Proyecto 1
Comunicación de procesos sincronizada

1st Acevedo Rodríguez Kevin
Instituto Tecnológico de Costa Rica
Principios de Sistemas Operativos
Cartago, Costa Rica
kevinar51@estudiantec.cr

2nd Camacho Hernández José Julián
Instituto Tecnológico de Costa Rica
Principios de Sistemas Operativos
Cartago, Costa Rica
jcamacho341@estudiantec.cr

3rd Venegas Vega Jose Agustín
Instituto Tecnológico de Costa Rica
Principios de Sistemas Operativos
Cartago, Costa Rica
joseagustinveneveg@estudiantec.cr

4th Solís Arguello Juan Antonio
Instituto Tecnológico de Costa Rica
Principios de Sistemas Operativos
Cartago, Costa Rica
JuanSa@estudiantec.cr

***Resumen*—This document contains the whole documentation of a Syncronized Process Communication project, such as a short introduction to explain important information to improve the comprehension of the different theoric topics, also contains the design details, usage instructions and develop activities per student, among others. This project has a lot of intersting common cases, corner cases and topics that will be explained later, semaphores usage, shared memory creation and usage, between them.**

***Index Terms*—Comunicación, Memoria compartida, Memoria circular, Semáforos, Sincronizador.**

I. Introducción

En el presente documento explicaremos la creación de un Sincronizador de Comunicación de Procesos, donde se buscará abarcar de forma clara y concisa principalmente la comunicación y sincronización entre estos. Cabe destacar que dichos procesos pueden ser generados de forma indiscriminada y sin límite alguno.

El proyecto a exponer consiste en el desarrollo de cuatro distintos procesos, los cuales tienen funcionalidades completamente distintas. Como se mencionó anteriormente, se pueden contar con ilimitados procesos ejecutándose de forma simultánea, de modo que se deberá crear y gestionar un sistema de memoria compartida sólido y útil para este fin, el cual en contraste con un sincronizador, mantendrá el sistema de ejecución ininterrumpido y libre de bloqueos.

Para terminar esta introducción, se desea brindar conocimiento claro diversos temas de importancia mencionados anteriormente:

- **Memoria Compartida:** Consiste en espacios creados de forma predefinida mediante uno de los procesos, el cual es el Inicializador, por otro lado, dicha memoria inicializada dependerá de diversos factores, como la cantidad de espacios de memoria circular y el largo del texto de entrada.
- **Memoria Circular:** Memoria dependiente de la cantidad de espacios que desea el usuario que los procesos de Emisión y Recepción tengan a su disposición para realizar sus labores.
- **Sincronizador:** Elemento que se encargará de resguardar la integridad de aquellos elementos considerados de acceso crítico, durante este proyecto, los sincronizadores a utilizar serán semáforos, los cuales serán explicados posteriormente.
- **Comunicación:** La comunicación es la acción de lectura y de escritura de un proceso en memoria compartida, para efectos de este proyecto, siempre que se requiera escribir o leer, se deberá estar sujeto a un sincronizador.

Inicialmente, abarcando la memoria compartida, esta

II. Ambiente de desarrollo

A continuación se detalla la configuración básica necesaria para la ejecución del proyecto. El mismo fue desarrollado en un ambiente Linux, por tanto fue posible la utilización del conjunto de herramientas base que ofrece dicha plataforma como GNU, *gcc*, entre otras bibliotecas básicas.

Adicionalmente, entre las herramientas, *frameworks*, bibliotecas y demás aplicaciones de desarrollo que fueron utilizadas para la implementación del proyecto se encuentran las siguientes:

- **Semaphore:** Biblioteca que fue fundamental para la administración de los recursos recursos compartidos mediante la utilización de los semáforos.
- **Time:** Biblioteca útil para la obtención de la hora en que fue introducido un caracter en memoria circular.
- **GitHub:** Plataforma que contiene el repositorio de la tarea. Fue de gran utilidad para el manejo de versiones, la sincronización y el acceso del código actualizado para los miembros del equipo.
- **GitKraken:** Software que facilitó el manejo del repositorio en *GitHub*, al brindar una representación visual de aspectos importantes como las *branches*, *commits*, entre otros.

- **Visual Studio Code:** Editor de texto que fue útil en el manejo y programación de los diferentes archivos de código en C, *makefile*, entre otros.

III. Atributos

III-A. Aprendizaje continuo

III-A1. Necesidades actuales de aprendizaje para enfrentar: Las necesidades actuales de este proyecto es el aprendizaje en la comunicación y sincronización de heavy process por medio de las técnicas de memoria compartida y semáforos.

III-A2. Tecnologías que se pueden utilizar para el desarrollo: Las diferentes tecnologías utilizadas para el desarrollo del proyecto son algunas como por ejemplo la librería Semaphore: Biblioteca que fue fundamental para la administración de los recursos recursos compartidos y también se utilizaron herramientas de inteligencia artificial como ChatGPT que nos facilitó la resolución de dudas y problemas presentes durante el desarrollo del proyecto.

III-A3. Acciones implementadas para el desarrollo del proyecto: El grupo tuvo una reunión inicial en la cual se decidió hacer investigación sobre las diferentes librerías o herramientas que nos podrían facilitar el desarrollo del proyecto, luego de esto nos reunimos una vez para realizar la planeación del mismo y ahí fuimos desarrollando en grupo y investigando de manera grupal,tríos, parejas, o de forma individual los problemas que se fueron presentando

III-A4. Evaluación de forma crítica de la eficiencia de las acciones implementadas en el contexto tecnológico: Desde un punto crítico se cumplen con todos los objetivos diseñados para la elaboración del proyecto y desde nuestro punto de vista lo hace de una manera eficiente y confiable, se utilizan múltiples herramientas que no permitieron desarrollar código de manera eficiente y segura también trabajamos de manera ordena y equipo por lo tanto el desarrollo del proyecto fue un éxito

III-B. Herramientas de ingeniería

III-B1. Herramientas, bibliotecas o recursos que se utilizaron en el proyecto: Se utilizan diferentes tecnologías, herramientas, y bibliotecas para el desarrollo del proyecto entre las más destacadas son la siguientes: Semaphore, time , github ,git kraken, visual studio code entre otras herramientas que han sido mencionadas en el paper que nos permitieron desarrollar el proyecto de una manera más eficiente.

III-B2. Manera en que se aplicaron los recursos o bibliotecas seleccionados: Tanto la biblioteca Semaphore se utilizó para el control de los diferentes semáforos utilizados en el desarrollo del proyecto , la biblioteca time se utilizó para la obtención de la hora en la cual fue introducción o leído un carácter, tanto github como gti kraken nos permitieron el alojamiento y la gestión del código y visual studio code como editor de texto con todas sus extensiones que nos permiten un desarrollo más cómodo del código.

III-B3. Manera en que se adaptaron los recursos para desarrollar el proyecto: La biblioteca Semaphore, se utiliza para controlar diferentes procesos o secciones críticas del código, evitando problemas de concurrencia o conflictos en la ejecución del software, la biblioteca time nos permite registrar y analizar los tiempos de respuesta del software y tanto GitHub como Git Kraken son útiles para colaborar en equipo y mantener un historial de cambios en el código fuente y con todas estas adaptaciones y el lenguaje de desarrollo C, se logró desarrollar de manera efectiva el proyecto.

IV. Detalles del diseño

A continuación, se presenta una serie de diagramas ayudan a describir el funcionamiento general del sistema.

IV-A. Diagrama de Funcionalidad

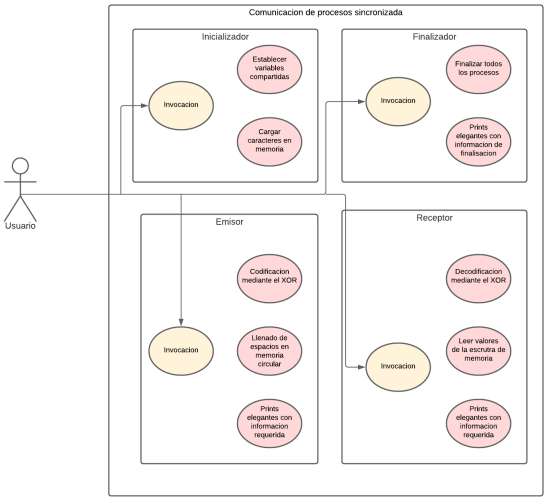

Figura 1. Diagrama de funcionalidad

En el fig 1 podemos observar el diagrama de funcionalidades presentes en el desarrollo de este proyecto, en este se puede observar cómo el usuario solo interactúa con los procesos ejecutando su innovación y pasando los parámetros necesarios para que este sea ejecutado de manera correcta, luego de esto cada uno de los procesos cumplen con sus funcionalidades.

IV-B. *Diagrama de Arquitectura*

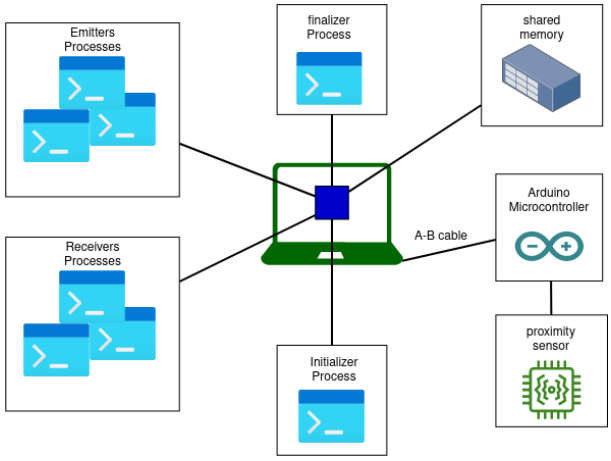

Figura 2. Diagrama de arquitectura de la solución planteada.

A como se puede ver en la figura 2, el diagrama de arquitectura muestra la organización y conexión de cada uno de los tipos de procesos (Que son ejecutados por el computador por medio de terminales de Linux) y los elementos de hardware externos (Arduino y el sensor de proximidad) que apoyan al proceso de finalizador.

IV-C. *Diagrama de Componentes*

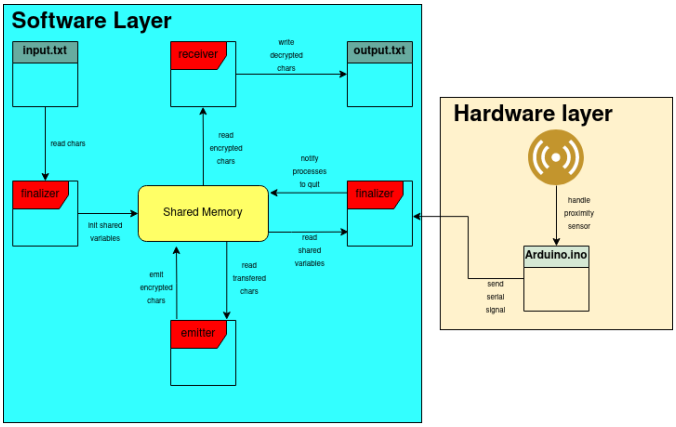

Figura 3. Diagrama de componentes de la solución planteada.

Tal y como se observa en la figura 3, los componentes principales del sistema se dividen en componentes de software y de hardware.

Entre los componentes de software se tienen los heavy processes (que son programas ejecutables de C), los 2 archivos de texto (El input que contiene los caracteres que se deben transferir, y el output que es en el cual se escriben los caracteres por el receptor), y un componente que representa la sección de memoria compartida.

En los componentes de hardware se tiene al módulo de ejecución del Arduino que se encarga de enviarle una señal al finalizador cuando el sensor de proximidad haya detectado algo.

IV-D. *Diagramas de secuencia*

Se presentan dos diagramas de secuencia para poder describir a grandes rasgos el funcianamiento principal del emisor y el receptor tomando en consideración objetos como la memoria compartida, terminal, semáforos y más.

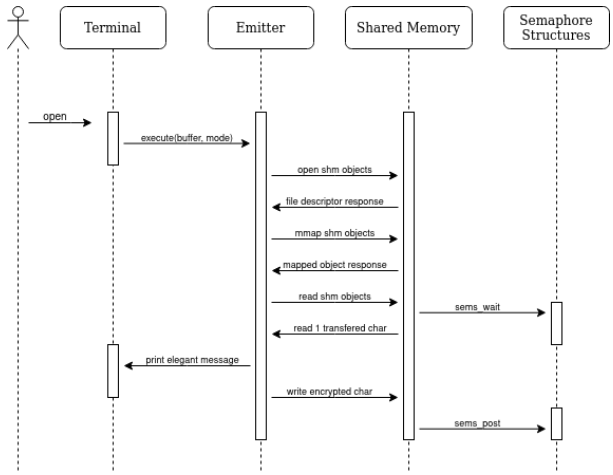

Figura 4. Diagrama de secuencia para describir al emisor.

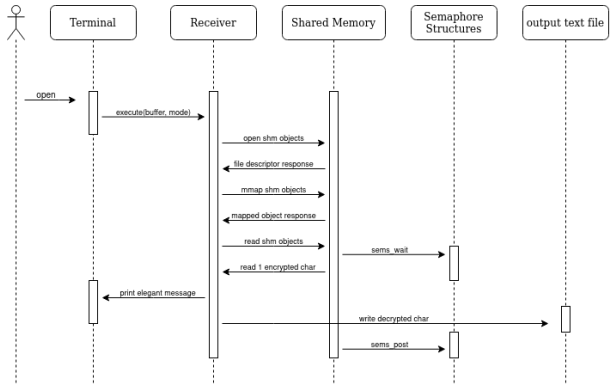

Figura 5. Diagrama de secuencia para describir al receptor.

En la figura 4 se describe de manera general la ejecución del emisor por medio de los siguientes mensajes entre objectos:

1. El usuario abre una terminal para ejecutar el proceso del emisor.
2. El emisor solita los objectos de memoria compartida y recibe un response con el file descriptor.
3. El emisor solicita mapear los objetos de memoria compartida y recibe un response de error o éxito.
4. El emisor intenta leer las variables de memoria compartida, por lo que se hace un sem-wait a los semáforos correspondientes.

5. El emisor lee un caracter transferido a memoria compartida.
6. El emisor le indica a la terminal que se imprima un mensaje elegante.
7. El emisor encripta el char leido y lo escribe en la memoria circular compartida.
8. Se ejecuta un sem-post para indicar que el emisor ya terminó de usar los recursos compartidos en el turno actual.

Seguidamente, en la figura 5 se muestra cómo el receptor funciona (De manera general) por medio de los siguientes mensajes entre objectos:

1. El usuario abre una terminal para ejecutar el proceso del receptor.
2. El receptor solita los objectos de memoria compartida y recibe un response con el file descriptor.
3. El receptor solicita mapear los objetos de memoria compartida y recibe un response de error o éxito.
4. El receptor intenta leer las variables de memoria compartida, por lo que se hace un sem-wait a los semáforos correspondientes.
5. El receptor lee un caracter encriptado de memoria circular y lo desencripta.
6. El receptor le indica a la terminal que se imprima un mensaje elegante.
7. El receptor escribe el caracter desencriptado en el archivo de texto de salida.
8. Se ejecuta un sem-post para indicar que el receptor ya terminó de usar los recursos compartidos en el turno actual.

V. Instrucciones de cómo se utiliza el proyecto

1. Como primer paso se debe ejecutar el Inicializador esto se puede realizar de dos manera desde el Makefile con el comando make init o ./initializer bufferMcK 3 10101010
2. Luego de esto podemos ejecutar el emisor esto de la mimsa forma en caso de que sea Automatico y desde el Makefile make emitAuto en caso que sea manual make emitManual o con los comando ./emitter bufferMcK (auto/manual)
3. Para ejecutar al receptor de la misma forma en caso de que sea Automatico y desde el Makefile make receiveAuto en caso que sea manual make receiveManual o con los comando ./receiver bufferMcK (auto/manual)
4. Para ejecutar el finalizador de la misma manera que inicializador tenemos dos manera de hacerlo make finalize o ./finalizer bufferMcK
5. Es recomendable antes de ejecutar el finalizador ejucar el archivo arduino para el uso del sensor de proximidad
6. Todos estos procesos deben ejecutarse desde una terminal diferente y tanto el emisor y receptor debe de tener el mismo modo para cada una de las instancias de estos

VI. Tabla de actividades por cada estudiante

A continuación se presenta la distribución de las tareas identificadas en la tarea por estudiante, así como datos como su fecha límite establecida, horas dedicadas y estado de completitud.

Descripción	**Responsable(s)**	**Horas de trabajo**	**Estado**
Crear una primera versión del inicializador mediante el uso de memoria compartida.	Agustín y Kevin	**4**	Completado
Generar espaciamiento dinámico en memoria compartida	Juan y Julián	**3**	Completado
Utilización de memoria comparida desde otro archivo	Juan y Kevin	**2**	Completado
Creación del proceso Emisor extracción e interpretación de datos	Julián y Agustín	**3**	Completado
Implementación de lógica de escritura	Kevin	**3**	Completado
Mejoramiento de inicializador y asignación de variables necesarias	Juan y Julián	**2**	Completado
Implementación de semáforos, uso mediante emisor	Agustín	**2**	Completado
Mejoramiento de emisor, prueba con múltiples emisores	Juan y Julián	**3**	Completado
Creación de una versión inicial del receptor	Agustín	**3**	Completado
Generación completa de receptores	Agustín y Julián	**3**	Completado
Interpretación de datos leídos Generación de archivo resultante	Kevin y Juan	**2**	Completado
Encriptación y Desencriptación de datos	Kevin	**1**	Completado
Creación de finalizador mediante entrada manual	Kevin y Agustín	**3**	Completado
Funcionalidad de ejecución por modo de operación	Julián	**4**	Completado
Finalización mediante señal externa	Juan	**4**	Completado

Cuadro I
TABLA DE ACTIVIDADES POR ESTUDIANTE

VII. Conclusiones

En el proceso de realización del trabajo fue posible comprobar la importancia de las diversas herramientas de trabajo y tecnologías que están a disposición para el implementación de este tipo de proyectos, ya que apoyan y facilitan las labores de los desarrolladores.

Adicionalmente, se reforzaron conceptos relacionados con el análisis de requerimientos, diseño de sistemas, y desarrollo de programas en lenguaje C.

De igual manera, se reforzó el tema visto en el curso sobre procesos pesados. Fue posible comprobar que estos no comparten espacios de memoria, sino que hay que crear por aparte dichos espacios con el fin de que ambos accedan y se puedan comunicar de esa manera.

Finalmente, el proyecto fue de gran provecho para comprender y aplicar el tema del curso relacionado con la sincronización del uso de recursos compartidos, como lo puede ser la memoria en este caso. Esto mediante el uso de semáforos de la biblioteca `<semaphore.h>`, los cuales hicieron posible a los distintos procesos pesados, leer y escribir en un mismo espacio de memoria.

VIII. Sugerencias y recomendaciones

El equipo reflexionó sobre todo lo realizado, y se considera que en base a nuestras experiencias, podríamos aportar cierto

conjunto de sugerencias y recomendaciones, las cuales serían muy útiles a futuro, para quien interese y desee entender y/o desarrollar un proyecto como el realizado:

- Si se trabajará en equipo, se recomienda mantener actividades, reglas y medidas que contribuyan a la compenetración del equipo, al desarrollo constante y al interés por el trabajo individual de cada uno, esto contribuirá al entendimiento general de la aplicación y su desarrollo.
- Para este tipo de proyectos donde la sincronización y el manejo de memoria es clave, se recomienda en gran medida realizar un diseño previo del trabajo. Esto con el objetivo de ubicarse con lo que se realizará en lugar de codificar sin una estructura definida.
- Respecto a sugerencias funcionales, para la creación de memoria compartida entre procesos pesados, se recomienda abstraer la misma por medio de mapeos, y accederla mediante la utilización de *file descriptors* con los cuales se logre identificar cada espacio compartido reservado.
- Con el fin de evitar el *busy waiting*, se sugiere la utilización de semáforos para sincronizar efectivamente el acceso a los espacios de recursos compartidos, en ese caso memoria, por parte de los *n* procesos que podrán accederla.
- En la utilización de los semáforos, se recomienda llevar un control detallado de las operaciones atómicas de *up* y *down* de cada semáforo, con el fin de administrar los recursos de la mejor manera posible, y así reducir la posibilidad de *starvation* de los procesos.
- Respecto a esto, también se sugiere identificar correctamente la región crítica de cada proceso.
- Finalmente, se sugiere manejar con especial cuidado los espacios de memoria compartida, ya que un mínimo detalle puede ocasionar un error de *segmentation fault* que puede ser fatal en el funcionamiento correcto del proyecto.

IX. Referencias:

[1] Tanenbaum. Operating systems: design and implementation. Prentice Hall. 1988

[2] Peterson. Operating Systems Concepts. Addison Wesley, Second Edition. 1985

[3] Lubomir y Shaw, The logical design of operating systems. Prentice Hall, Second Edition. 1988.

[4] Kamburugamuve, S., Wickramasinghe, P., Govindarajan, K., Uyar, A., Gunduz, G., Abeykoon, V., Fox, G. (2018, July). Twister: Netcommunication library for big data processing in hpc and cloud environments. In 2018 IEEE 11th International Conference on Cloud Computing (CLOUD) (pp. 383-391). IEEE.

[5] Liguori, A. N., Wilson, M. S., Nowland, I. P. (2018). U.S. Patent No. 9,886,297. Washington, DC: U.S. Patent and Trademark Office.