Julio mario monterrosa paternina

c.c 1038123206

Informe lab. 2.

Estrategia:

El lab comienza en el main, donde tanto para el modo batch como para el modo interactivo se tiene que leer línea por línea de la entrada estándar y del archivo que se le pase como parámetro al ejecutable del wish respectivamente, por lo cual uso la función getline(…) y guardo ese string en una variable a la cual se le quita el '\n' del final y se reemplaza con '\0' con la función limpiarSaltoln(char *) que toma como parámetro la línea, y que además devuelve 1 si la línea no es vacía y 0 si lo es. Si no es vacía se la paso como parámetro a la función pillarComandos(char *), dentro de esta función lo primero que se hace es averiguar el número de comandos separados por '&', para que cuente, el comando tiene que ser válido, así que se hace una copia de la línea porque se usará mucho la función strtok(..) y strsep(..) que destruyen la línea a separar, entonces se separa por '&' y cada pedazo resultante se tiene que verificar que no sea una secuencia de solo espacios vacíos, strtok(' ') (espacio en blanco) sirve para tal fin porque devuelve NULL si el pedazo de línea que se le pasa son solo espacios vacíos. Una vez se tiene el número de comandos que luego servirá para hacer varios wait(..) para los comandos paralelos, se procede a despedazar la línea con strsep('&'), cada pedazo es un comando, a ese comando se le tiene que determinar si lleva redirección o no, y si esta es correcta. Se asume en principio que todos los comandos llevan redirección, y se usa un vector 'pedazos' cuya primera posición llevará la parte a la izquierda del signo '>' y la segunda posición la derecha, la tercera posición se le pone un NULL por si algo. El primer pedazo extraído, el de la posición 0 del vector es el char* del comando y sus argumentos, por lo cual se guarda directamente en el vector luego de verificar que no sea vacío y el segundo pedazo, el que va a la derecha del '>' es el nombre del archivo donde va a quedar la salida del programa, se debe verificar que sea solo un nombre de archivo por lo cual se hace una copia de dicho pedazo y se separa con strtok(' ') (espacio en blanco) y se cuentan los pedacitos, que deber ser solo 1, de lo contrario hay un error de formato en redirección, también lo hay si se cuentan mas de dos pedazos separados por ´>´ o si hay varios '>' consecutivos, caso en el que la función strsep('>') devuelve '\0'. Nótese que el vector de pedazos comienza con todos los valores en NULL, entonces luego de pedazos se puede saber si el comando es válido porque la posición 0 debería ser diferente de NULL y no se pilló ningún error de formato de redirección, también se puede saber si hay redirección o no porque el vector de pedazos en la posición 1 es diferente de NULL y de carácter EOF. Una vez que el proceso anterior se completó y el char* del comando y sus argumentos quedaron en la posición 0 del vector, se debe hacer el conteo de argumentos en dicho char* (el nombre del comando cuenta: 'ls', 'pwd', etc) por lo que se hace una copia para después y se separa el char* en espacios en blanco, con el numero de argumentos se inicializa el vector de argumentos del tamaño del conteo más 1 para incluir NULL en la última posición y se procede a despedazar el la copia y se guardan los argumentos en el vector. Ya con todos los argumentos se empieza a mirar si se trata de un comando integrado comparando la posición 0 del vector de argumentos (contiene el nombre del comando: 'ls', 'pwd', etc) usando la función strcmp(..) con 'exit', caso en que los argumentos no deben ser mas de 1 o será un error; luego se

compara con ‘path’, donde se libera la memoria dinámica usada por el vector mypath con la función liberarmd() que le hace un free(..) a cada posición de la variable mypath y luego a la misma variable mypath, después de liberarla se lleva a NULL y se le hace un realloc(..) de tantas posiciones como argumentos haya, ya que mypath en la última posición también tiene NULL y por último se rellena con los demás argumentos que son rutas; luego se compara con ‘cd’, comando que no debe llevar mas de una ruta como argumento de lo contrario es error; por ultimo si el comando no fue ninguno de los anteriores, entonces es un comando externo. Para tal fin se usa la función lanzarProceso(int, char*, char**, char*) que retorna 1 si encuentra el comando, 0 si no, y cuyo primer parámetro es si hay o no redirección, el segundo es la concatenación de una ruta más el nombre del comando (argumentos[0]) o solo el nombre del comando: ‘ls’, ‘pwd’, etc. El tercero es el vector con todos los argumentos, y el cuarto es el nombre del archivo para la redirección en caso de que la haya, en esta función se hace un access(..) para saber si el comando se encuentra, si es así crea un proceso hijo con fork() y lo transforma con execv() no sin antes mirar si hay redirección y en tal caso redirigir la salida estándar y la salida de error estándar al archivo especificado en el cuarto parámetro usando las funciones open(..) y dup2(..), después se hace el execv() para transformar el proceso hijo y si eso falla se imprime el error, se libera la memoria dinámica con liberarmd() y se mata al proceso hijo con exit(1). Primero se lanza el proceso con la ruta local, que es llevándole al segundo parámetro de lanzarProceso(..) la posición 0 del vector de argumentos, es decir, no se le concatena ninguna ruta, así el access() y el excecv() que están dentro de lanzarProceso() lo reconoce como un comando local, si no se encuentra el comando se procede a recorrer cada ruta dentro de mypath y a concatenar cada ruta con el comando en la posición 0 del vector argumentos, esa concatenación se le pasa como segundo parámetro a lanzarProceso(). al terminar de recorrer las rutas, si el comando no se encontró entonces eso es un error. Cada comando separado por ‘&’ se ejecuta de manera paralela puesto que no se espera a que termine si no que se lanza el proceso y se procesa el siguiente comando. Al final, luego de lanzar todos los comandos de una línea se hacen tantos wait(NULL) como numero de comandos se haya contado, sin importar que los comandos hayan terminado puesto que con la función wait() no hay problema. Y ya.

Reto

Para mí el reto fue la redirección porque no tenía idea de que había funciones que la facilitaban, pensé que la tendría que programar desde 0 y no tenía ni idea de como desviar la salida de un ejecutable de esos a un archivo cuyo nombre era pasado en la línea. Pero ya con el ejemplo que usted nos dio eso fue muy fácil. Después me pasó el problema de las rutas, que no leía nada en local, pensé que estaba correcto así por lo que dice en la guía, que había que buscarlo en cada ruta registrada y ya no dice mas nada. La solución fue dejar de concatenar el ‘/’ al principio de cada ruta que se pasaba con el comando path porque en ese caso se está buscando en el directorio raíz, y también mirar primero en local sin concatenarle nada al argumento.

Diferencias con C

Las de siempre. En otros lenguajes seguro se habría facilitado mucho lo de extraer los pedazos o los argumentos con alguna función que solo sea pasarle la línea y te devuelva el vector bien bonito de una sin dañar la línea, seguro la hay.