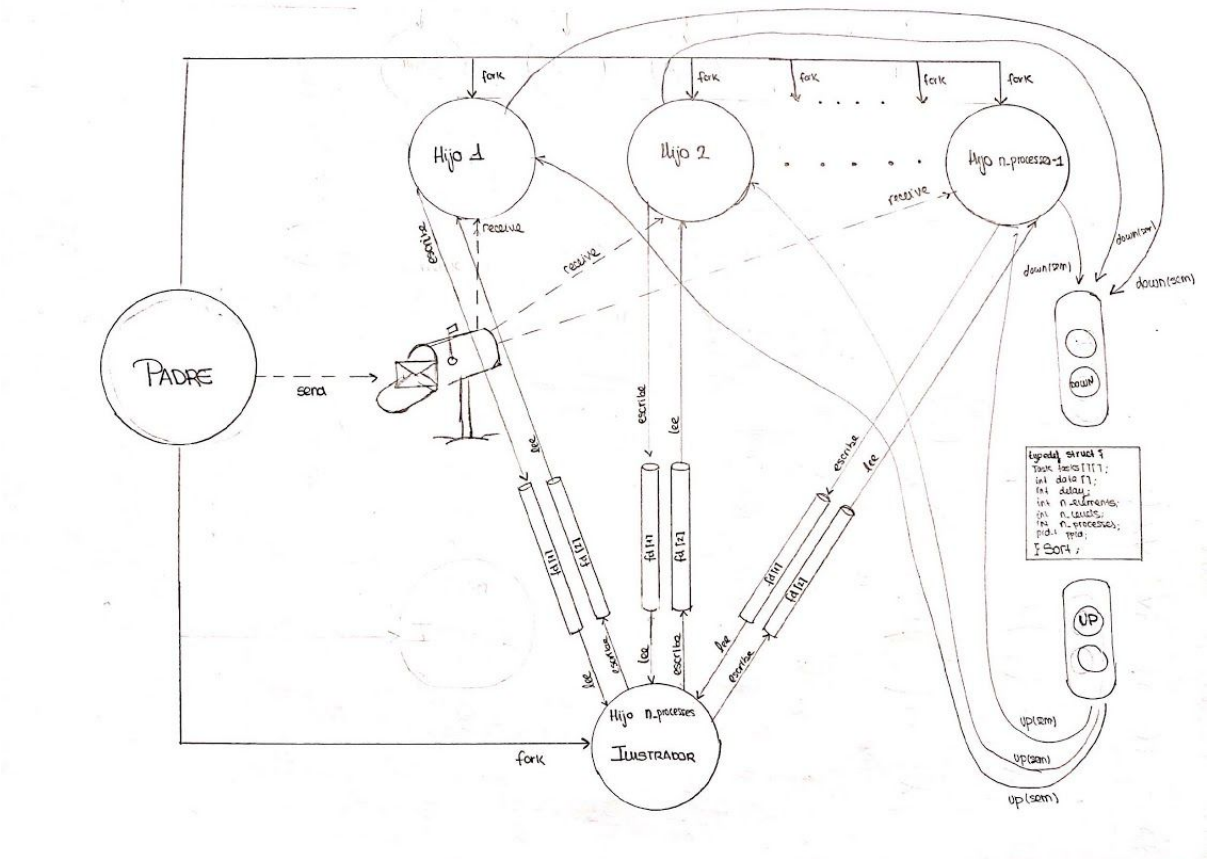
# MEMORIA PROYECTO FINAL

## SISTEMAS OPERATIVOS

Rubén García de la Fuente
Elena Cano Castillejo
Pareja 04
Grupo 2202

**Ejercicio 1:** **Memoria** *(1,50 ptos.)*

a. **Realizar un diagrama que muestre el diseño del sistema, incluyendo los distintos componentes (procesos) y sus jerarquías, así como los mecanismos de comunicación y sincronización entre ellos.** *(0,50 ptos.)*




En el siguiente diagrama representamos de la mejor forma posible y más visual el diseño de nuestro sistema. Podemos observar que aparece el proceso padre el cual crea mediante forks a sus hijos. El número máximo de hijos creados se pasa mediante el argumento n_processes, en nuestro caso el último hijo creado es el ilustrador como se puede ver en la imagen.

Hemos representado como un mailbox la cola de mensajes, pues así se puede ver de forma más significativa. Podemos observar que el padre es el que envía mensajes a la cola y son los hijos los que los reciben.

Además, los hijos se comunican con el ilustrador mediante tuberías. La tubería fd[1] es en la cual escribe el hijo un mensaje que contiene el status y el pid del proceso y lo lee el ilustrador, mientras que en fd[2] el ilustrador escribe “Continue” indicando que ya ha recibido los datos y que los hijos pueden continuar tras la lectura de dicho mensaje.

Finalmente hemos querido incluir los semáforos que protegen una zona crítica del código, en el cual los hijos quieren acceder a la estructura de datos de sort. Se puede ver como los hijos para acceder a dicha estructura primero tienen que hacer un down del semáforo y al salir de la zona crítica realizan un up. De esta manera

hemos querido ilustrar lo mejor posible los mecanismos de comunicación y sincronización entre los distintos procesos.

**b. Realizar un diagrama que muestre el diseño del sistema, incluyendo los distintos componentes (procesos) y sus jerarquías, así como los mecanismos de comunicación y sincronización entre ellos.** *(1,00 ptos.)*

A la hora de implementar el proyecto decidimos ir siguiendo las pautas especificadas en el PDF pues nos pareció más sencillo ya que en cada paso se especifica lo que hay que se debe hacer.

Paso 1: Consistía en transformar la memoria estática en memoria compartida lo cual no tenía mucho misterio. Creamos el segmento de memoria, lo truncamos debidamente, lo enlazamos y llevamos a cabo las respectivas comprobaciones de errores. Nos aseguramos además de liberar la memoria reservada al final de los programas.

Paso 2: Llevamos a cabo la creación de un trabajador único, en este caso creamos un nuevo hijo para cada tarea distinta dentro de cada nivel. El único hijo se encargaba de ejecutar la tarea y finalizar, el padre lo espera y muestra el estado del sistema.

Paso 3: En esta caso crearemos trabajadores múltiples pero no serán reutilizables ya que cada uno realizará una tarea. El código de estos hijos se ejecutará dentro del bucle que recorre los niveles y la tarea asignada a cada uno la habremos guardado previamente en una variable al crearlos, tras finalizar todas las tareas de cada nivel se imprime el estado del sistema. Cada hijo realizara el task correspondiente y finalizará, el padre recogerá a todos los hijos por lo que hará un wait por cada uno de ellos e imprimirá el resultado final del sistema.

Paso 4: En este paso tenemos que llevar a cabo la implementación de la cola de mensajes. Creamos tantos hijos como tareas haya en cada nivel. El padre por cada nivel ejecutara un for en el que enviará en cada pasada un mensaje mediante la cola con las tareas a realizar de cada nivel y ejecutará otro for para crear tantos waits como hijos haya. Cada hijo leerá un mensaje de la cola, realizará la correspondiente tarea y finalizará correctamente. Se imprimirá el estado del sistema en cada nivel y al finalizar.

Paso 5: Este fue sin duda el paso más laborioso pues es hay que implementar distintos manejadores y conseguir que los trabajadores sean reutilizables. En el proceso principal creamos los trabajadores principales al inicio y comprobamos que, si son los hijos almacenen su pid en un array, pues luego los necesitaremos.
El código del hijo consiste en un bucle infinito en el cual recibe un mensaje de la cola y llama a la función solve indicando adecuadamente el nivel y tarea a realizar, cuyos datos obtiene del mensaje recibido. Una vez ha realizado la tarea, accederá a la

estructura sort y a su vez a la estructura del task correspondiente para marcar el estado como completado. El acceso a esta zona se realizará de forma segura mediante semáforos, asegurando la exclusión mutua. Una vez se haya llevado a cabo la tarea mandará la señal SIGUSR1 al padre para indicárselo y comenzará a realizar una nueva tarea de ese mismo nivel hasta que se completen todas y reciban la señal SIGTERM.
El código del padre irá recorriendo los niveles y por cada uno de estos sus respectivas partes. Cada nivel y parte serán almacenadas en una estructura mensaje que hemos creado para mandarla mediante la cola de mensaje a los hijos y así sepan qué tarea deben ejecutar. Esta decisión de diseño nos llevó bastante tiempo tomarla pues al principio estábamos intentando enviar el mensaje de la cola como un char en el que escribíamos dos números, el nivel y la parte. Sin embargo, cuando el hijo recibía el mensaje intentabamos hacer un atoi para poder obtener los dos números de forma separada pero no conseguimos que funcionará por lo que al final tuvimos la idea de crear esta estructura de mensaje. Una vez que dentro del nivel se hayan enviado todos los mensaje con las respectivas partes para resolver todos los taskas el padre realizará un sigsuspend. Estará a la espera de la señal SIGUSR1 que le enviará cada hijo cuando termine. Cada vez que la reciba llamará al correspondiente manejador, en el cual tenemos una flag a 1, recorrerá todas las partes de cada nivel comprobando que todos los tasks estén completados, de no ser así la flag cambiará a 0. A la hora de realizar esta comprobación hemos implementado un semáforo para proteger dicha sección crítica y asegurar la exclusión mutua para el correcto funcionamiento del programa. El do while del sigsuspend estará comprobando la flag y no saldrá del bucle hasta que la flag sea 1, indicando que todos los tasks de ese nivel han sido completados. Una vez haya hecho esta comprobación imprimirá el estado del sistema y comenzará nuevamente a enviar mediante la cola de mensajes las correspondientes partes del siguiente nivel a los hijos. Cuando el algoritmo se haya completado, el padre enviará la señal SIGTERM a todos sus hijos que ejecutarán el correspondiente manejador, el cual hará que los hijos liberen los recursos y terminen correctamente. El padre los recogerá a todos y finalizará.

Parte 6: Al comenzar este apartado nos dimos cuenta que nuestro programa hasta el momento repetía gran cantidad de código a la hora de liberar los recursos en caso de error. Por lo tanto lo primero que hicimos fue crear una función a la que llamamos freeAll que se encargaba de terminar la ejecución del programa correctamente llevando a cabo la liberación de los recursos. Implementar este paso en sí no fue muy complicado pues era básicamente añadir otra señal y manejador a lo que ya teníamos. Los procesos hijos al recibir la señal les hemos indicado que la ignoren mientras que el padre si la captura ejecutará el debido manejador. Éste enviará la señal SIGTERM a todos los hijos, lo que producirá que terminen su ejecución, y el padre también liberará todos sus recursos y finalizará.

Parte 7: Esta parte también fue una de las más complicadas a la hora de realizarlo puesto que no supimos muy bien cómo resolverlo al principio. Al principio pensamos en hacer únicamente 2 tuberías para todos los programas, pero acabamos

descartando esta idea puesto que pensamos que se podría perder parte de la información necesaria para la impresión del estado de la ordenación además de que podría complicarnos bastante el código. Al final decidimos hacer un par de tuberías por trabajador, que son las que se conectarán con el proceso ilustrador.
En los procesos trabajadores creamos una alarma que se enviará cada segundo, cuando ocurra, el proceso enviará al ilustrador su pid, estado, nivel y parte de trabajo (si es que lo está realizando), inicio y final de la parte. A continuación esperará por la tubería la confirmación del ilustrador de que ha impreso el estado y puede continuar trabajando. Por último volverá a establecer una alarma en un segundo y seguirá con la siguiente tarea asignada.
El ilustrador es el último de los procesos hijos que se crea, establece el manejador de la señal SIGINT para ignorarla, tras cerrar las tuberías de entrada/salida que no le corresponden entra en un bucle while infinito en el que esperará que todos los trabajadores le envíen su estado de trabajo. Preparará una ilustración que consiste en el estado actual del vector a ordenar y una lista con todos los procesos en el que se incluirá su pid, su estado, nivel, parte, inicio y final del trabajo que están realizando. Por último enviará un mensaje de confirmación a todos los trabajadores para que puedan continuar su ejecución y volverá a empezar. Este proceso se repetirá de manera infinita hasta que reciba una señal SIGTERM por padre del proceso principal, tras lo cual liberará todos los recursos y finalizará correctamente.