

At the very start of main, there are some strings that the user should set depending on the name of their data file and what they want the names of the output files to be:

```
char* inname = "d1.dat";  
char* fname1 = "half1";  
char* fname2 = "half2";  
char* outname = "sorted";
```

The first two pipes and children are created in the following manner:

```
int* pipe1 = createpipe();  
int child1 = createchild();  
if(!child1) {  
    childsort(pipe1, fname1);  
}  
close(pipe1[0]);
```

The createpipe() function returns an array of two integers which are the read and write file descriptors of the pipe that was made. This array is created using malloc so pipe1 should be freed later on in main. The childsort function is only called by the first two children which act as sorters, and it reads their pipe until EOF into an array of strings, sorts the array, and writes it to disk.

```
int* counts = splitfile(inname, pipe1[1], pipe2[1]);
free(pipe1);
free(pipe2);
waitpid(child1, NULL, 0);
waitpid(child2, NULL, 0);
```

The above code calls `splitfile()`, which splits the data file in a parallel manner and sends half to each pipe. This function returns an array of integers, much like `createpipe()` does, but this time the integers are the sizes of the two halves. After this, pipes one and two are not used, so their pointers can be freed (their file descriptors are already closed by this point). Then we wait for the children to finish by calling `waitpid()`.

```
int* pipe3 = createpipe();
int child3 = createchild();
if(!child3) {
    childmerge(pipe3, outname, counts[0], counts[1]);
}
close(pipe3[0]);

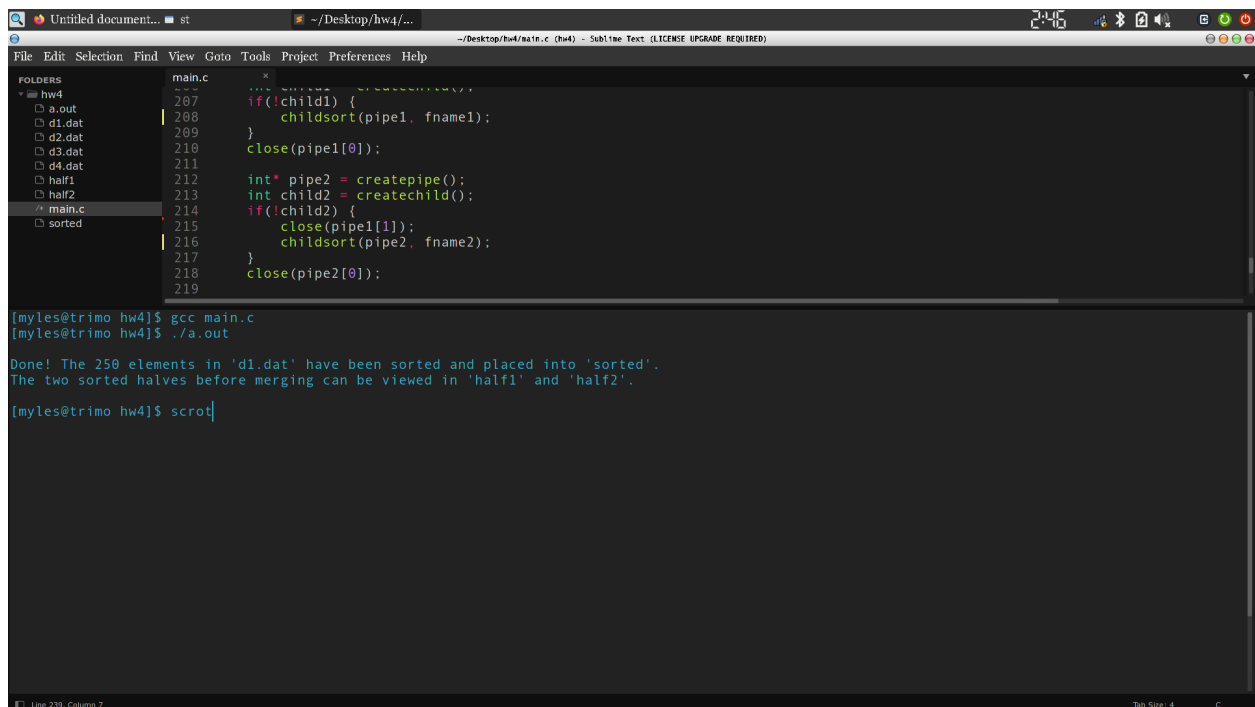
sendformerge(pipe3[1], fname1, fname2);
free(pipe3);
waitpid(child3, NULL, 0);
```

This code creates our third pipe and child, except this one calls `childmerge()` instead of `childsort()`. Inside `childmerge()`, the child's read pipe is read until `counts[0]` (or EOF) and stored in string array one, and then the read pipe is read until `counts[1]` (or EOF) and stored in string array two. These two arrays are sent to the `merge()` function, which merges them and returns a string array of size `counts[0] + counts[1]`,

which was the original size of the input file. This third array is written to disk, and the memory allocated by all three arrays is freed by calling the `freearr()` function:

```
void freearr(char** arr, int size) {  
    for(int i = 0; i < size; i++) free(arr[i]);  
    free(arr);  
    return;  
}
```

Once the program is done waiting for `child3`, the final sorted and merged file should be written to disk. Tarball of source code is attached.



```
File Edit Selection Find View Goto Tools Project Preferences Help  
FOLDERS  
  hw4  
    a.out  
    d1.dat  
    d2.dat  
    d3.dat  
    d4.dat  
    half1  
    half2  
    main.c  
    sorted  
main.c  
207  if(!child1) {  
208      childsort(pipe1, fname1);  
209  }  
210  close(pipe1[0]);  
211  
212  int* pipe2 = createpipe();  
213  int child2 = createchild();  
214  if(!child2) {  
215      close(pipe1[1]);  
216      childsort(pipe2, fname2);  
217  }  
218  close(pipe2[0]);  
219  
[myles@trimo hw4]$ gcc main.c  
[myles@trimo hw4]$ ./a.out  
Done! The 250 elements in 'd1.dat' have been sorted and placed into 'sorted'.  
The two sorted halves before merging can be viewed in 'half1' and 'half2'.  
[myles@trimo hw4]$ scrot
```