INFORME PROYECTO SOPER

Álvaro Castillo García

Pablo Ernesto Soëtard

Universidad Autónoma de Madrid

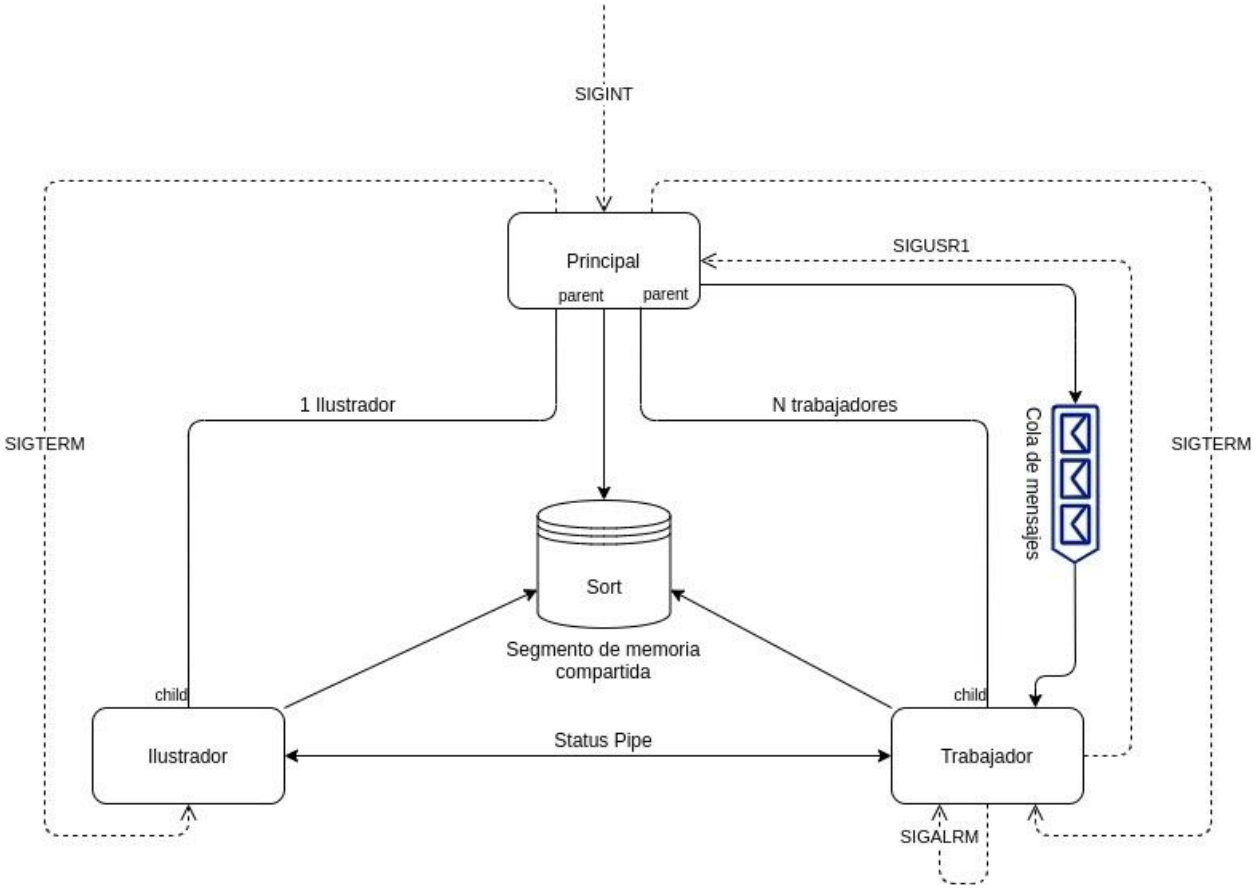
Escuela Politécnica Superior

Ingeniería Informática bilingüe, grupo 2292

Sistemas Operativos

ÍNDICE

APARTADO A



En este diagrama observamos el diseño del sistema creado.

APARTADO B

El sistema creado consta de tres tipos de procesos.

Por una parte se encuentra el proceso principal que se encargará de iniciar todos los componentes comunes y de crear al inicio los trabajadores y el ilustrador. Tras esto mandará a los trabajadores las tareas mediante una cola de mensajes.
Se ha implementado el apartado opcional, consiguiendo así que las tareas que se envíen no sean necesariamente del mismo nivel. Cada vez que recibe la señal SIGUSR1 comprobará si se puede incluir en la cola de mensajes alguna tarea más.
Finalmente, cuándo haya comprobado que se han completado todas las tareas, enviará la señal SIGTERM tanto a los trabajadores, como al ilustrador, los esperará y liberará o cerrará los recursos correspondientes.

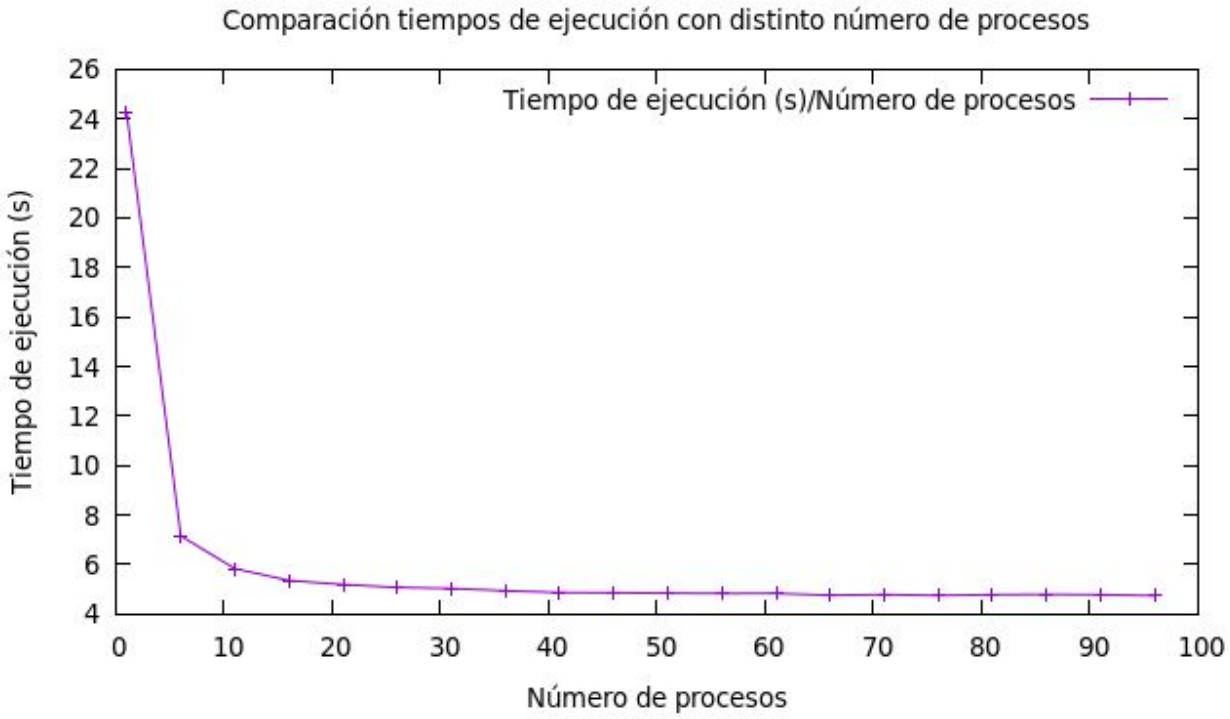
Por otra parte están los procesos trabajadores. Estos, primero crean y se ponen una máscara de señales para ignorar SIGINT, cierran los extremos correspondientes de los pipes para poder comunicarse con el proceso ilustrador y ponen una alarma de 1 segundo. A continuación entrarán en un bucle que realizarán hasta que reciban la señal SIGTERM. A su llegada simplemente liberarán los recursos y acabarán.
Dentro del bucle el trabajador leerá un mensaje de la cola, recibiendo una tarea a realizar. Modificará el estado de la tarea y la realizará. Cuando la complete, cambiará de nuevo el estado de la tarea y enviará al padre la señal SIGUSR1 avisando de que ha acabado de procesar su tarea.
Mientras que ejecuta el código del bucle, le puede llegar la señal de alarma SIGALRM, entrando en ese momento en el manejador correspondiente. En este manejador el proceso enviará por un pipe al ilustrador, la parte y el nivel que está tratando de ordenar, así como su pid y esperará a recibir la respuesta CONTINUA. Cuando la reciba, se pondrá una nueva alarma de 1 segundo y saldrá del manejador.

Por último se encuentra el proceso ilustrador, que al igual que los trabajadores creará una máscara ignorando SIGINT y cerrará los extremos correspondientes de los pipes. Llamará entonces a la función ilustrador(), dónde quedará en un bucle infinito del que sólo saldrá cuando reciba la señal SIGTERM o alguna señal no capturada. A la llegada de SIGTERM liberará recursos y acabará.
En el bucle infinito leerá un mensaje del pipe de entrada y si el nivel y la parte son válidos imprimirá en pantalla la información del sistema. Finalmente escribirá en el pipe de salida el mensaje CONTINUA para que el proceso que haya enviado anteriormente su estado, lo lea.

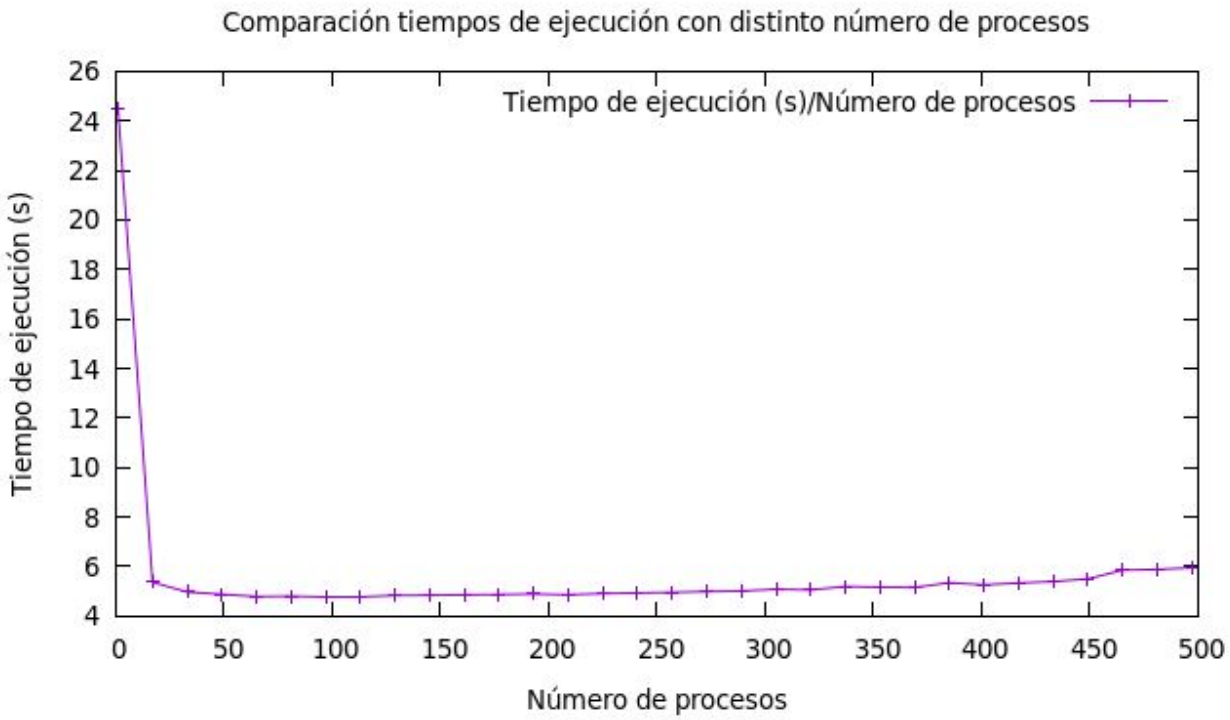
No creemos que haya limitaciones, pero pensamos que se podría mejorar el sistema. En vez de ser el usuario el que introduzca un número de procesos y niveles, se podría crear un programa que encontrase la cantidad óptima de ambos valores para realizar la ordenación lo más rápido posible.

La parte que más nos ha costado ha sido el implementar los pipes entre el ilustrador y los trabajadores. Al principio pensamos crear nºprocesos*4 descriptores de fichero para tener dos pipes por cada proceso. Pero de esta manera superábamos el límite de descriptores abiertos por el sistema operativo. Así que luego nos dimos cuenta de que podíamos crear sólo dos pipes, al inicio del código del padre, ya que se iban a copiar cuando llamáramos a la función fork().
Tras realizar esta implementación hemos tenido varios problemas con la lectura del mensaje por parte del ilustrador y del envío por parte del trabajador.
El proceso trabajador envía por el pipe una estructura con el número de parte y de nivel en el que está trabajando, así como su id, pero surge un fallo. Si al trabajador no se le ha asignado tarea y le llega la señal de alarma, enviará datos incompletos o erróneos al ilustrador. De esta manera hemos tenido que realizar varias comprobaciones en el ilustrador, para que cuando reciba un mensaje verifique los datos y acceda a la estructura sort sólo cuando haya leído datos razonables.
Por otro lado, al realizar la escritura en el pipe, se crea una condición de carrera, así que hemos tenido que añadir un semáforo. De esta manera los procesos escriben completa su estructura correspondiente en el pipe.

APARTADO C



Tras ejecutar el sistema multiproceso sobre el DataLarge.dat, con 10 niveles y con distinto número de procesos, en este caso, del 1 al 100 para mayor claridad. Podemos observar como hay una rápida disminución del tiempo de ejecución entre el 1 y el 10, y que a partir de ahí los tiempo bajan muy lentamente, esto se debe a que el ordenador tiene un número limitado de cores, por lo que, llegado un cierto punto, no todos los procesos se ejecutan en paralelo, sino que es el planificador del sistema el que va intercalando su ejecución. De ahí que se observe un límite en torno a los 4 segundos.



Esta gráfica se ha obtenido tras ejecutar el sistema multiproceso sobre el DataLarge.dat, con 10 niveles y con distinto número de procesos, en este caso, del 1 al 512. Como en la gráfica anterior, se puede observar una rápida caída de los tiempos entre 1 y 10, y después lentamente va disminuyendo, hasta que a partir de 300 procesos, el tiempo empieza a aumentar poco a poco de nuevo. Esto se debe a que, al aumentar la cantidad de procesos, aumenta el número de procesos que el padre debe crear y liberar, de ahí el aumento del tiempo.