Scaletta Gioele ID: 811106170

CSCI4730/6730 – Operating Systems

Project #1: Multi-process and IPC

Finding offset and number of bytes to read for each process

In the submitted program the size that each process has to read is calculated dividing the total size of the file by the number of processes (*numJobs*). In order to handle the additional bits (remainder of the division between size of file and n of processes), the last process will have a slightly different size to read. I have decided to calculate the number of bits to read for the last process with the following formula:
Size (that has to be read)= filesize – (numJobs-1)*filesize/numJobs.

NB: The offset of the last process will be equal to (numbJobs-1)*filesize/numJobs. So the Size for the last process will be equal to the filesize minus its offset.

Creation of child processes and pipes to enable them to communicate with the father

Using a for loop the command *pipe* (before) and *fork* (after) will be executed as many times as many processes have to be created (so that for every child process there will be a pipe). Inside the loop, after each *fork*, every child will write in the pipe a structure of type *count_t* after having called the function *wordcount*. The father will save in an array named *plist* the offset, the size to read and the pipe array for each pid of each child after each fork. The child, after the write, will return so that it will stop iterating and it won't do any wrong *fork* thus avoiding the creation of zombies processes.

Waiting for child processes to finish and handling of the possible crashes

The *wait* will be inside a while loop. The loop will be escaped only when the counter, which counts the number of processes which have accomplished the counting without crashing, will equal the total number of processes. When the first (finishes before) process reaches the wait, its status is checked through *WIFEXITED(status)*.

If the process has crashed a new process and a new pipe are created. After the fork the new child will execute again *wordcount* using the same offset and size to read of the crashed child (they have been saved in *plist*) and the father will upload the pid in *plist* with the one of the new child. If the new child with the same task crashes again, when he reaches the wait the just descripted procedure will be executed again.

If the process has not crashed the total number of characters, words and lines read will be updated adding the result sent through the pipe by the process who has just reached the *wait.* The pipe will be read using *read.* (The pipe array is obtained searching in the array *plist* the pid). And the counter is incremented.

Conclusion

The above described implementation allows the task to be divided between more than one process, making the whole reading faster. Knowing in advance that the only need of communication will consist in passing the result from child to father, not involving other processes, the use of pipes as IPC should be convenient. The method to handle the crashes allow to create a new process as soon as the crashed process reaches the wait thus avoiding any waste of time. Moreover, when a crash happens and a new process is created, the wait will handle the first available process not having to wait for any specific process to finish, avoiding waste of time. For instance, if process number 1 crashes, the program creates another process number 3 and then wait for the first process which finishes which can be number 2 (for example) and not necessarily number 3.