Universidad Simón Bolívar

División de Ciencias Físicas y Matemáticas

Departamento de Computación y Tecnología de la Información

CI3825: Sistemas de Operación I

PROYECTO I:

TRANSPORTE USB

Profesor:

Fernando Torre Mora

Estudiantes:

Junior Lara, 17-10303

Laura Parilli, 17-10778

Jhonaiker Blanco, 18-10784

Astrid Alvarado, 18-10938

Sartanejas, 24 de marzo de 2023

Introducción

En el siguiente trabajo se presentan los detalles referentes a la implementación, en lenguaje C y empleando los estándares de UNIX, de un programa capaz de simular un día de servicio del transporte universitario con el fin de medir la suficiencia de este al ejecutar diferentes escenarios ejemplificados en archivos de texto, en particular en formato “TXT” y “CSV”. De esta forma, se espera determinar la eficacia de este servicio para así poder tener un estimado de su capacidad y, en consecuencia, analizar el mayor obstáculo para retomar la modalidad de clases presencial.

Los detalles expuestos a continuación exploran a profundidad las elecciones tomadas al momento de diseñar el programa, las Estructuras de Datos empleadas para el manejo de los datos ingresados, dificultades encontradas junto a las soluciones propuestas para las mismas, además de presentar los resultados obtenidos luego de ejecutar el programa en diversas circunstancias, evaluando variaciones en los casos de prueba.

Decisiones de Diseño e Implementación

El programa descrito a lo largo de este trabajo es capaz de ser ejecutado desde la cónsola mediante el comando

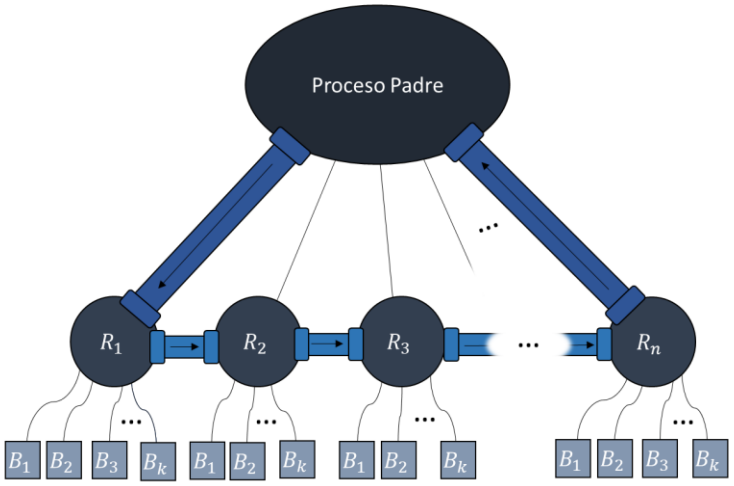
```
$ ./simutransusb -s <archivo> [-c <archivo>] [-t <num>]
```

donde

- `simutransusb` corresponde con el nombre del archivo ejecutable creado una vez el programa se haya compilado correctamente
- `-s` indica el nombre del archivo de caracterización de servicios, donde se almacena para cada parada la hora y los minutos de partida de cada autobús junto su capacidad. Este es un archivo de texto obligatorio para el funcionamiento del programa, en caso de no especificar ninguno se da un mensaje de error.
- `-c` denota el nombre del archivo de caracterización de carga. Este es un archivo de texto que separa los valores que contiene mediante el uso de comas, a este tipo de archivo se le conoce como “comma-separate values” (CSV) y sirve para la representación de tablas. Aquí yace información más detallada de cada ruta, como viene siendo el código y nombre de esta, la hora y los minutos correspondientes a la duración del recorrido; y finalmente la cantidad de usuarios que llega a una hora determinada. Este es un argumento opcional y, en caso de no ser indicado, se supone el archivo “carga.csv”.
- `-t` representa la cantidad de segundos que debe durar un minuto en la simulación, por lo que para la actualización del tiempo debe ser en función a

este argumento. Este es un parámetro opcional por lo que, si no se indica, se supone el valor de 0.25 segundos.

La idea para simular el servicio de transporte USB consiste en la creación de un proceso principal que lleve el control de las horas transcurridas y dé la impresión por pantalla del progreso de la simulación, así como la , para esto se realizará la creación de tantos procesos hijo como paradas se desee emular, y estos a su vez desencadenarán en la creación de suficientes hilos para poder representar los buses que atenderán a cada ruta en una hora determinada, tal como se ilustra en el anexo siguiente:



Aquí, es posible apreciar que el programa es implementado con el fin de realizar una comunicación entre procesos donde el primer y el último proceso hijo son los encargados de transferir información desde y hasta el proceso padre. De la misma manera, y una vez se haya ejecutado la acción de cada hilo, cada proceso transmite la información entre ellos, para que así los procesos ejecuten su código de manera ordenada, como una cadena. Asimismo, es posible observar cómo en cada proceso se derivan múltiples hilos, por lo que la sincronización de estos juega un papel importante para poder obtener los datos correspondientes.

Al finalizar el programa, se muestra en pantalla a tiempo simulado el progreso de los autobuses, tomando en cuenta el siguiente formato:

```
Simulation Time: HH:MM
```

P_1: u_1 [(>||<||…)$t_{1,1}$( )10-$t_{1,1}$] $B_{1,1}$…[(>||<||…)$t_{k_1,1}$( )10-$t_{k_1,1}$] $B_{k_1,1}$

P_2: u_2 [(>||<||…)$t_{1,2}$( )10-$t_{1,2}$] $B_{1,2}$…[(>||<||…..)$t_{k_2,2}$( )10-$t_{k_2,2}$] $B_{k_2,2}$

⋮

P_n: u_n [(>||<||…)$t_{1,n}$( )10-$t_{1,n}$] $B_{1,n}$…[(>||<||…..)$t_{k_n,n}$( )10-$t_{k_n,n}$] $B_{k_n,n}$

donde

- `HH:MM` es la hora y el minuto del día simulado
- n es el número de paradas que tiene el servicio
- P_i es el código de la i-ésima parada que tiene servicio
- u_i es el número de usuarios esperando en la i-ésima parada
- k_i es el número de autobuses actualmente efectuando recorridos desde o hacia la i-ésima parada
- $B_{j,i}$ es el j-ésimo autobús efectuando un recorrido desde o hacia la i-ésima parada
- $t_{j,i}$ es la posición del j-ésimo autobús efectuando un recorrido desde o hacia la i-ésima parada, medida en décimos
- “>” y ”<” indican la dirección del autobús, es decir, si se dirigen dirección a la universidad o dirección hacia la parada respectivamente. Asimismo, “…” indica al usuario cuando un autobús se encuentra esperando por personas en la parada

Una vez impreso esto, se da una lista por cada ruta del nivel de eficiencia o ineficiencia obtenida, estos siendo medidos contando la cantidad de personas

Entrando más a detalle, para el funcionamiento del programa se decidió plantear la creación de un archivo de cabecera “standard_lib.h” en el cual se almacenan todas las librerías requeridas, variables globales y estructuras de datos empleadas, así como macros y directivas definidas para mayor facilidad de implementación. Por lo que, en el archivo de código fuente, la misma es incluida para el correcto funcionamiento del programa.

Una vez expuesto esto, se hablará con sumo detalle sobre el programa contemplando cuatro aspectos fundamentales: almacenamiento de los datos ingresados, creación y comunicación entre procesos; creación y sincronización entre hilos y finalmente la actualización del tiempo transcurrido.

Almacenamiento de Datos

Para mayor facilidad al momento de acceder y manipular los datos, se crean diversas estructuras de datos de suma importancia, las cuales se ubican en el archivo de cabecera “standard_lib.h”, a saber:

- **struct charge:** estructura para almacenar los datos relacionados al archivo de carga, esta posee los atributos de
  - **empty:** variable del tipo entero que funciona como indicador de vacío, este ayuda a saber si la estructura fue actualizada con los datos correspondientes.

- **queue_per[max_hour]:** cola de usuarios que llegan a una hora determinada. En esta se almacena en la posición de la hora la cantidad de usuarios que llegan a la misma.
- **code[4]:** variable del tipo de carácter para almacenar el código correspondiente a la parada. Se reserva un tamaño de 4 para poder almacenar en su totalidad el código respectivo incluido el carácter nulo (\0)
- **name[64]:** variable del tipo carácter para almacenar el nombre completo de la parada. Se reserva un tamaño de 64 para tener espacio suficiente para almacenar el nombre respectivo, incluido el carácter nulo (\0)
- **min_travel:** variable del tipo entero para almacenar el tiempo de recorrido del bus.
- **peopleThatDidnotGetTheBus:** variable del tipo entero para contabilizar la cantidad de personas que no consiguieron subir a tiempo en cada parada.
- **totalPersonInRoute;** variable del tipo entero para tener la cantidad de personas en total que harán uso del servicio de transporte en cada parada.

- **struct services:** estructura para almacenar los datos relacionados al archivo de servicio, esta posee los atributos de
  - **empty:** variable del tipo entero que funciona como indicador de vacío, este ayuda a saber si la estructura fue actualizada con los datos correspondientes.
  - **code[4]:** variable del tipo de carácter para almacenar el código correspondiente a la parada. Se reserva un tamaño de 4 para poder almacenar en su totalidad el código respectivo incluido el carácter nulo (\0)
  - **struct time_b leaveing:** estructura de tiempo para almacenar las horas y los minutos en la que los autobuses parten desde la universidad hasta sus respectivas paradas. Esta estructura nos permite almacenar en variables separadas el valor de las horas y el valor de los minutos.
  - **c_capacity:** variable del tipo entero para almacenar la capacidad de cada unidad de transporte que arribará a la parada.
  - **travel_time:** variable del tipo entero para almacenar el tiempo que tarda el bus en cumplir su ruta.
  - **progressPercentage:** variable del tipo entero para almacenar el progreso del trayecto correspondiente a cada bus. Esta sirve para poder actualizar correctamente la simulación y representar su recorrido
  - **isWaitingForPeople:** variable del tipo entero que funciona como indicador para saber si un autobús se encuentra en estado de espera por pasajeros.

    - **isReturningToUniversity:** variable del tipo entero que funciona como indicador para saber si un autobús se encuentra regresando a la universidad.
    - **peopleCharged:** variable del tipo entero para contabilizar la cantidad de personas que han subido a un autobús determinado.
- **amountOfBusesUsedByRoute[n_routes]:** arreglo de enteros para almacenar la cantidad de autobuses que serán utilizados por una ruta. Aquí se almacena para cada ruta i el total de los autobuses usados. Este arreglo tendrá como tamaño predeterminado `n_routes`, macro definida con el valor de 50.
- **amountOfBusesFinishedByRoute[n_routes]:** arreglo de enteros para almacenar la cantidad de autobuses que ya culminaron su recorrido en una determinada ruta. Aquí se almacena para cara ruta i todos los autobuses terminados. Este arreglo tendrá como tamaño predeterminado `n_routes`, macro definida con el valor de 50.
- **struct charge total_cha[n_routes]:** arreglo del tipo "struct charge" para almacenar todos los datos ingresados por el archivo de carga. Aquí, cada ruta tendrá almacenado los datos ingresados respectivos. Este arreglo tendrá como tamaño predeterminado `n_routes`, macro definida con el valor de 50.
- **struct services total_ser[n_routes][max_bus]:** arreglo bidimensional del tipo "struct services" para almacenar para cada ruta, los datos correspondientes de cada autobús proporcionados por el archivo de servicio. Este arreglo tendrá como tamaño predeterminado `n_routes x max_bus`, macros definidas con el valor de 50 y 200 respectivamente.

La lectura de los archivos está sujeta a las funciones siguientes:

- **ReadCacCharge():** función dedicada a extraer la información del archivo de carga y almacenar estos datos en el arreglo "total_cha".
- **ReadCacServices():** función dedicada a extraer la información del archivo de servicios y almacenar estos datos en el arreglo bidimensional "total_ser".
- **open_files(int, char**):** función para poder abrir los archivos dados. Esta función tiene como argumento la cantidad de archivos a ser abiertos, así como los archivos correspondientes
- **initial_structs():** función cuyo objetivo es el de inicializar las estructuras "total_cha" y "total_ser" con valores por defecto, esto con el fin de evitar problemas con el manejo de datos.
- **update_structs():** función cuyo objetivo es el de actualizar las estructuras "total_cha" y "total_ser" con valores por defecto, esta vez tomando en consideración datos como el número de procesos a crear (dependientes de la cantidad de rutas extraídas), esto con el fin de evitar problemas con el manejo de datos.

- **ErrorArgument(int, char**):** función para la detección de errores al momento de llamar el programa. Esta indica al usuario cuando no ingresó el archivo de servicio necesario para la ejecución de la simulación.

Creación y Comunicación entre Procesos

Tal como se pudo apreciar en la imagen antes mostrada, el programa consta de un proceso principal que genera tantos procesos hijo como rutas se vaya a emular en la simulación, ejecutándose a manera de ciclo donde son el primer y último proceso hijo los que tienen comunicación con el proceso padre, y en cambio el resto de procesos se comunican entre sí. Crear un proceso es posible gracias a la utilización de la librería "unistd.h", y en este programa en particular, la realización de estos procesos se hace de forma iterativa, almacenando en un arreglo los identificadores de los nuevos procesos (denominados "pids") generados mediante el uso de la función "fork()", en donde el proceso padre será aquel cuyo "pid" sea distinto del valor 0.

Al crear múltiples procesos que necesitan compartir información, es de suma importancia la creación de "pipes" para la comunicación entre ellos. Para conseguir esto, se hace uso de la función "pipe(int[2])", el argumento ingresado es un archivo denominado "file descriptor" y en este es posible controlar la lectura y la escritura en el mismo, y en concreto, este programa consigue esto mediante el uso de en un arreglo bidimensional con tamaño suficiente para crear un pipe para cada proceso.

Teniendo esto en cuenta, es importante aclarar que para la solución de este problema el procesamiento de los datos relacionado a las rutas es ejecutado por los procesos hijos, el padre sólo se encargará de la actualización del tiempo simulado y su respectiva impresión. A continuación, se explica cada función involucrada con esta parte del funcionamiento del programa:

- **child_function():** función encargada de ejecutar todos los procedimientos encargados del proceso hijo. Aquí, se calcula la cantidad de personas que no llegaron a la universidad o aquellas personas que lograron llegar pero posterior a 1½ horas, así como aquellas personas que sí consiguieron subir a tiempo. De la misma manera, es aquí donde se controla la sincronización de los hilos que simulan el recorrido de los buses. Esta función hace uso de
  - **print_bus(int, int):** función auxiliar para la impresión de un autobús, esta toma como parámetros la el porcentaje del recorrido y la dirección en la que se encuentra avanzando.
- **child_pids[num_of_process + 2]:** arreglo donde se almacenan los "pids" de cada proceso hijo creado. Este tiene tamaño suficiente para lograr la comunicación en ciclo entre todos los proceso.

- **files_desc[num_of_process + 2][2]:** matriz con tantas filas como número de procesos haya y dos columnas, esta se encarga de almacenar todos los "pipes" creados para la comunicación entre cada proceso.

Creación y Sincronización entre Hilos

Para este programa, es necesario atender a todos los buses de manera simultánea para poder tener cálculos más veloces, por lo tanto el uso de hilos es una solución viable para distribuir el trabajo de las rutas. Un hilo es la subdivisión de un proceso en múltiples subprocesos, esto con el fin de que trabajen de forma separada y agilizar las acciones del proceso que los creó, esto siendo posible mediante la librería "pthread.h" la cual ofrece todas las funciones necesarias para la utilización de este método.

Aquí, cada hilo representará un autobús que atenderá la parada correspondiente, y estos hilos se encargan de manejar la actualización de los buses, es decir, calcular su porcentaje del trayecto recorrido y la dirección en la que avanza el mismo. Para el mismo se utiliza

- **pthread_t listOfPthreads[n_routes][max_bus]:** arreglo bidimensional para almacenar los hilos creados por cada ruta, este con tamaño suficiente para poder generar todos los hilos correspondientes a la cantidad máxima de autobuses.
- **void *showBus(void *data):** función de tipo "void*" que caracteriza a los hilos. En esta función obtenemos la actualización del porcentaje del recorrido por cada bus y maneja la dirección en la que se dirige. Además, esta función hace uso de:

Al requerir que todos los hilos trabajen a la vez, es necesario sincronizar estos para evitar condición de carrera, es decir, evitar conflictos de uso de recursos por dos o más hilos al mismo tiempo. Para eso, se establece un turno para cada hilo para que así estos se ejecuten de manera ordenada, cosa que brinda el uso de una matriz de hilos, pues el orden se establece al momento de obtener que un parte a su respectiva parada.

Simulación del Tiempo transcurrido

La simulación del tiempo es controlada por el padre, esta es medida cada vez que se completa un ciclo del programa, es decir, luego de la ejecución de los primeros buses de todas las rutas se busca mantener el ritmo de la hora simulada. Esta se realiza mediante el uso de las librerías "sys/time.h" y "unistd.h", para poder utilizar las funciones "usleep" y "getttimeofday". Por efectos de implementación, la simulación inicia 5 minutos antes de que el primer autobús inicie su recorrido.

Para tener la precisión de que el programa avanza como debe, se toma el tiempo antes y después de escribir y leer los pipes que existen entre el proceso padre y el primer y último proceso hijo, esto empleando la función de "gettimeofday" y su estructura correspondiente

“struct timeval” para poder almacenar los segundos y milisegundos transcurridos. Una vez hecho esto, se busca calcular el tiempo transcurrido mediante la diferencia de estos, y hacer la conversión necesaria para tener este tiempo en unidades de milisegundos, luego, se verifica si el tiempo transcurrido de escribir y leer los pipes es menor que el tiempo que se debe simular, y en caso de ser así, se busca quitar ese tiempo transcurrido del tiempo a simular. Una vez hecho esto, se suspende el programa dicha cantidad de tiempo y se aumenta el tiempo de la simulación. Es necesario tomar el tiempo de ejecución de esta suspensión puesto que se debe tener en cuenta que la naturaleza de “usleep” es dormir el programa tomando como cota inferior el tiempo solicitado, por lo que si hubo un exceso, es necesario eliminar el mismo del tiempo que debe simularse el programa y así conseguir acumular todas estos excedentes para poder avanzar como es requerido. En caso de que el tiempo transcurrido de escribir y leer los pipes es mayor o igual que el tiempo que se debe simular, simplemente se aumenta la hora simulada.

Para este fin se hace uso de la siguiente función:

- **tracker_hour(int):** función que dado un entero es capaz de hacer la obtención de las horas y minutos del argumento dado y realizar su impresión con el formato de “Simulation Time: HH:MM”. Esta, recibe la hora simulada en unidades de minutos para realizar los cálculos necesarios y obtener el formato deseado de la simulación.

Dificultades Encontradas y Decisiones posteriores

A continuación, se exponen las dificultades presentadas a lo largo de la realización de este programa y las decisiones que se tomaron para solventarlas. Además, se muestra algunas decisiones de implementación realizadas para mejor manejo de datos y hacer más fácil la escritura de código.

- La **lectura de los archivos** representó un desafío para la obtención de los datos, pues era necesario la correcta separación de estos y que cada valor pudiera ser almacenado correctamente. Por otro lado, la correcta forma de almacenar estos en caso de que los archivos de servicio y carga presenten diferencias, es decir, que en caso de que estos archivos no coincidan en rutas se busca sólo mantener aquellos que sí tenga correspondencia.
- La **simulación del tiempo** de forma correcta y precisa fue un inconveniente al momento de implementarla. Esto es debido a que, como se mencionó con anterioridad, la función “usleep” puede suspender el programa por más tiempo que el debido, asimismo, tomar en consideración el tiempo que tarda la comunicación entre los procesos y que esta información sea reportada al proceso padre.
- El uso de **semáforos** con múltiples hilos fue un conflicto a la hora de implementación. Dada a la gran cantidad de estos hilos, hacer uso de semáforos

para conseguir la sincronización de estos fue complejo de llevar a cabo, por lo que se decidió emular esta funcionalidad con otorgándole a cada hilo un turno en función a los autobuses que empezaban su recorrido.

- Buscar recrear **el ciclo deseado de la comunicación entre procesos padre e hijos** fue un desafío para conseguir ser ejecutado satisfactoriamente. Se requirió de la comprensión del funcionamiento de pipes, y de suma atención a la hora de manejar la correctamente la información transferida entre los procesos hijos y la obtención de esta en el proceso padre.
- La **obtención de la suficiencia o insuficiencia de las rutas** ocasionó conflictos al momento de calcularlos. Esto es debido a que se debe tener en consideración el tiempo de espera en la cola, del mismo modo, se debe llevar un seguimiento del trayecto de los buses para determinar si estos pasajeros llegaron tarde a la universidad.
- La **decisión de empezar la simulación 5 minutos antes de que el primer autobús inicie su recorrido** otorga mayor facilidad para el manejo de la hora que debe simular el programa, por lo que fue una decisión para evitar posibles problemas con el manejo del mismo.
- Se decidió realizar **modificaciones en el formato de salida** para hacer más legible la simulación. Se le ofrece al usuario más visión de cómo esta realiza su recorrido y el momento justo en que estos autobuses esperen en la parada.
- La decisión de **definir de macros y directivas** para mayor facilidad al momento de implementar el programa. Con esto se busca tener fácil acceso a valores constantes de uso constante, así como la creación de directivas capaces de ejecutar un for-loop y calcular el máximo o mínimo de dos números dados. Esto ofrece mayor claridad al momento de leer el código, así como mejor funcionalidad en las operaciones.
- La **utilización de arreglos para almacenar los datos de los archivos** fue una decisión que permitió el fácil acceso a los mismos. Esto es gracias a la simplicidad con la que obtener elementos de un arreglo, unidimensional o bidimensional, conlleva y la velocidad constante que esta tiene.

Evaluación de las Configuraciones

El programa se sometió a diversas pruebas con el fin de medir la suficiencia de los transporte. Los datos obtenidos fueron extraídos de los archivos de otorgados “carga.csv”, “servicio2019.txt”, “carga2007.csv” y “servicio2007.txt”, dando la información necesaria para poder emular un día de servicio. Es claro enfatizar que los archivos tienen correspondencia entre sí, es decir, que los datos de “servicio2019.txt” corresponden a los datos de “carga.csv” y, del mismo modo, los datos ofrecidos por el archivo “servicio2007.txt” corresponden al archivo de “carga2007.csv”. Es por esta razón que el programa es capaz de ejecutarse cuando el único argumento dado por consola es el archivo que representa un día de servicio del año 2019, puesto

que se supone por defecto que el archivo de caracterización de carga es el de este año. Es así como, para la obtención de la simulación de un día de servicio del año 2007, es necesario darle la información correcta al programa.

El formato que deben seguir los archivos de carga y servicios son los siguientes

- Archivo de caracterización de la carga del sistema

  `Cod, Nombre, Recorr, h1, h2,…, h8`

  P_1, PL_1, $hh_1 : mm_1$, $u_{1,1}$, $u_{1,2}$,…, $u_{1,8}$

  P_2, PL_2, $hh_2 : mm_2$, $u_{2,1}$, $u_{2,2}$,…, $u_{2,8}$

  ⋮

  P_n, PL_n, $hh_n : mm_n$, $u_{n,1}$, $u_{n,2}$,…, $u_{n,8}$

  donde

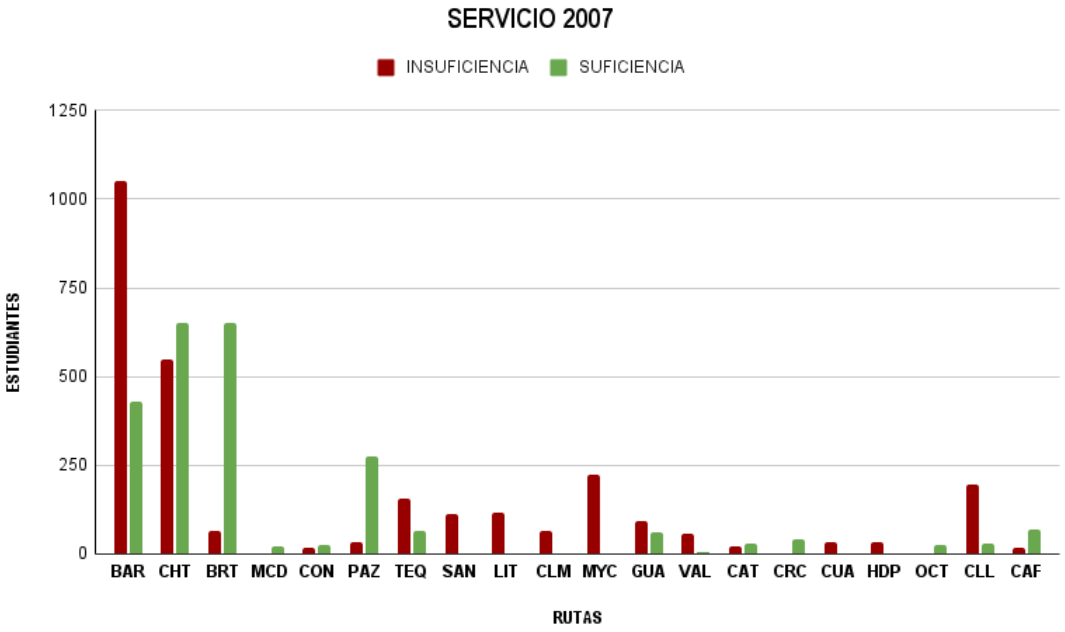
  - `hk` es la k-ésima hora a la que arriban usuarios a la parada indicado como un número entero.
  - PL_i es el nombre largo de la i-ésima parada.
  - $hh_i : mm_i$ es la duración del recorrido entre la universidad y la i-ésima parada.
  - $u_{i,k}$ es el número de personas que llegaron a la i-ésima parada.
- Archivo de caracterización del servicio del sistema

  P_1 $hh_{1,1} : mm_{1,1}(c_{1,1})$ $hh_{1,2} : mm_{1,2}(c_{1,2})$ … $hh_{1,n1} : mm_{1,n1}(c_{1,n1})$
  P_2 $hh_{2,1} : mm_{2,1}(c_{2,1})$ $hh_{2,2} : mm_{2,2}(c_{2,2})$ … $hh_{2,n2} : mm_{2,n2}(c_{2,n2})$
  ⋮
  P_n $hh_{n,1} : mm_{n,1}(c_{n,1})$ $hh_{n,2} : mm_{n,2}(c_{n,2})$ … $hh_{n,nk} : mm_{1,nk}(c_{1,nk})$

  donde
  - $hh_{i,j} : mm_{i,j}$ es la hora en la que parte el j-ésimo autobús de la universidad a la i-ésima parada.
  - $c_{i,j}$ es la capacidad del j-ésimo autobús a la i-ésima parada.

Una vez establecido esto, se procede a evaluar los resultados obtenidos luego de realizar la ejecución del programa. En los siguientes gráficos, se hace una comparativa de aquellos estudiantes que llegaron a la universidad luego de 1½ horas, mostrados como “insuficiencia” del transporte universitario en el mismo, con aquellos estudiantes que por el contrario, sí logran llegar antes de esperar 1½ horas, esto denotado como “suficiencia” del transporte universitario en los gráficos que serán mostrados a continuación.

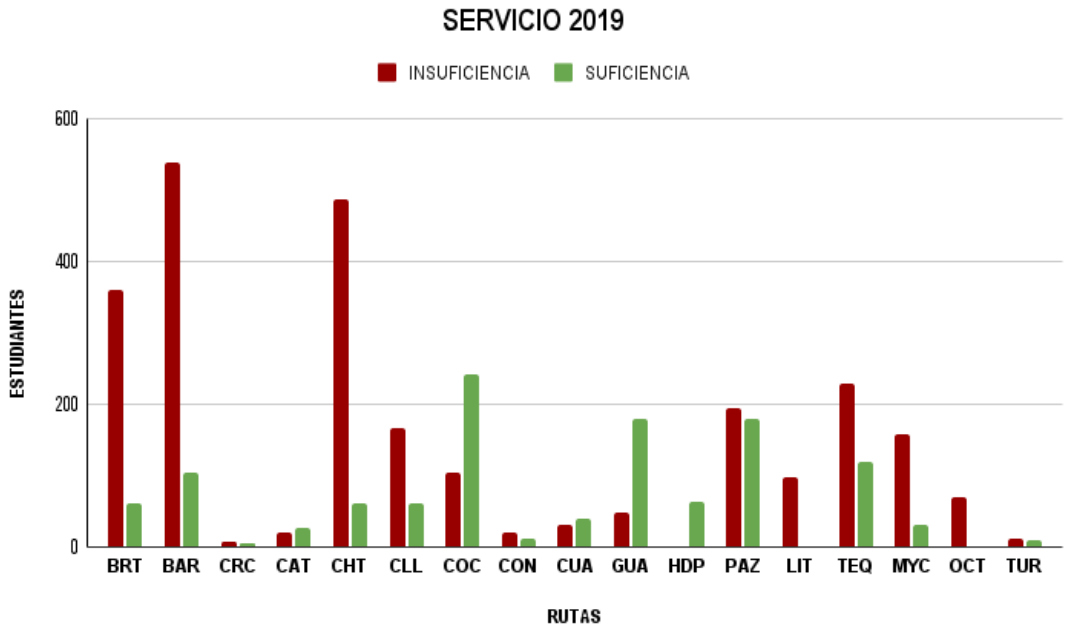
En primer lugar, se analizará los resultados obtenidos luego de hacer la prueba siguiendo los datos de “servicio2007.txt” y “carga2007.csv”



Es posible apreciar como para el año 2007, el esquema de transporte sí cumple con la eficacia esperada en el servicio. Esto se puede ver puesto que las rutas más concurridas como Chacaito, Baruta y La Paz consiguen satisfacer la demanda de las mismas, consiguiendo que la mayoría de las personas lleguen a tiempo a la universidad, aunque para el caso de Bellas Artes, que es una ruta masiva, no es posible satisfacer la demanda de la misma. Rutas menos solicitadas como Litoral y Catia La Mar no son eficientes al momento de cumplir con la demanda de cada usa, esto es debido al largo tiempo del viaje que este conlleva y la poca asignación de buses para estas, por otro lado, San Antonio, Los Teques, Maracay, tampoco consiguen cumplir con el objetivo por la misma razón, aunque cabe destacar que la solicitudes de estas paradas son mayores que el Litoral y Catia La Mar, haciendo ver una mala distribución en los autobuses asignados.

Sin embargo, se puede concluir que para aquellas rutas donde la cantidad de personas es mayor se consigue dar abasto, y en consecuencia las rutas cuyo recorrido es más tardío o son menos concurridas son opacadas por las rutas masivas. Por lo tanto mantener este esquema realizando ligeras modificaciones, como asignar una mayor cantidad de autobuses a las rutas antes expuestas que mostradas deficiencia, o la reasignación de buses con mayor capacidad para el caso de Bellas Artes, podría ser una solución a esta insuficiencia.

Ahora, se analizará los resultados obtenidos luego de hacer la prueba siguiendo los datos de “servicio2019.txt”



Aquí, se puede observar que para 2019 el esquema del servicio del transporte de la universidad tiende a no satisfacer la demanda de las rutas. Se puede apreciar como las rutas más concurridas, como es el caso de Baruta, Bellas Artes, Chacaito y La Paz, pese a tener mayor cantidad de autobuses con mayor capacidad, esto no da abasto para poder hacer que las personas lleguen a tiempo. Por otro lado, en rutas como Caricuao y Concresa/Santa Fe la demanda no es satisfecha pues, a pesar de ser rutas con un tiempo de recorrido corto, sólo le es asignado un bus en un único horario, por lo que las personas que no puedan tomar el bus a la hora establecida, pierden la oportunidad de subir a la universidad haciendo uso del transporte.

Para rutas como Charallave y Los Teques, pese a ser rutas de alta demanda, el tiempo de recorrido afecta a que la eficiencia del transporte se cumpla, pues los autobuses parten a una hora tardía. Del mismo modo, rutas como Litoral, Maracay, Santa Teresa/Ocumare del Tuy y Turgua/Sabaneta no son rutas de alta demanda, por lo que sólo se ven afectadas por la duración de su recorrido.

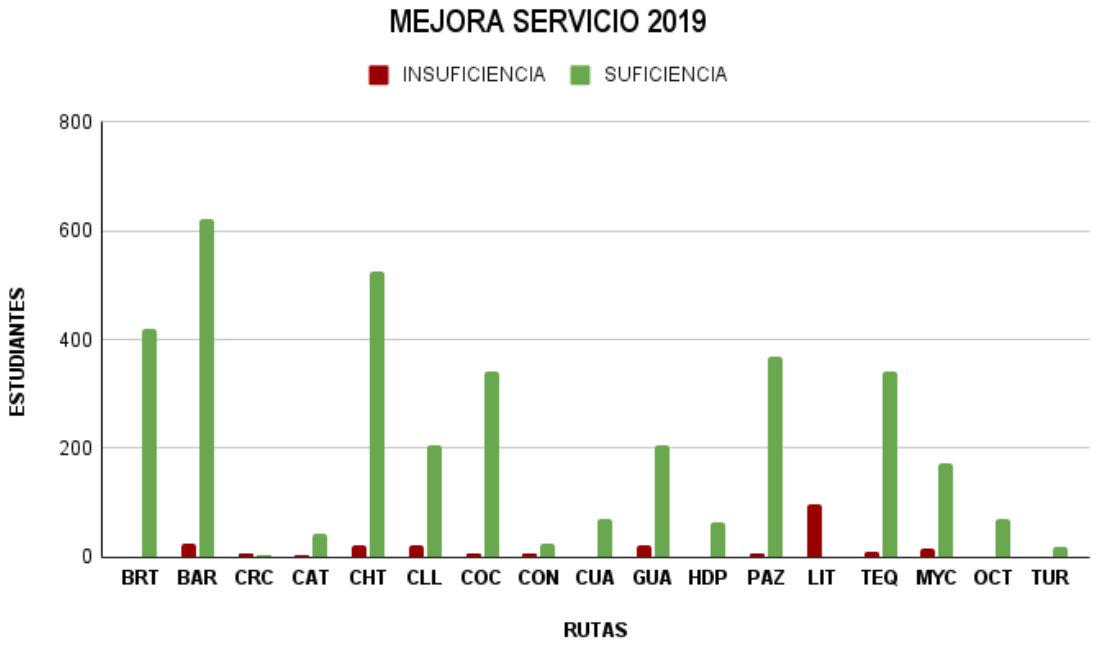
Por lo que, teniendo esto en cuenta, se puede apreciar que el servicio del transporte no cumple con la demanda, sin embargo, es importante aclarar que en comparación con los resultados obtenidos del año 2007, existe una mejoría en rutas de larga duración o con menos solicitudes, pues antes no era posible lograr un mínimo de suficiencia. No obstante, se puede apreciar como las rutas más concurridas empeoraron su desempeño, pues ahora en ninguna de estas rutas es posible que las personas lleguen a tiempo a la universidad. Es por esto que es necesario replantear el esquema de este año, tomando en cuenta el esquema del 2007, para así conseguir un equilibrio entre ambos resultados obtenidos.

Configuración Propuesta

Para conseguir mejorar la eficiencia del servicio, se propone calcular un rango en el que los autobuses partan de la universidad tal que consigan cumplir su eficiencia. Este rango es calculado tomando en cuenta la hora en la que lleguen mayor cantidad de personas, la cual se denominará como hora máxima, siendo cota inferior 9 minutos antes de esta restado con el recorrido total, esto es debido a que se toma en consideración que el tiempo de espera de cada autobús es de 10 minutos en la parada y luego emprende su viaje hacia la universidad, por lo que se empieza justo en el momento antes de su partida, a este valor se le denotará como hora final en la que los autobuses pueden partir. Por otra parte, la cota superior debe ser hora y media después de la hora máxima, restado con el recorrido total de cada autobús, es importante aclarar que se espera 1½ horas después dado que es ese tiempo el que determina el límite de la suficiencia, y es ésta a la que se llamará como hora inicial en la que los autobuses pueden partir.

Siguiendo la misma idea, teniendo en cuenta la capacidad de los buses y la mayor cantidad de personas halladas en la hora máxima se puede determinar los buses que son requeridos para satisfacer la demanda de cada ruta, y distribuir los mismo en intervalos equitativos, siendo asignados tomando como referencia la hora final en la que los buses deben iniciar su recorrido hacia la parada correspondiente. Es importante aclarar que para esta solución el esquema original sufrió una reasignación de buses, quitando o agregando buses para tener una mejoría en los resultados en las horas de mayor solicitud.

Para mostrar esto, se realizará pruebas siguiendo el esquema presentado en el archivo "servicio2019.txt". En la gráfica a continuación se puede ver el resultado obtenido



Aquí, es posible ver que la propuesta consigue el objetivo de mejorar el funcionamiento del servicio de transporte. Es posible ver cómo se logra una mejora del 93.52% en comparación con el esquema original, donde se obtuvo el 31.91% de suficiencia, mostrando así como el esquema propuesto aporta una gran solución a la insuficiencia del transporte. Esta propuesta se encuentra adjunta en el archivo entregado para mejor visualización de los datos cambiados.

Conclusiones

Al realizar este programa se pudo ver que el esquema solución planteado de mejoras servicios USB sirve como propuesta para la estructuración de futuros esquemas de transporte, pues dicho esquema cuenta con una estructura que es posible generalizar para cualquier plan de servicio funcional. En concreto, con el ejemplo de los resultados de "servicio2019.txt" se puede ver que, además de la solución mostrada, reasignar eliminando o agregando buses de varias rutas con el fin de enfocarse en horas de alta demanda, logrando reducir hasta un 60% la insuficiencia del plan original, incluso proyectando mejoras a nivel de inversión de unidades de transporte. Por lo que en conclusión, esta perspectiva permite cuestionar la problemática de la eficiencia basada en aquel grupo de personas que llegan antes de las 1½ horas luego de su llegada a la parada, dando oportunidad a cambiar el objetivo de eficiencia meta del plan a construir.

La implementación de este programa permitió mayor comprensión del manejo diversos procesos para realizar una tarea como lo es recrear un día de servicio del transporte universitario, y de la misma manera, fue de utilidad para el entendimiento del uso de hilos y que los procesos generados pudieran tener una distribución en su tarea. Gracias a esto, se pudo conseguir un mejor entendimiento de la comunicación entre procesos y sincronización entre los hilos, así como el manejo del tiempo del programa para la correcta simulación del tiempo transcurrido, aprendiendo y perfeccionando el uso de todas las librerías y funciones diseñadas para este fin.