de a quien se conecte, el proceso Cliente siempre enviará y esperará recibir lo mismo frente a un mensaje específico. Toda la especificación de estos mensajes (API) se encuentra definida en Anexo I.

[1] Definición de polimorfismo

Toda actualización recibida o enviada deberá ser almacenada en un archivo de LOG con el formato que el grupo determine. Teniendo en cuenta que el cliente va a aceptar mensajes por consola de forma continua, estos logs no deberán ser mostrados por pantalla.

Por último, el módulo contará con 2 posiciones (X e Y), las cuales deberán informarse al módulo App y cuya utilización se verá detallada en los lineamientos de implementación del módulo App.

Para las conexiones con los distintos módulos, se deberá implementar un handshake en el que se informen los datos necesarios para reconocer al Cliente (Id y posición).

Archivo de Configuración

Campo	Tipo	Descripción
IP	[String]	El IP del servidor al cual se conectará.
PUERTO	[Numérico]	El puerto del servidor al cual se conectará.
ARCHIVO_LOG	[String]	Path del archivo de log donde se almacenará el log obligatorio.
POSICION_X	[Numérico]	Posición en X que deberá ser enviada al módulo App
POSICION_Y	[Numérico]	Posición en Y que deberá ser enviada al módulo App
ID_CLIENTE	[String]	Id único que representa al cliente.

Ejemplo de Archivo de Configuración

```
IP=127.0.0.1
PUERTO=5001
ARCHIVO_LOG=/utnso/logs/cliente.log
POSICION_X=1
POSICION_Y=2
ID_CLIENTE=Cliente1
```

Módulo App

Será el encargado de la administración centralizada de los pedidos a distintos Restaurantes. Para esto, planificará los mismos entre los distintos repartidores que disponga y los hará moverse, en un eje de dos coordenadas, para ir a buscarlos y entregarlos.

Lineamiento e implementación

Será un proceso Multihilo que tendrá las siguientes funciones:

1. Administrar los pedidos a los distintos Restaurantes planificandolos en los distintos Repartidores.
2. Ser el intermediario entre el cliente y un restaurante específico.
3. Informar, guardar y obtener dichos pedidos que almacenará el módulo Comanda.

De esta manera, este proceso tendrá dos sub módulos específicos:

1. Administración de Mensajes
2. Planificación de Pedidos

Inicialización

Al momento de inicializar el proceso se realizarán las siguientes operaciones:

1. Se conectará al módulo Comanda.
2. Abrirá un puerto de escucha para que los distintos Restaurantes se puedan conectar a él.
3. Obtendrá de su archivo de configuración los distintos Repartidores con sus frecuencias y tiempos de descanso.

Administración de Mensajes

El módulo APP reconocerá los siguientes mensajes que le lleguen por sockets:

1. Consultar Restaurantes
2. Seleccionar Restaurante
3. Consultar Platos
4. Crear Pedido
5. Añadir Plato
6. Plato Listo
7. Confirmar Pedido
8. Consultar Pedido

Consultar Restaurantes

Se deberán obtener todos los nombres de los restaurantes conectados a dicho módulo y retornarlos. En caso que no exista ninguno conectado se deberá retornar el Restaurante con nombre “Resto Default”. Este último restaurante nos permitirá la prueba de dicho módulo sin la conexión de restaurantes al mismo.

Seleccionar Restaurante

Este mensaje se encargará de asociar, de manera lógica, a un Cliente con un Restaurante. Esto permitirá que en la posterior creación o consulta de un pedido se sepa sobre qué restaurante se está realizando. Tener en cuenta que se podrá volver a seleccionar un restaurante para dirigir los mensajes posteriores al nuevo restaurante.

Consultar Platos

Este mensaje permitirá la consulta de las recetas del Restaurante seleccionado. De esta manera, el módulo APP servirá de "pasamano" del mensaje al modulo Restaurante requerido. Cabe aclarar que en caso que no exista ningún Restaurante conectado se retornarán la lista de platos por default informado en el archivo de configuración.

Crear Pedido

Este mensaje permitirá la creación de un Pedido sobre el restaurante seleccionado. La funcionalidad de este mensaje se descompone en los siguientes pasos:

1. Enviar el mensaje al Restaurante seleccionado y obtener el ID de mensaje que este ultimo genero. En caso que no exista ningún restaurante conectado se generará un ID arbitrario que defina al pedido (tener en cuenta que debe ser ÚNICO).
2. Enviar el mensaje Guardar Pedido al Módulo Comanda informando la creación de un nuevo pedido. Si este módulo informa un error deberá informarse que el pedido está rechazado (la casuística para que esto pase se explica en dicho módulo).
3. Retorna el ID del pedido al Cliente que solicitó el pedido.

Añadir Plato

Este mensaje permitirá añadir un plato a un determinado pedido. Para esto se recibirá tanto el plato como el ID de Pedido destino. La funcionalidad de este mensaje se descompone en los siguientes pasos:

1. Enviar al Restaurante dueño de dicho Pedido el mensaje que nos llegó. En caso que se trate del "Restaurante Default" pasar al punto 3.
2. Si el Restaurante dueño de dicho pedido retorna un error, se debe informar el rechazo al Cliente solicitado
3. Informar al Módulo Comanda la adhesión de un nuevo Plato a dicho Pedido. Si el módulo informa un error deberá informarse el rechazo del mismo (la casuística para que esto pase se explica en dicho módulo).
4. Retornar OK al cliente que lo solicitó.

Plato Listo

Cuando llegue este mensaje desde el Restaurante, la App deberá reenviarselo a la comanda para que sume uno a la cantidad lista de dicho plato. La funcionalidad de este mensaje se descompone en los siguientes pasos:

1. Enviar el mensaje Plato Listo a la Comanda.

2. Se deberá ejecutar el mensaje Obtener Pedido a la Comanda con el fin de comparar la cantidad con la cantidad lista. En el caso de que sean iguales, significará que el repartidor ya puede retirar el pedido del Restaurante, desencadenando los eventos necesarios en la planificación.

Confirmar Pedido

Este mensaje permitirá confirmar un pedido creado previamente. Para esto se recibirá el ID de pedido destino. La funcionalidad de este mensaje se descompone en los siguientes pasos:

1. Obtener el Pedido desde el Módulo Comanda. Enviar al Restaurante dueño de dicho Pedido el mensaje que nos llegó. En caso que se trate del "Restaurante Default", pasar al punto 3.
2. Si el restaurante dueño de dicho pedido retorna un error, se debe informar el rechazo al cliente solicitado.
3. Generar el PCB (Pedido Control Block) del Pedido en cuestión y dejarlo en el ciclo de planificación (que se explicará a continuación en el módulo de Planificación de Pedidos).
4. Informar al Módulo Comanda la actualización del estado del pedido.
5. Informar al Módulo Cliente que su pedido fue confirmado

Cabe aclarar que el PCB deberá contener solamente la información necesaria para planificar el pedido, como por ejemplo: instrucciones necesarias, repartidor seleccionado, posición actual del repartidor, etc. No es correcto almacenar todo el estado actual del pedido dado que la información del mismo debe guardarse y consultarse al Módulo Comanda.

Consultar Pedido

Este mensaje permitirá obtener el estado de un pedido previamente confirmado. Para esto se obtendrán los datos desde el Módulo Comanda y se retornarán:

1. El Restaurante
2. El estado del pedido
3. La lista de platos con sus cantidades

Planificación de Pedidos

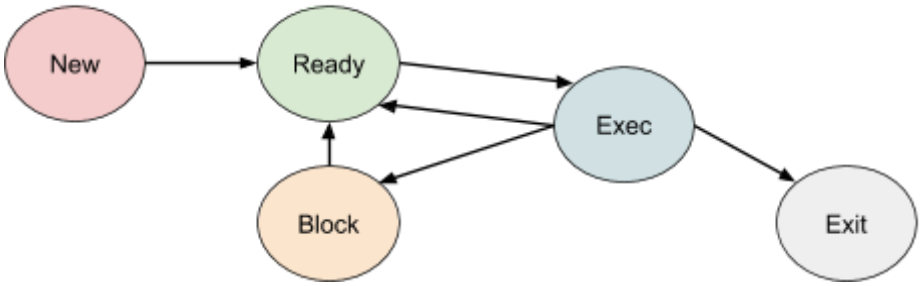
En el momento que un pedido sea confirmado y se haya creado su PCB, el mismo entrará en este submódulo para su planificación. Esta planificación implica:

1. Verificar si existen Repartidores libres para planificarlo. En caso que exista más de uno se seleccionará al más cercano. Esto representará el Planificador a largo plazo de nuestro módulo APP.
2. Una vez asignado a nuestro Repartidor, el Pedido entrará en estado Ready y estará listo para que nuestro Planificador a Corto Plazo lo administre.

Grado de Multiprogramación

Solo se permitirá que un repartidor atienda simultáneamente un pedido. De esta manera, nuestro grado de multiprogramación estará definido por la cantidad de Repartidores que el sistema disponga.

Diagrama de estados



Todo Pedido al crear su PCB se incorporará en la cola New. A medida que se vayan asignando Repartidores a dichos pedidos los mismos pasarán a estado Ready. El objetivo del repartidor será moverse hasta el restaurante destino y, una vez que todos los platos estén listos, moverse a la posición del Cliente.

En caso que existan varios repartidores libres al momento de la creación de un pedido se deberá asignar al repartidor más cercano al restaurante del pedido. En caso contrario, si existen varios pedidos en New y solo un repartidor, siempre se deberá tomar el pedido por su orden de llegada.

Dado que nuestra aplicación tiene algunas limitaciones, se definirá por archivo de configuración el máximo de repartidores que podremos administrar simultáneamente (grado de multiprocesamiento); lo que definirá cuántos de nuestros repartidores podrán estar en el estado Exec al mismo tiempo.

Cada ciclo de CPU utilizado por cualquier repartidor se verá afectado por un retardo configurado desde el archivo de configuración.

Cada repartidor tendrá una frecuencia de cansancio que definirá que cada X ciclos de CPU deberá descansar un tiempo definido. Cuando un repartidor alcance dicho estado se deberá bloquear (pasar a estado bloqueado) por un tiempo de descanso. Ambos valores están definidos por el archivo de configuración.

Cuando el Repartidor alcance la posición del Restaurante, deberá verificar contra el Módulo Comanda si todos los platos de dicho Pedido están listos. En caso que así sea, se deberá mover directamente a la posición del Cliente. En caso contrario, se deberá pasar a estado Bloqueado hasta que se informe que dichos Platos están OK.

Cabe aclarar que, si el pedido es sobre el “Restaurante Default” se dará por sentado que al momento de llegar al mismo el pedido ya se encuentra listo.

Una vez que el Repartidor llegue a la posición del Cliente se dará por concluido el pedido, realizando los siguientes pasos:

1. Informar al Módulo Comanda la finalización de dicho pedido.
2. Informar al Cliente la entrega del pedido.
3. Pasar el PCB asociado Exit.

4. Marcar al Repartidor como “Libre” para que pueda recibir otro pedido.

Como se nombró en un paso anterior, cada vez que un Plato pase a estado listo llegará una notificación a la Aplicación por parte del Restaurante. Nuevamente esto quiere decir que la comunicación entre ambos módulos funcionará de manera bidireccional.

Algoritmos de Planificación

Por último y no menos importante, para la planificación de los distintos repartidores se implementaran los algoritmos FIFO, HRRN y SJF (sin desalojo). Todos los algoritmos serán aplicados como se vieron en la teoría.

Logs Mínimos y Obligatorios

Para permitir la verificación/validación del módulo se exigirá tener un archivo de log específico e independiente que contenga, como mínimo, la información indicada en esta sección. ***No se permite la exclusión de ninguno de los mismos*** (la falta de alguno puede implicar la desaprobación del grupo). Cada operación debe loguearse en una única línea indicando la misma y sus datos.

Las acciones a loguear en este archivo son:

1. Cambio de un repartidor de cola de planificación (indicando la razón del porqué).
2. Movimiento de un repartidor (indicando la ubicación a la que se movió).
3. Operación de asignación de pedido (indicando la ubicación del mismo y el Restaurante correspondiente).
4. Operación de entrega de pedido (indicando el repartidor involucrado).

Archivo de Configuración

Campo	Tipo	Descripción
IP_COMANDA	[String]	El IP del servidor del proceso Comanda.
PUERTO_COMANDA	[Numérico]	El puerto del servidor del proceso Comanda.
PUERTO_ESCUCHA	[Numérico]	Puerto en el cual la APP escuchara las distintas conexiones de los proceso Cliente y Restaurante
RETARDO_CICLO_CPU	[Numérico]	Tiempo en segundos para el retardo de la ejecución de cada ciclo de cpu.
GRADO_MULTIPROCESAMIENTO	[Numérico]	Cantidad de Repartidores que pueden estar en Exec simultáneamente
ALGORITMO_PLANIFICACION	[String]	El tipo de algoritmo de planificación que se va a utilizar (FIFO/HRRN/SJF-SD).
ALPHA	[Numérico]	El valor del alpha en caso de que el algoritmo utilice SJF y HRRN.

ESTIMACION_INICIAL	[Numérico]	El valor de la estimación inicial para SJF en caso de que aplique.
REPARTIDORES	[Lista]	Contiene la lista de las posiciones de cada repartidor.
FRECUENCIA_DE_DESCANSO	[Lista]	Contiene la lista del frecuencia de descanso de cada repartidor
TIEMPO_DE_DESCANSO	[Lista]	Contiene la lista de la cantidad de ciclos de cpu que debe descansar cada repartidor
ARCHIVO_LOG	[String]	Path del archivo de log donde se almacenará el log obligatorio.
PLATOS_DEFAULT	[Lista]	Listado de platos Default que poseerá el módulo
POSICION_REST_DEFAULT_X	[Numérico]	Posición en X del restaurante Default
POSICION_REST_DEFAULT_Y	[Numérico]	Posición en Y del restaurante Default

Ejemplo de Archivo de Configuración

```
IP_COMANDA=127.0.0.1
PUERTO_COMANDA=5001
PUERTO_ESCUCHA=5004
GRADO_DE_MULTIPROCESAMIENTO=1
ALGORITMO_DE_PLANIFICACION=FIFO
ALPHA=0,5
ESTIMACION_INICIAL=2
REPARTIDORES=[1|1,5|2,1|4]
FRECUENCIA_DE_DESCANSO=[8,5,10]
TIEMPO_DE_DESCANSO=[2,1,4]
ARCHIVO_LOG=/utnso/logs/app.log
PLATOS_DEFAULT=[Plato1, Plato2, Plato3]
POSICION_REST_DEFAULT_X=0
POSICION_REST_DEFAULT_Y=0
```

Módulo CoMAnda (Compartimiento de Memoria Auxiliar No Distribuido Académicamente)

El módulo de Comanda tiene la finalidad de guardar los pedidos de los distintos restaurantes desde que se crean hasta que se finalizan. Para mantener la información de los pedidos, se deben soportar las siguientes operaciones:

1. Guardar pedido
2. Guardar Plato
3. Obtener pedido
4. Confirmar pedido
5. Plato listo
6. Finalizar pedido

Estas operaciones van a ser ejecutadas por los Clientes y recibidas a través de los mismos o del proceso App. Todas las interfaces de dichos mensajes se encuentran definidas en el Anexo I.

Lineamiento e Implementación

Este módulo debe procesar múltiples operaciones en simultáneo, es decir, debe ser multihilo. ***Cualquier otra implementación, será motivo de desaprobación directa***.

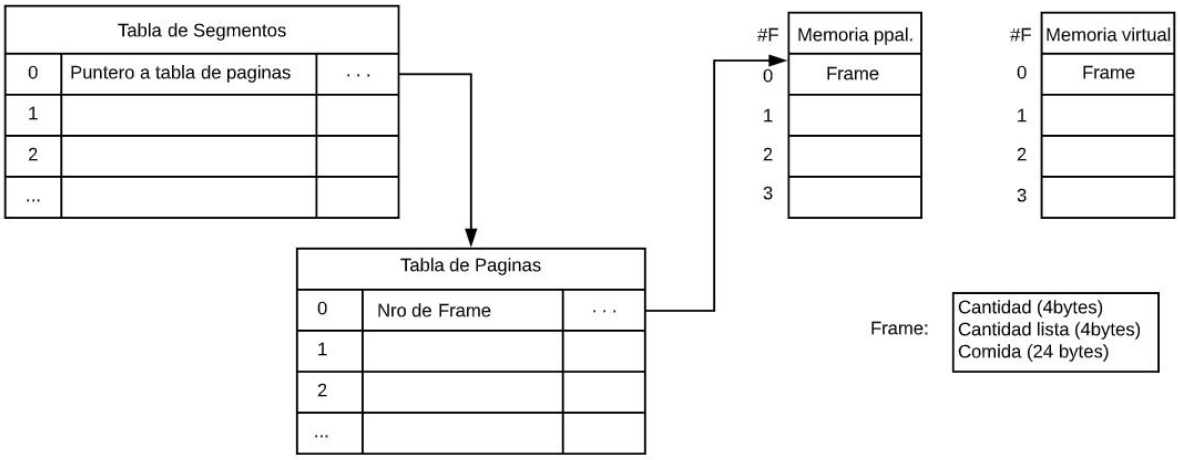
El esquema de administración de memoria a implementar será segmentación paginada con SWAP, contando con las siguientes estructuras administrativas:

- Memoria principal: Memoria contigua donde se aloja la información del detalle del pedido (explicado en los siguientes apartados). La misma debe ser reservada al iniciar el módulo y el tamaño viene por archivo de configuración. Debe contener la información que está dentro de las páginas.
- Tabla de segmentos: Estructura administrativa que contiene el listado de los segmentos y toda la información necesaria para el manejo de los mismos.
- Tabla de páginas: Esta será la estructura administrativa que contiene el listado de páginas para cada segmento. Cada segmento tendrá su correspondiente tabla de páginas. Es responsabilidad del grupo definir qué campos debe tener la tabla.
- Área de swap: Espacio de memoria secundaria, en disco, que puede ser direccionada como si fuera memoria real.

La información que guardaremos en memoria serán dos uint32_t, que hacen referencia a la cantidad total de dicha comida respecto del pedido, y a la cantidad lista (ya cocinada por el restaurante) respectivamente. También existirá un vector de tamaño estático de 24 bytes, donde se guardará el nombre de la comida, logrando así un tamaño de página fijo de 32 bytes.

Cada restaurante tiene su propia tabla de segmentos, donde cada segmento es un pedido. Este último, tiene un conjunto de páginas asociadas al pedido donde se guardaran los ítems del mismo.

Toda página debe existir inicialmente en SWAP moviéndose a Memoria solo en el caso que sea necesario, quedando dicha página tanto en memoria real como en SWAP hasta que sea reemplazada. El funcionamiento del mismo se corresponde al explicado en teoría.



En el caso de que no haya más espacio direccionable en la memoria, no se debe permitir realizar ninguna operación de adición.

Cuando la memoria real no tenga más espacio para guardar una nueva página se deberá realizar el proceso de reemplazo para enviar una página vieja a SWAP y traer la primera. Este proceso se realizará mediante el algoritmo LRU y Clock-mejorado. El reemplazo funcionará de manera global.

Administración de Mensajes

Guardar Pedido

Este mensaje cumplirá la función de inicializar todos los datos asociados a este pedido. Para esto recibirá:

1. El nombre del Restaurante
2. El ID del pedido

Al recibir esta información se realizarán las siguientes operaciones:

1. Verificar si existe la tabla de segmentos de dicho Restaurante. Si la tabla de segmentos no existe, deberá ser creada. En caso de no poder crearla, se deberá informar dicha situación.
2. Luego, se procederá a crear la tabla de páginas dentro del nuevo segmento.
3. Responder el mensaje indicando si se pudo realizar la operación correctamente (Ok/Fail).

Guardar Plato

Este mensaje cumplirá la función de agregar un nuevo plato a un determinado pedido. Para esto recibirá:

1. El nombre del Restaurante
2. El ID del pedido
3. El plato que se va a agregar

4. La cantidad del plato a agregar

Al recibir esta información se realizarán las siguientes operaciones:

1. Verificar si existe la tabla de segmentos de dicho Restaurante. En caso de no existir se deberá informar dicha situación.
2. Verificar que exista el segmento de dicho pedido dentro de la tabla de segmentos del Restaurante. En caso de no existir se deberá informar dicha situación.
3. Verificar si existe el plato dentro de la tabla de páginas del pedido. En caso contrario, se deberá asignar una nueva página al segmento.
4. En caso de que la página que corresponde al pedido no se encuentre cargada en memoria principal, se deberá cargar la misma desde swap (iniciando la elección de víctima de ser necesario). En caso contrario, proceder al paso número 5.
5. Se deberá agregar el plato y anexar la cantidad que se tiene que cocinar de dicho plato.
6. Responder el mensaje indicando si se pudo realizar la operación correctamente (Ok/Fail).

Obtener Pedido

Este mensaje cumplirá la función de obtener todos los datos de un determinado pedido. Para esto recibirá:

1. El nombre del Restaurante
2. El ID del pedido

Al recibir esta información se realizarán las siguientes operaciones:

1. Verificar si existe la tabla de segmentos de dicho Restaurante. En caso de no existir se deberá informar dicha situación.
2. Verificar que exista el segmento de dicho pedido dentro de la tabla de segmentos del Restaurante. En caso de no existir se deberá informar dicha situación.
3. En caso de que las páginas que corresponden al pedido no se encuentren cargadas en memoria principal, se deberán cargar las mismas desde swap (iniciando la elección de víctimas de ser necesario). En caso contrario, proceder al paso número 4.
4. Responder el mensaje indicando si se pudo realizar en conjunto con la información del pedido si correspondiera.

Confirmar Pedido

Este mensaje cumplirá la función de cambiar el estado de un determinado pedido. Para esto recibirá:

1. El nombre del Restaurante
2. El ID del pedido

Al recibir esta información se realizarán las siguientes operaciones:

1. Verificar si existe la tabla de segmentos de dicho Restaurante. En caso de no existir se deberá informar dicha situación.
2. Verificar que exista el segmento de dicho pedido dentro de la tabla de segmentos del Restaurante. En caso de no existir se deberá informar dicha situación.

3. Verificar que el pedido esté en estado “Pendiente”. En caso contrario se deberá informar dicha situación.
4. Cambiar el estado del pedido de “Pendiente” a “Confirmado”.
5. Responder el mensaje indicando si se pudo realizar la operación correctamente (Ok/Fail).

Plato Listo

Este mensaje cumplirá la función de aumentar en uno la cantidad de platos listos para un determinado pedido de un determinado Restaurante. Para esto recibirá:

1. El nombre del Restaurante.
2. El ID de Pedido.
3. El plato que está listo.

Al recibir esta información se realizarán las siguientes operaciones:

1. Verificar si existe la tabla de segmentos de dicho Restaurante. En caso de no existir se deberá informar dicha situación.
2. Verificar que exista el segmento de dicho pedido dentro de la tabla de segmentos del Restaurante. En caso de no existir se deberá informar dicha situación.
3. Verificar si existe el plato dentro de la tabla de páginas del pedido. En caso contrario, se deberá informar dicha situación.
4. En caso de que las páginas que corresponden al pedido no se encuentren cargadas en memoria principal, se deberán cargar las mismas desde swap (iniciando la elección de víctimas de ser necesario). En caso contrario, proceder al paso número 4.
5. Verificar que el pedido esté en estado “Confirmado”. En caso contrario se deberá informar dicha situación.
6. Se deberá aumentar en uno la cantidad lista de ese plato. En caso de que todos los platos estén listos, se deberá cambiar el estado del pedido a “Terminado”.
7. Responder el mensaje indicando si se pudo realizar la operación correctamente (Ok/Fail).

Finalizar Pedido

Este mensaje cumplirá la función de eliminar de la memoria los datos asociados al pedido. Para esto recibirá:

1. El nombre del Restaurante.
2. El ID de Pedido.

Al recibir esta información se realizarán las siguientes operaciones:

1. Verificar si existe la tabla de segmentos de dicho Restaurante. En caso de no existir se deberá informar dicha situación.
2. Verificar que exista el segmento de dicho pedido dentro de la tabla de segmentos del Restaurante. En caso de no existir se deberá informar dicha situación.
3. Se deberá proceder a eliminar las páginas correspondientes a dicho segmento.
4. Por último, se procederá a eliminar el segmento correspondiente.
5. Responder el mensaje indicando si se pudo realizar la operación correctamente (Ok/Fail).

Logs Mínimos y Obligatorios

Para permitir la verificación/validación del módulo se exigirá tener un archivo de log específico e independiente que contenga, como mínimo, la información indicada en esta sección. ***No se permite la exclusión de ninguno de los mismos*** (la falta de alguno puede implicar la desaprobación del grupo). Cada operación debe loguearse en una única línea indicando la misma y sus datos.

Las acciones a loguear en este archivo son:

1. Almacenado de un mensaje dentro de la memoria (indicando posición de inicio de su partición).
2. Eliminado de una partición de memoria (indicado la posición de inicio de la misma).
3. Comienzo de reemplazo de página.
4. Víctima seleccionada para reemplazo.

Archivo de Configuración

Campo	Tipo	Descripción
PUERTO_ESCUCHA	[Numérico]	Puerto en el cual el proceso Comanda escuchará las conexiones de la App
TAMANIO_MEMORIA	[Numérico]	Tamaño en bytes de la memoria real
TAMANO_SWAP	[Numérico]	Tamaño en bytes de la memoria secundaria
ALGORITMO_REEMPLAZO	[String]	Algoritmo de reemplazo de páginas [LRU/CLOCK_MEJ]
ARCHIVO_LOG	[String]	Path del archivo de log donde se almacenará el log obligatorio.

Ejemplo de Archivo de Configuración

```
PUERTO_ESCUCHA=5001
TAMANIO_MEMORIA=1024
TAMANIO_SWAP=2048
ALGORITMO_REEMPLAZO=LRU
ARCHIVO_LOG=/utnso/logs/comanda.log
```

Módulo Restaurante

Este módulo se encargará de la administración y planificación de platos de los distintos pedidos dentro de un Restaurante. Cada uno de ellos guardará y obtendrá sus datos por medio del módulo Sindicato. El mismo soportará los siguientes mensajes:

1. Consultar Platos
2. Crear Pedido
3. Añadir Plato
4. Confirmar Pedido
5. Consultar Pedido

Lineamiento e implementación

Será un proceso multihilo **(*cualquier otra implementación, será motivo de desaprobación directa)*** que se encargará de:

1. Creación y administración de pedidos.
2. Planificación de los platos de diversos pedidos entre los cocineros disponibles
3. Administración de los distintos Hornos (Entradas/Salidas) disponibles.

De esta manera, este proceso tendrá dos sub módulos específicos:

1. Administración de Mensajes
2. Planificación de Platos

Inicialización

Al momento de inicializarse de deberán realizar las siguientes operaciones:

1. Obtención de metadata del restaurante
2. Creación de las distintas colas de planificación.

Metadata del restaurante

Se ejecutará el mensaje Obtener Restaurante al módulo Sindicato obteniendo todos los datos del mismo, siendo algunos de ellos: Cantidad de cocineros y sus afinidades, posición del restaurante en el mapa, recetas disponibles con sus precios y la cantidad de hornos. A su vez, deberá retornar la cantidad de pedidos que ya disponga para no pisar los ID con los nuevos generados.

Creación de colas de planificación

En base a los Hornos y los cocineros con sus afinidades que se obtuvieron, se deberán crear las distintas colas de ready y de entrada salida.

De esta manera, si tengo dos cocineros que tienen como afinidad a Milanesa se deberá crear dos colas de Ready en las cuales todas las milanesas irán a dicha cola y los demás platos irán a otra. Un cocinero con afinidades solo podrá cocinar aquellas recetas a las cuales es afín.

Por otro lado, durante la ejecución de un plato puede darse que se requiera enviarlo al horno, en estos casos existirá una cantidad de hornos finitos dentro de cada restaurante que funcionarán como entradas salidas visto en la teoría.

Administración de Mensajes

Este módulo reconocerá los siguientes mensajes:

1. Consultar Platos
2. Crear Pedido
3. Añadir Plato
4. Confirmar Pedido
5. Consultar Pedido

Consultar Platos

Se consulta al Módulo Sindicato, todos los platos del Restaurante; de esta forma, se podrá consultar información de los platos que el restaurante sepa cocinar.

Crear Pedido

El Restaurante enviará Guardar Pedido al Módulo Sindicato para la creación de un pedido. Para esto, el restaurante generará un ID de pedido nuevo UNICO que permitirá identificarlo dentro de dicho restaurante.

Añadir Plato

A través del envío del mensaje Guardar Plato al Módulo Sindicato, agrega un plato correspondiente a un pedido específico, que se encontrará relacionado con el Restaurante que envió dicho mensaje. Solo se podrá crear platos sobre pedidos existentes.

Confirmar Pedido

Este mensaje permitirá confirmar un pedido creado previamente. Para esto se recibirá el ID de pedido destino. La funcionalidad de este mensaje se descompone en los siguientes pasos:

1. Obtener el Pedido desde el Módulo Sindicato.
2. Generar el PCB (Plato Control Block) de cada plato del pedido en cuestión y dejarlo en el ciclo de planificación (que se explicará a continuación en el módulo de Planificación de Platos). Para esto, se deberá ejecutar el mensaje Obtener Receta al Módulo Sindicato para saber la trazabilidad que deberá tener al momento de su ejecución. Cabe aclarar que el número de pedido se deberá guardar dentro del PCB para su futuro uso.
3. Informar al Módulo que lo invocó que su pedido fue confirmado.

Consultar Pedido

Se le enviará al Módulo Sindicato el mensaje Obtener Pedido especificando un pedido del Restaurante, con la finalidad de obtener información actualizada del mismo, como por ejemplo, la cantidad lista.

Planificación de Platos

En el momento que un pedido sea confirmado y se hayan creado los distintos PCB de los distintos platos, entrará en este submódulo para la planificación de los mismos. Para esto, se utilizará un diagrama de 5 estados manejando múltiples colas Ready (por las afinidades) y colas de entrada y salida para los distintos hornos donde los platos funcionarán como procesos y los distintos cocineros como procesadores.

La receta dispondrá de tres tipos de operaciones distintas que todas vendrán informadas por medio de su identificador y un número:

1. Reposar: Implica enviar a bloqueado a dicha operatoria por la cantidad de ciclos de CPU que se indique.
2. Hornear: Implica enviar a realizar entrada salida a dicho plato a un horno. En caso que no haya hornos disponibles deberá esperar a que uno de ellos se libere manejando una cola FIFO.
3. Otro: Implica la ejecución de la cantidad de ciclos de CPU que se indique.

Un ejemplo de una receta puede ser:

```
Milanesa
        Trocear 4
        Empanar 5
        Reposar 3
        Hornear 10
```

Por otro lado, este módulo maneja una planificación por afinidad en base a los platos a los cuales cada cocinero es afín. Si un cocinero dispone de una de ellas sólo podrá cocinar dichos platos, mientras que si uno no dispone ninguna podrá cocinar todos los pedidos restantes.

Al momento de inicializar dicho módulo, se deberán crear las distintas colas de ready de manera que al llegar un plato para ser planificado el mismo sepa en cuál de ellas debe entrar.

Cuando un Plato haya sido finalizado (cuando alcance la cola de Exit) se deberá informar al Módulo Sindicato para que actualice el estado del pedido y luego al módulo que solicitó el pedido.

Algoritmos de Planificación

Para la planificación de los distintos cocineros se implementarán los algoritmos FIFO y RR. Todos los algoritmos serán aplicados como se vieron en la teoría funcionando con afinidad si la configuración de dicho restaurante lo dispone.

Logs Mínimos y Obligatorios

Para permitir la verificación/validación del módulo se exigirá tener un archivo de log específico e independiente que contenga, como mínimo, la información indicada en esta sección. ***No se permite la exclusión de ninguno de los mismos*** (la falta de alguno puede implicar la desaprobación del grupo). Cada operación debe loguearse en una única línea indicando la misma y sus datos.

Las acciones a loguear en este archivo son:

1. Creación de un plato (indicando el PCB correspondiente al mismo)
2. Finalización de un PCB (indicando los platos elaborados)
3. Inicio de operación de los platos (ej. cuando comienza a hornearse una milanesa)
4. Finalización de la operación de los platos.

Archivo de Configuración

Campo	Tipo	Descripción
PUERTO_ESCUCHA	[Numérico]	Puerto por el cual el Restaurant escuchara las distintas conexiones.
IP_SINDICATO	[String]	El IP del servidor del proceso Sindicato.
PUERTO_SINDICATO	[Numérico]	El puerto del servidor del proceso Sindicato.
IP_APP	[String]	El IP del servidor del proceso App.
PUERTO_APP	[Numérico]	El puerto del servidor del proceso App.
QUANTUM	[Numérico]	Quantum de RR
ARCHIVO_LOG	[String]	Path del archivo de log donde se almacenará el log obligatorio.
ALGORITMO_PLANIFICACION	[String]	El tipo de algoritmo de planificación que se va a utilizar [FIFO/RR].
NOMBRE_RESTAURANTE	[String]	El nombre del Restaurante.
RETARDO_CICLO_CPU	[Numérico]	Tiempo en segundos para el retardo de la ejecución de cada ciclo de cpu.

Ejemplo de Archivo de Configuración

```
PUERTO_ESCUCHA=5002
IP_SINDICATO=127.0.0.1
PUERTO_SINDICATO=5004
IP_APP=127.0.0.1
PUERTO_APP=5004
```

```
QUANTUM=2
ARCHIVO_LOG=/utnso/logs/THEFOODTRUCKSTORE.log
ALGORITMO_PLANIFICACION=RR
NOMBRE_RESTAURANTE=THEFOODTRUCKSTORE
RETARDO_CICLO_CPU=5
```

Módulo Sindicato

Este módulo nos permitirá implementar nuestro propio almacenamiento de archivos. Los datos a almacenar serán los Restaurantes que se encuentren en el mapa, el historial de pedidos de cada restaurante y las distintas recetas que existan. Para esto, se deberá implementar el FileSystem AFIP explicado en los siguientes apartados.

Este proceso se comunica con los demás a través de un socket de escucha en el cual podrá recibir mensajes de los distintos Restaurantes o Clientes.

AFIP (Administracion de FileSystem de Interrelación Personalizada)

El FileSystem AFIP es un componente creado con propósitos académicos para que el alumno comprenda el funcionamiento básico de la gestión de archivos en un sistema operativo.

Su estructura se basa en una estructura de árbol de directorios para representar la información administrativa y los datos de los Restaurantes y sus respectivos pedidos en forma de archivos. El árbol de directorios tomará su punto de partida del punto de montaje del archivo de configuración.

Durante las pruebas no se proveerán archivos que tengan estados inconsistentes respecto del trabajo práctico, por lo que no es necesario tener en cuenta dichos casos.

Metadata

Este archivo tendrá la información correspondiente a la cantidad y el tamaño de los bloques. Dentro del archivo se encontrarán los siguientes campos:

- **Block_size**: Indica el tamaño en bytes de cada bloque
- **Blocks**: Indica la cantidad de bloques del File System
- **Magic_Number**: Un string fijo con el valor “AFIP”

Ej:

```
BLOCK_SIZE=64
BLOCKS=5192
MAGIC_NUMBER=AFIP
```

Dicho archivo deberá encontrarse en la ruta `[Punto_Montaje]/Metadata/Metadata.AFIP`

Bitmap

Este será un archivo de tipo binario donde solamente existirá un bitmap[2], el cual representará el estado de los bloques dentro del FS, siendo un 1 que el bloque está ocupado, y un 0 que el bloque está libre. Los valores del bitmap deben ser a nivel de bit.

La ruta del archivo de bitmap es: `[Punto_Montaje]/Metadata/Bitmap.bin`

[2] Se recomienda investigar sobre el manejo de los bitarray de las commons library.

Cualquier otra implementación que no conste de utilizar un archivo Bitmap.bin binario será motivo de desaprobación directa.

Files

Restaurantes

Los archivos de cada restaurante, dentro del FS, se encontraran en un path compuesto de la siguiente manera:

```
[Punto_Montaje]/Files/Restaurantes/[Nombre_Restaurante]
```

Donde el path del archivo incluye todos los archivos de dicho restaurante, la información del mismo y su historial de pedidos.

Los pedidos deberán almacenarse siguiente el esquema de:

```
[Punto_Montaje]/Files/Restaurantes/[Nombre_Restaurante]/Pedido[ID_Pedido]
```

A fin de que se los pueda encontrar fácilmente.

Ej:

```
[Punto_Montaje]/Files/Restaurantes/Restaurante1/Info.AFIP
[Punto_Montaje]/Files/Restaurantes/Restaurante1/Pedido1.AFIP
[Punto_Montaje]/Files/Restaurantes/Restaurante1/Pedido2.AFIP
```

Dentro de cada archivo se encontrarán los siguientes campos:

- **Size**: indica el tamaño real del archivo en bytes.
- **Initial_Block**: es el número del bloque inicial en donde se encuentran los datos propiamente dichos de ese archivo.

Ej:

```
SIZE=250
INITIAL_BLOCK=40
```

Recetas

Los archivos de cada receta se encontraran en un path compuesto de la siguiente manera:

```
[Punto_Montaje]/Files/Recetas/[Nombre_Receta]
```

Donde el path del archivo incluye todas las recetas que todos los restaurantes conocen.

Ej:

```
[Punto_Montaje]/Files/Recetas/Milanesa.AFIP
```

Dentro de cada archivo se encontrarán los mismos campos que en los Files Restaurantes.

Bloques

Los datos estarán repartidos en archivos de texto nombrados con un número, el cual representará el número de bloque. (Por ej.: `1.afip`, `2.afip`, `3.afip`). La asignación de los bloques va a ser de tipo **ENLAZADO,** para lo cual, los últimos 4 bytes de cada bloque serán reservados para almacenar el puntero al siguiente bloque. **Este puntero debe ser de tipo `uint_32`.**

Es importante destacar que hasta que el bloque no se haya llenado completamente, no se podrá solicitar un nuevo bloque. Otra implementación que no contemple esto, será motivo de

desaprobación directa.

Dichos archivos se encontraran dentro de la ruta:

`[Punto_Montaje]/Blocks/[Numero_Bloque].AFIP`

Ej:

`/home/utnso/AFIP/Blocks/1.AFIP`

`/home/utnso/AFIP/Blocks/2.AFIP`

Es importante destacar que todos los bloques del FS deben existir al momento de utilizar el FS (Pueden ser creados apenas se levante el proceso en caso de que no existan).

Datos de Restaurante

Los restaurantes constaran de un archivo para almacenar su información y múltiples archivos para representar el historial de pedidos.

Info del restaurante

Dentro de la info del restaurante se deben almacenar los siguientes campos:

- **Cantidad_Cocineros**: indica la cantidad de cocineros con los que cuenta el restaurante.
- **Posicion**: indica la posición del mapa en la que el restaurante está ubicado (x, y).
- **Afinidad_Cocineros**: es un array que indica la afinidad de cada cocinero.
- **Platos**: es un array que indica los platos que este restaurante sabe preparar.
- **Precio_Platos**: es un array que indica el precio de cada plato.
- **Cantidad_Hornos**: indica la cantidad de hornos con los que cuenta el restaurante.
- Cantidad_Pedidos: indica la cantidad de pedidos con los que cuenta el restaurante.

Ej:

```
CANTIDAD_COCINEROS=5
POSICION=[4,5]
AFINIDAD_COCINEROS=[Milanesas]
PLATOS=[Milanesas,Empanadas,Ensalada]
PRECIO_PLATOS=[200,50,150]
CANTIDAD_HORNOS=2
```

Pedidos

Dentro del archivo de cada pedido se deben almacenar los siguientes campos:

- **Estado_Pedido:** Indica el estado del pedido. Puede ser Pendiente/Confirmado/Terminado.
- **Lista_Platos**: es un array que indica las recetas que se deben preparar.
- **Cantidad_Platos**: es un array que indica la cantidad la cantidad de cada receta que se deben preparar.
- **Cantidad_Lista**: es un array que indica la cantidad de cada receta que ya se encuentra lista.
- **Precio_Total**: indica el precio total del pedido.

Ej:

```
ESTADO_PEDIDO=Confirmado
LISTA_PLATOS=[Milanesa,Empanadas,Ensalada]
CANTIDAD_PLATOS=[2,12,1]
CANTIDAD_LISTA=[1,6,0]
```

```
PRECIO_TOTAL=1150
```

Datos de Receta

Las recetas constaran de un único archivo para almacenar su información.

Receta

Dentro de la receta se deben almacenar los siguientes campos:

- **Pasos**: un array que indica los pasos que se deben realizar en orden.
- **Tiempo_Paso**: indica los ciclos de CPU que requiere cada paso.

Ej:

```
PASOS=[Trocear,Empanar,Reposar,Hornear]
TIEMPO_PASOS=[4,5,3,10]
```

Lineamiento e Implementacion

Este proceso gestionará un FileSystem que será leído e interpretado como un árbol de directorios y sus archivos utilizando el FileSystem AFIP.

Este proceso deberá ser implementado de manera multi hilo, con el objetivo de que el mismo pueda atender de una manera más eficiente y de manera concurrente a los diferentes restaurantes y a la consola.

Queda a decisión de cada grupo el implementar un hilo por mensaje o un hilo por cliente, teniendo en cuenta los pros y contras de cada implementación.

Administración de Mensajes

El módulo Sindicato reconocerá los siguientes mensajes que le lleguen por sockets:

1. Consultar Platos
2. Guardar Pedido
3. Guardar Plato
4. Confirmar Pedido
5. Obtener Pedido
6. Obtener Restaurante
7. Plato Listo
8. Terminar Pedido
9. Obtener receta

Consultar Platos

Este mensaje cumplirá la función de obtener todos los platos que puede preparar un determinado Restaurante. Para esto recibirá, el nombre del restaurante, del cual tiene que obtener los platos. Al recibir este mensaje se deberán realizar las siguientes operaciones:

1. Verificar si el Restaurante existe dentro del File System. Para esto se deberá buscar dentro del directorio Restaurantes si existe un subdirectorio con el nombre del Restaurante. En caso de no existir se deberá informar dicha situación.

2. Obtener los platos que puede preparar dicho Restaurante del archivo info.AFIP.
3. Responder el mensaje indicando los platos que puede preparar el Restaurante.

Guardar Pedido

Este mensaje cumplirá la función de crear un nuevo pedido para un Restaurante determinado. Para esto se recibirá el nombre del Restaurante dentro del cual se tiene que crear el pedido; y al recibir este mensaje se deberán realizar las siguientes operaciones:

1. Verificar si el Restaurante existe dentro del File System. Para esto se deberá buscar dentro del directorio Restaurantes si existe un subdirectorio con el nombre del Restaurante. En caso de no existir se deberá informar dicha situación.
2. Verificar que el ID de pedido no exista para dicho restaurante.
    a. En caso de existir se deberá informar sobre dicha situación.
    b. En caso de que **no** exista, se deberá crear el archivo.
3. Responder el mensaje indicando si se pudo realizar la operación correctamente (Ok/Fail).

Guardar Plato

Este mensaje cumplirá la función de agregar un nuevo plato a un determinado pedido. Para esto recibirá:

1. El nombre del Restaurante
2. El ID del pedido
3. El plato que se va a agregar
4. La cantidad del plato a agregar

Al recibir este mensaje se deberán realizar las siguientes operaciones:

1. Verificar si el Restaurante existe dentro del File System. Para esto se deberá buscar dentro del directorio Restaurantes si existe un subdirectorio con el nombre del Restaurante. En caso de no existir se deberá informar dicha situación.
2. Verificar si el Pedido existe dentro del File System. Para esto se deberá buscar dentro del directorio del Restaurante si existe dicho pedido. En caso de no existir se deberá informar dicha situación.
3. Verificar que el pedido esté en estado “Pendiente”. En caso contrario se deberá informar dicha situación.
4. Verificar si ese plato ya existe dentro del archivo. En caso de existir, se deberán agregar la cantidad pasada por parámetro a la actual. En caso de no existir se deberá agregar el plato a la lista de Platos y anexar la cantidad que se tiene que cocinar de dicho plato y aumentar el precio total del pedido.
5. Responder el mensaje indicando si se pudo realizar la operación correctamente (Ok/Fail).

Confirmar Pedido

Este mensaje cumplirá la función de cambiar el estado de un determinado pedido. Para esto recibirá:

1. El nombre del Restaurante
2. El ID del pedido

Al recibir este mensaje se deberán realizar las siguientes operaciones:

1. Verificar si el Restaurante existe dentro del File System. Para esto se deberá buscar dentro del directorio Restaurantes si existe un subdirectorio con el nombre del Restaurante. En caso de no existir se deberá informar dicha situación.
2. Verificar si el Pedido existe dentro del File System. Para esto se deberá buscar dentro del directorio del Restaurante si existe dicho pedido. En caso de no existir se deberá informar dicha situación.
3. Verificar que el pedido esté en estado “Pendiente”. En caso contrario se deberá informar dicha situación.
4. Cambiar el estado del pedido de “Pendiente” a “Confirmado” (se debe truncar el archivo en caso de ser necesario).
5. Responder el mensaje indicando si se pudo realizar la operación correctamente (Ok/Fail).

Obtener Pedido

Este mensaje cumplirá la función de obtener todos los datos de un determinado pedido. Para esto recibirá:

1. El nombre del Restaurante
2. El ID del pedido

Al recibir este mensaje se deberán realizar las siguientes operaciones:

1. Verificar si el Restaurante existe dentro del File System. Para esto se deberá buscar dentro del directorio Restaurantes si existe un subdirectorio con el nombre del Restaurante. En caso de no existir se deberá informar dicha situación.
2. Verificar si el Pedido existe dentro del File System. Para esto se deberá buscar dentro del directorio del Restaurante si existe dicho pedido. En caso de no existir se deberá informar dicha situación.
3. Responder el mensaje indicando si se pudo realizar en conjunto con la información del pedido si correspondiera.

Obtener Restaurante

Este mensaje cumplirá la función de obtener todos los datos de un determinado Restaurante. Para esto recibirá: el nombre del restaurante, del cual tiene que obtener la información. Al recibir este mensaje se deberán realizar las siguientes operaciones:

1. Verificar si el Restaurante existe dentro del File System. Para esto se deberá buscar dentro del directorio Restaurantes si existe un subdirectorio con el nombre del Restaurante. En caso de no existir se deberá informar dicha situación.
2. Obtener todo los datos del archivo `info.AFIP`.
3. Responder el mensaje indicando los datos del Restaurante.

Plato Listo

Este mensaje cumplirá la función de aumentar en uno la cantidad de platos listos para un determinado pedido de un determinado Restaurante. Para esto recibirá:

1. El nombre del Restaurante.
2. El ID de Pedido.
3. El plato que está listo.

Al recibir este mensaje se deberán realizar las siguientes operaciones:

1. Verificar si el Restaurante existe dentro del File System. Para esto se deberá buscar dentro del directorio Restaurantes si existe un subdirectorio con el nombre del Restaurante. En caso de no existir se deberá informar dicha situación.
2. Verificar si el Pedido existe dentro del File System. Para esto se deberá buscar dentro del directorio del Restaurante si existe dicho pedido. En caso de no existir se deberá informar dicha situación.
3. Verificar que el pedido esté en estado “Confirmado”. En caso contrario se deberá informar dicha situación.
4. Verificar si ese plato ya existe dentro del archivo. En caso de existir, se deberá aumentar en uno la cantidad lista de ese plato. En caso contrario se deberá informar dicha situación.
5. Responder el mensaje indicando si se pudo realizar la operación correctamente (Ok/Fail).

Obtener Receta

Este mensaje cumplirá la función de obtener la receta de un plato. Para esto recibirá:

1. El nombre del plato.

Al recibir este mensaje se deberán realizar las siguientes operaciones:

1. Verificar si existe el plato dado dentro del directorio de recetas.En caso de no existir, se deberá informar dicha situación.
2. Responder el mensaje con la receta solicitada.

Terminar Pedido

Este mensaje cumplirá la función de cambiar el estado de un pedido a Terminado. Para esto recibirá:

1. El nombre del Restaurante.
2. El ID del Pedido.

Al recibir este mensaje se deberán realizar las siguientes operaciones:

1. Verificar si el Restaurante existe dentro del File System. Para esto se deberá buscar dentro del directorio Restaurantes si existe un subdirectorio con el nombre del Restaurante. En caso de no existir se deberá informar dicha situación.
2. Verificar si el Pedido existe dentro del File System. Para esto se deberá buscar dentro del directorio del Restaurante si existe dicho pedido. En caso de no existir se deberá informar dicha situación.
3. Verificar que el pedido esté en estado “Confirmado”. En caso contrario se deberá informar dicha situación.
4. Cambiar el estado del pedido a “Terminado”
5. Responder el mensaje indicando si se pudo realizar la operación correctamente (Ok/Fail).

Consola

Mediante una consola, el módulo Sindicato deberá facilitar al usuario las siguientes operaciones:

1. Crear Restaurante: Se deberá crear una nueva carpeta restaurante, con su respectivo info.AFIP explicado anteriormente. El mensaje tendrá el siguiente formato:

```
CrearRestaurante [NOMBRE] [CANTIDAD_COCINEROS] [POSICION]
[AFINIDAD_COCINEROS] [PLATOS] [PRECIO_PLATOS] [CANTIDAD_HORNOS]
```

```
Ej:
CrearRestaurante MiRestaurante 5 [4,5] [Milanesa]
[Milanesa,Empanadas,Ensalada] [200,50,150] 2
```

2. Crear Receta: Se deberá crear un nuevo archivo de receta siguiendo los lineamientos de lo detallado anteriormente. El mensaje tendrá el siguiente formato:

```
CrearReceta [NOMBRE] [PASOS] [TIEMPO_PASOS]
```

```
Ej:
CrearReceta Milanesa [Trocear,Empanar,Reposar,Hornear] [4,5,3,10]
```

Logs Mínimos y Obligatorios

Para permitir la verificación/validación del módulo se exigirá tener un archivo de log específico e independiente que contenga, como mínimo, la información indicada en esta sección. ***No se permite la exclusión de ninguno de los mismos*** (la falta de alguno puede implicar la desaprobación del grupo). Cada operación debe loguearse en una única línea indicando la misma y sus datos.

Las acciones a loguear en este archivo son:

1. Conexión de un proceso Restaurante.
2. Conexión de un proceso Cliente.
3. Recepción de cualquier mensaje mediante socket.
4. Respuesta de cualquier mensaje mediante socket.
5. Asignación de un nuevo bloque para un archivo y su respectivo número.
6. Desasignación de un bloque y su respectivo número.
7. La creación de un nuevo archivo, ya sea por mensaje o por consola.

Archivo de Configuración

Campo	Tipo	Descripción
PUERTO_ESCUCHA	[Numérico]	El puerto del servidor del proceso Sindicato.
PUNTO_MONTAJE	[String]	Indica el punto de partida para AFIP

Ejemplo de Archivo de Configuración

```
PUERTO_ESCUCHA=5003
PUNTO_MONTAJE=/home/utnso/desktop/afip
```

Anexo I - API

El objetivo de este anexo es la definición de la API Global de nuestro sistema. A continuación se explicará cual es la interfaz (que debe recibir cada uno de los mensajes y que debe retornar) y que módulos de nuestro sistema utilizan cada una de ellas.

Por cada método se describirán los siguientes puntos:

- Descripción
- Qué módulo lo recibe
- Su interfaz (que espera recibir y retornar)

Consultar Restaurantes

- Descripción: sin ningún parámetro de entrada, se deberán obtener todos los nombres de los Restaurantes que se encuentran conectados a la App.
- Módulos que reciben el mensaje:
    - App
- Interfaz:
    - Retorna: [nombreDeRes1, nombreDeRes2, ...,nombreDeResN]

Seleccionar Restaurante

- Descripción: dado un Cliente y un Restaurante, se asocia el Cliente a dicho Restaurante. Se esperará la confirmación de la operación.
- Módulos que reciben el mensaje:
    - App
- Interfaz:
    - Recibe
        - Cliente (opcional)
        - Restaurante
    - Retorna: Ok/Fail

Obtener Restaurante

- Descripción: dado un Restaurante, se obtendrá toda la información del mismo que se encuentra persistida en el módulo Sindicato.
- Módulos que reciben el mensaje:
    - Sindicato
- Interfaz:
    - Recibe:
        - Restaurante
    - Retorna:

```
{
        [afinidad,..., afinidad],
        posX,
```

```
        poxY,
        [{receta1,precio1}, ..., {recetaN,precioN}],
        cantHornos,
        cantPedidos
}
```

Consultar Platos

- Descripción: la App consultara los platos de un Restaurante específico, quien obtendrá los mismos enviando el mensaje 'Consultar Platos' al módulo Sindicato, encargado de consultar en sus archivos la información actualizada de los platos.
- Módulos que reciben el mensaje:
    - App
    - Restaurante
    - Sindicato
- Interfaz:
    - Recibe:
        - Restaurante (en el caso de que se envíe del Restaurante al Sindicato).
    - Retorna: [comida1, comida2, ...]

Crear Pedido

- Descripción: el mensaje puede enviarse tanto al Restaurante como a la App. Cuando este se envía al Restaurante, se genera el ID del pedido y se envía 'Guardar Pedido' al módulo Sindicato quien almacenará el mismo.
  En cambio, cuando se ejecuta al módulo App, se enviará el mismo mensaje al restaurante en cuestión donde se obtendrá el ID (igual que en el caso anterior) y luego se ejecutará 'Guardar Pedido' al módulo Comanda.
  En cualquiera de ambos casos se deberá retornar el ID al módulo que lo solicitó.
- Módulos que reciben el mensaje:
    - App
    - Restaurante
- Interfaz:
    - Retorna:
        - ID del pedido

Guardar Pedido

- Descripción: se le enviará al módulo Comanda un nombre de Restaurante específico y un ID de pedido, y este se encargará de guardar el pedido en memoria; análogamente, será enviado al módulo Sindicato, quien se encargará de persistir el pedido dentro de los archivos del Restaurante correspondiente.
- Módulos que reciben el mensaje:
    - Comanda
    - Sindicato

- Interfaz:
    - Recibe
        - Nombre del Restaurante
        - ID de pedido
    - Retorna:
        - Ok/Fail

Añadir Plato

- Descripción: dado un plato y un ID de pedido, el módulo App le informará al Restaurante la adhesión de un plato. En caso de que el plato exista se le deberá sumar 1 a la cantidad de platos existentes. Una vez almacenado, se deberá informar si se pudo concretar la operación.
- Módulos que reciben el mensaje:
    - App
    - Restaurante
- Interfaz:
    - Recibe:
        - Plato
        - ID de pedido
    - Retorna: Ok/Fail

Guardar Plato

- Descripción: el módulo App le informará al módulo Comanda la adhesión de un plato a un pedido, siendo este último el encargado de almacenar en memoria el nuevo plato; mientras que el modulo Restaurante es quien le informará al módulo Sindicato la adhesión del plato al pedido, siendo este último el encargado de actualizar el archivo del pedido.
- Módulos que reciben el mensaje:
    - Comanda
    - Sindicato
- Interfaz:
    - Recibe
        - Restaurante
        - ID de pedido
        - Comida
        - Cantidad
    - Retorna: Ok/Fail

Confirmar Pedido

- Descripción: dado un ID de pedido y un Restaurante, se enviará al Restaurante el pedido a confirmar, este será el encargado de informarle al Sindicato el nuevo estado del pedido, quien lo actualizará en el archivo correspondiente. Una vez que se confirme el pedido en el Restaurante, se le informará a la App quién le ejecutará el mensaje ‘Confirmar Pedido’ al módulo Comanda, quien actualizará el estado del pedido, recibiendo el ID del pedido y el

Restaurante en cuestión. Finalmente, el módulo App, si se procesaron correctamente todas las operaciones, informará al Cliente si se pudo procesar la confirmación.

- Módulos que reciben el mensaje:
    - App
    - Restaurante
    - Comanda
    - Sindicato
- Interfaz:
    - Recibe
        - ID de pedido
        - Restaurante (En caso de Comanda y Sindicato)
    - Retorna: Ok/Fail

Plato Listo

- Descripción: el Restaurante, indicando el ID del pedido, le informará al Sindicato el plato que finalizó, siendo este último el encargado de actualizar los datos dentro del archivo del pedido correspondiente. A su vez, le informará al módulo App el ID del pedido y plato finalizado, quien le informará al módulo Comanda para que este último actualice los datos dentro de la memoria.
- Módulos que reciben el mensaje:
    - App
    - Comanda
    - Sindicato
- Interfaz:
    - Recibe
        - Restaurante
        - ID de pedido
        - Comida
    - Retorna: Ok/Fail

Consultar Pedido

- Descripción:
    - A través de la App, dado un ID de pedido, se consultará la información del mismo al módulo Comanda.
    - Cuando el que responda el mensaje sea enviado al módulo Restaurante, se consultará la información al módulo Sindicato.
- Módulos que reciben el mensaje:
    - App
    - Restaurante
- Interfaz:
    - Recibe
        - ID de pedido
    - Retorna: {Restaurante, estado,[{comida1, cantTotal1, cantLista1}, {comida2, cantTotal2, cantLista2}, ...}

Obtener Pedido

- Descripción: dado un ID de un pedido y un restaurante, el módulo Comanda deberá acceder a su memoria y retornar toda la información del pedido; y el módulo Sindicato deberá acceder a los pedidos correspondientes del Restaurante e informar la información del pedido.
- Módulos que reciben el mensaje:
    - Comanda
    - Sindicato
- Interfaz:
    - Recibe
        - Restaurante
        - ID de pedido
    - Retorna: {estado, [{comida1, cantTotal1, cantLista1}, {comida2, cantTotal2, cantLista2}, ...]}

Finalizar Pedido

- Descripción: el módulo App, una vez que finalice un pedido, se lo deberá informar al módulo Comanda a través del mensaje 'Finalizar Pedido', indicando el ID del pedido y el Restaurante correspondiente.
- Módulos que reciben el mensaje:
    - Comanda
    - Cliente
- Interfaz:
    - Recibe
        - Restaurante
        - ID de pedido
    - Retorna: Ok/Fail

Terminar Pedido

- Descripción: este mensaje será enviado del Restaurante al Sindicato para informar que el pedido fue terminado en su totalidad.
- Módulos que reciben el mensaje:
    - Sindicato
- Interfaz:
    - Recibe:
        - ID del Pedido
        - Restaurante
    - Retorne:
        - Ok/Fail

Obtener Receta

- Descripción: este mensaje será enviado del Restaurante al Sindicato para obtener la receta de un plato específico.
- Módulos que reciben el mensaje:
    - Sindicato
- Interfaz:
    - Recibe:
        - Nombre del plato
    - Retorne:
        - Receta

Anexo II - Ejemplos

Ejemplos de bloques en File System

Los siguiente ejemplos se encuentran bajo la siguiente configuración del `Metadata.afip`:

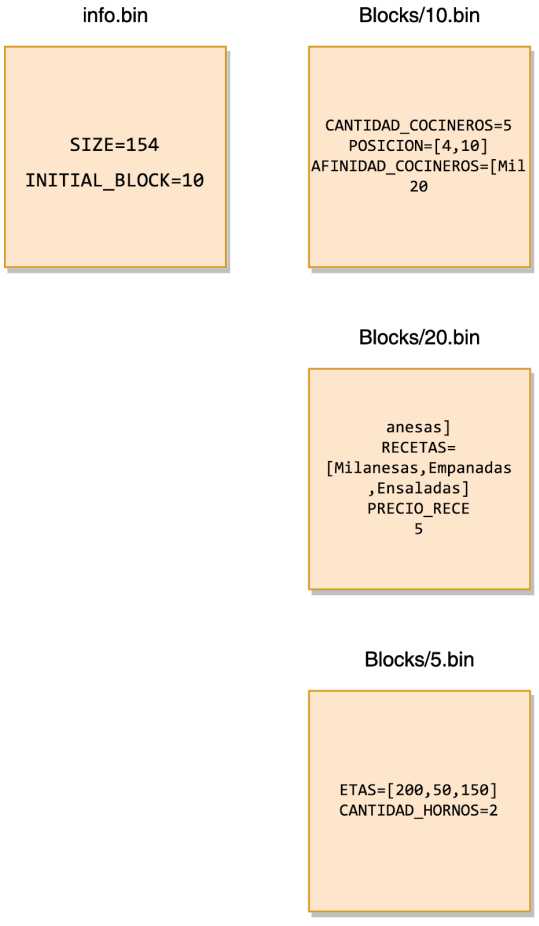
```
BLOCK_SIZE=64
BLOCKS=5192
MAGIC_NUMBER=AFIP
```

Archivo Info

Teniendo el siguiente archivo `info.afip`:

```
CANTIDAD_COCINEROS=5
POSICION=[4,10]
AFINIDAD_COCINEROS=[Milanesas]
RECETAS=[Milanesas,Empanadas,Ensaladas]
PRECIO_RECETAS=[200,50,150]
CANTIDAD_HORNOS=2
```

Se guardará de la siguiente forma en los bloques, los últimos cuatro bytes corresponden al siguiente número de bloque (es decir, el nro. 20 del bloque 10, corresponde al siguiente bloque).

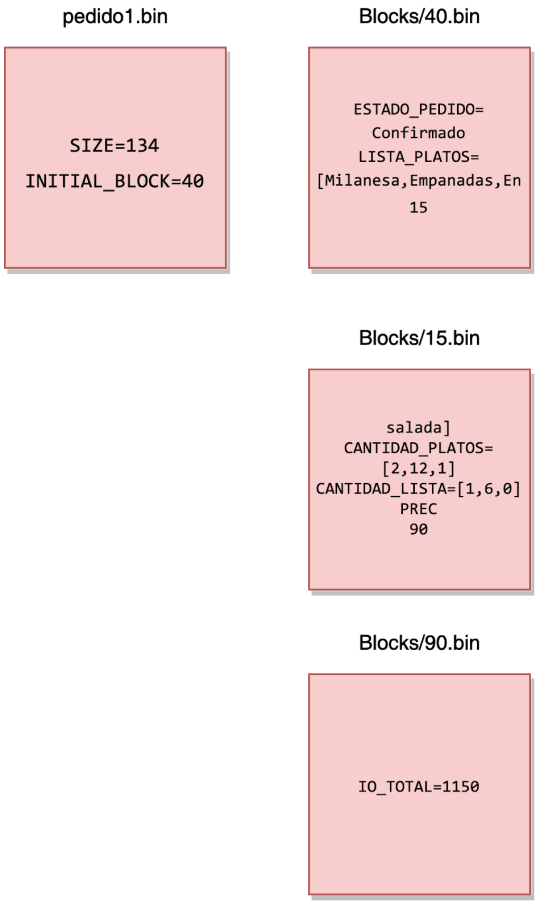


Este archivo es de longitud 150, provocando que se deba dividir en múltiples bloques y, como se especificó anteriormente, se deben llenar todos los bloques con la información que se deba persistir. Cualquier otra implementación será considerada incorrecta.

Archivo Pedido

Teniendo el siguiente archivo `pedido1.afip`:

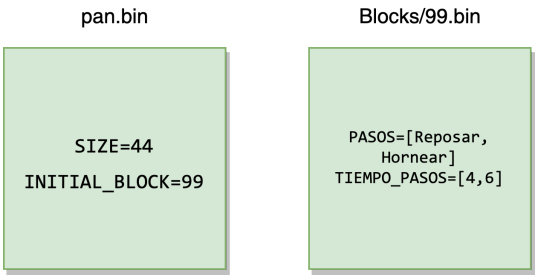
```
ESTADO_PEDIDO=Confirmado
LISTA_PLATOS=[Milanesa,Empanadas,Ensalada]
CANTIDAD_PLATOS=[2,12,1]
CANTIDAD_LISTA=[1,6,0]
PRECIO_TOTAL=1150
```



Archivo Receta

Teniendo el siguiente archivo `pan.afip`:

```
PASOS=[Reposar, Hornear]
TIEMPO_PASOS=[4,6]
```


pan.bin
SIZE=44
INITIAL_BLOCK=99
Blocks/99.bin
PASOS=[Reposar,
Hornear]
TIEMPO_PASOS=[4,6]

Descripción de las entregas

Hito 1: Conexión Inicial

Fecha: 12/09

Objetivos:

- Familiarizarse con Linux y su consola, el entorno de desarrollo y el repositorio.
- Aplicar las Commons Libraries, principalmente las funciones para listas, archivos de conf y logs.
- Definir el Protocolo de Comunicación.
- Comenzar el desarrollo de las consolas del Cliente y Sindicato.

Lectura recomendada:

- Tutorial de "Cómo arrancar" de la materia: http://faq.utnso.com.ar/arrancar
- Beej Guide to Network Programming - https://beej.us/guide/bgnet/
- SO UTN FRBA Commons Libraries - https://github.com/sisoputnfrba/so-commons-library
- Guía de Punteros en C - http://faq.utnso.com.ar/punteros

Hito 2: Avance del Grupo

Fecha: 03/10

Objetivos:

- **Proceso Cliente**: Desarrollar consola y serialización de mensajes.
- **Proceso App**: Desarrollar inicialización del módulo, permitir la planificación de forma FIFO.
- **Proceso Comanda**: Desarrollar inicialización del módulo.
- **Proceso Sindicato**: Finalizar consola del proceso.

Lectura recomendada:

- Sistemas Operativos, Stallings, William 5ta Ed. - Parte IV: Planificación
- Sistemas Operativos, Silberschatz, Galvin 7ma Ed. - Capítulo 5: Planificación

Hito 3: Checkpoint Obligatorio "Presencial" - Vía pantalla compartida

Fecha: 24/10

Objetivos:

- **Proceso Cliente**: Finalización y cierre de detalles del módulo.
- **Proceso App**: Desarrollo de planificación por HRRN y administración de movimientos de repartidores.
- **Proceso Restaurante**: Desarrollar inicialización del módulo, permitir la planificación de forma FIFO sin Entrada/Salida.
- **Proceso Comanda**: Implementación de sistema de memoria sin SWAP.
- **Proceso Sindicato**: Desarrollar mensajes del módulo con datos de prueba (para permitir la inicialización del proceso Restaurante).

Lectura recomendada:

- Sistemas Operativos, Stallings, William 5ta Ed. - Parte IV: Planificación
- Sistemas Operativos, Silberschatz, Galvin 7ma Ed. - Capítulo 5: Planificación
- Sistemas Operativos, Stallings, William 5ta Ed. - Parte VII: Gestión de la memoria (Cap. 7)
- Sistemas Operativos, Silberschatz, Galvin 7ma Ed. - Capítulo 8: Memoria principal

Hito 4: Avance del Grupo

Fechas: 07/11

Objetivos:

- **Proceso App**: Finalización del módulo APP.
- **Proceso Restaurante**: Desarrollar colas con Afinidad y RR.
- **Proceso Comanda**: Comienzo de implementación de memoria con SWAP.
- **Proceso Sindicato**: Manejo de estructura de árbol y del Bitmap. Permitir la escritura de archivos. Comienzo de desarrollo de lectura de archivo.

Lectura recomendada:

- Sistemas Operativos, Stallings, William 5ta Ed. - Parte VII: Memoria virtual (Cap. 8)
- Sistemas Operativos, Silberschatz, Galvin 7ma Ed. - Capítulo 9: Memoria virtual
- Sistemas Operativos, Stallings, William 5ta Ed. - Parte V: Gestión de ficheros (Cap. 12)
- Sistemas Operativos, Silberschatz, Galvin 7ma Ed. - Capítulo 10: Interfaz del sistema de archivos

Hito 5: Entregas Finales

Fechas: 28/11 - 05/12 (o 12/12, a confirmar) - 19/12

Objetivos:

- Probar el TP en un entorno distribuido
- Realizar pruebas intensivas
- Finalizar el desarrollo de todos los procesos
- Todos los componentes del TP ejecutan los requerimientos de forma integral, bajo escenarios de stress.

Lectura recomendada:

- Guías de Debugging del Blog utnso.com - https://www.utnso.com.ar/recursos/guias/
- MarioBash: Tutorial para aprender a usar la consola - http://faq.utnso.com.ar/mariobash
- Tutoral de como desplegar un proyecto - https://github.com/sisoputnfrba/so-deploy