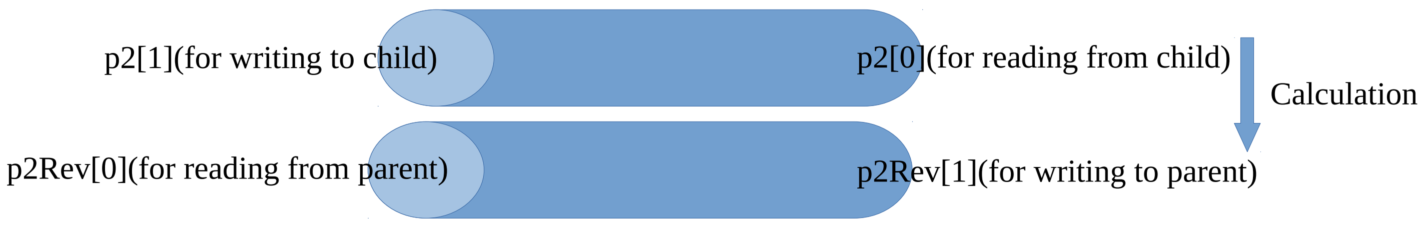
CSE344 HW3 REPORT

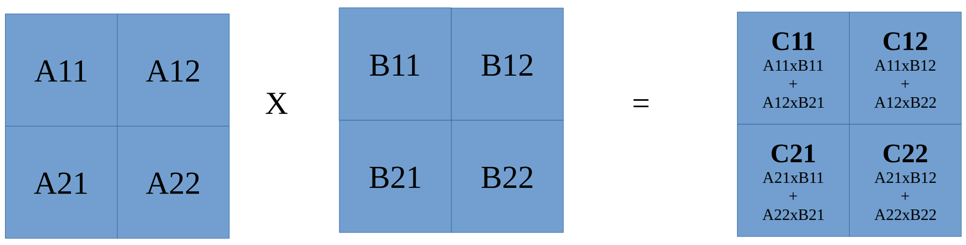
In this homework we're asked to implementing parallel matrix multiplication using pipes. For that purpose, I take two input file and 'n' which is determine read character number. 'n' is given parameter but number of needed characters from an input determined by 2^n x 2^n. This number, 2^n is the one dimension of the matrix. After take this number and file, I created pipes. To create a bidirectional pipe we must mus two pipe. One for sending and the other for receiving. Example structure for process 2 shown below:



p2[1](for writing to child)
p2[0](for reading from child)
Calculation
p2Rev[0](for reading from parent)
p2Rev[1](for writing to parent)

With this pipes, data will flow from parent to child and child to parent after calculation.

After that I opened files and readed character from these files. Number of read character is 2^n x 2^n. I take these values on single dimensional char arrays. Before opening second file, I checked if they're same file. If so, I used dup syscall to create copy of first file description and seeked file to the beginning. Now we have arrays but not the quarters. So, I dynamically created char arrays for each quarter using malloc. After that, I coppied corresponding elements to these quarters from matrixA and matrixB using memcpy. Since quarters is ready, we can fork.



If we think in case of process 2 which calculates C11. After creating process first I created result array for that case C11 which is single dimensional integer array and I decreased dimension to half since we deal with a smaller size matrix. Also created quarter size integer arrays to store the information that will be come from parent through pipe. For child we have needed quarters. After that I closed unnecessary files which not be used throughout process. Writing end of p2Pipe and reading end of p2Rev and also reading end of barrier are unnecessary.

Now we can take quarters from pipe. I taked quarter elements one by one using p2Pipe read end and dimension. First quarterA1 and quarterB1, secondly quarterA2 and quarterB2. When one quarter filled up, other starts to reading from pipe. Since I used only single dimensional arrays, I generally used nested for loops to keep track of matrix structure. In a for loop 'i' and 'j', (i*dimension)+j will give us the current element. After reading all quarters from parent process, I send each relevant quarters to standard matrix multiplication. After the process of multiplication done, I collected two result and sum them and assigned to C11. Since calculation is done we can say that P2 is reached the barrier. Now we can close writing end of barrier to let P1 to continue. From there We have the result of array of single quarter. I sended elements of result quarter matrix one by one. At the end of process, I closed remaining pipe file descriptors and exited process by success. This is how child process quarters and calculate result. But child cannot proceed and blocked if parent don’t send any data.

Since after fork parent continues I can use the quarters that created above to make calculation. Simply, I sended quarters elements one by one, first converting them to integer and then send them. The sending sequence is same as child’s needed. By that I provided same quarters to child. This is how a child handles readed data from parent and send it back to parent after calculation.

To reap child process and not to create any orphan or zombie child I added a loop at the end of parent process but before that I blocked SIGCHLD at whole process to create synchronous wait. Since I blocked SIGCHLD even if child send SIGCHLD it will not be delivered immediately and I can reap after all process done. After all process done, at the end of parent, I setted a suspend block mask that blocks nothing. By doing that, parent will be blocked until a signal is delivered. At that case if SIGCHLD arrives to parent, SIGCHLD handler(childHandler) take the execution and performs waitpid. By doing that for every reaped child, it decreases live child number. This process waits for any child and continues until there is no child. By doing that all childs status will be reaped, there will be zombie child. Status of each child will be taken and childs will be deleted from process table.

In any case of exit, I created a function will do all clean up processes. ‘cleanUp’ function closes all files and frees all allocated memory. To do that at first I created two global array and their global counters. After that to accomplish adding pointers and file descriptors to these array, I created two function that takes pointer or file descriptor and add it to array. I called these function after any file or pipe opening, or any case of memory allocation to record these values to clean after. I registered this cleanUp function via atexit syscall to run it in any case of exit. By doing that I released all resources that process using to system.

We also asked to implement in any case of Ctrl-C which is the case of sending SIGINT, exit all processes gracefully. Since we cannot know when signal occurs and which process catch it, I created two handler for that signal, one is for parent and other one is for childs. The sendChild handler handles SIGINT for child and acknowledges parent that SIGINT delivered start exiting process by sending SIGINT to parent. After parent receive this signal, it handles signal by using termParent function. This function terminates all child gracefully in sequence. First it checks if the process is exist or not by sending NULL signal. If the process exist then it sends SIGTERM signal to that particular child. Child handles SIGTERM by termChild handler. This handler simply calls exit syscall. Since we registered cleanUp function for clean all resources it handles this for every child. All childs exits by using this structure. But since not to create zombie processes we must reap child’s exit status. For that we use waitpid again to take status of childs. After reaping status of childs, parent calls exit and exits gracefully.