CSE344 – System Programming - Homework #3 Report

Fun with Pipes and Algebra

Student Name: Ozan Şelte

Student Number: 161044061

In this homework to be POSIX compatible with the latest standart I decided to define "_POSIX_C_SOURCE" as "200809", "_FILE_OFFSET" as "64" and "_GNU_SOURCE".

Singular Value Decomposition part is taken from Dianne Cook(dicook@iastate.edu) and edited for this project.

Problem #1:

The pipes must be bidirectionals and for synchronization there must be another pipe. All of them should closed before exits.

Solution #1:

I used 9 pipes, 0 for synchronization, odd numbers for Parent to Child and even numbers for Child to Parent. When a new process created it closes old pipes and unused pipes of itself.

```
if (-1 == close(pipes[0][RD])) {
    errExit("initProcesses, close, pipes[0][RD]");
}
if (-1 == close(pipes[i][WR])) {
    errExit("initProcesses, close, pipes[i][WR]");
}
if (-1 == close(pipes[i+1][RD])) {
    errExit("initProcesses, close, pipes[i+1][RD]");
}
```

```
size_t side = longSide / 2;
for (uint8_t i = 0; i < 4; ++i) {
    div[i] = (uint8_t **)xcalloc(side, sizeof(uint8_t *));
}
for (size_t i = 0; i < side; ++i) {
    div[0][i] = M[i];
    div[1][i] = &(M[i][side]);
    div[2][i] = M[i+side];
    div[3][i] = &(M[i+side][side]);
}
```

Problem #2:

Dividing a large matrix into four quarter matrices.

Solution #2:

I have choosen to use pointers for this.

Problem #3:

The files size is not limited so any file can be input but read and pipe has limited buffers.

Solution #3:

I use a dividing algorithm for this problem. If data is larger than a defined piece size limit, it will be covered as seperated pieces.

```
for (size_t i = 0; i < side; ++i) {
    size_t piece = PIECE_SIZE / sizeof(uint64_t);
    uint64_t *ptr = M[i];
    for (size_t left = side; 0 < left; left -= piece) {
        piece = MIN(left, piece);
        sendPiece64(fd, ptr, piece);
        ptr += piece;
    }
}
```

```
for (size_t i = 0; i < side; ++i) {
    size_t piece = PIECE_SIZE / sizeof(uint64_t);
    ssize_t pos = 0;
    for (size_t left = side; 0 < left; left -= piece) {
        piece = MIN(left, piece);
        for (uint8_t j = 0; j < 4; ++j) {
            receivePiece64(fd[j], M[j][i]+pos, piece);
        }
        pos += piece;
    }
}
```

Problem #4:

Parent should send all childs to their data but if matrices are large enough, when one child is working other ones must sleep. It is also happening again when parent receives data from all childs.

Solution #4:

I have used my dividing technique to send and receive each child same amount of numbers.

Problem #:5

We should handle signals, SIGINT and SIGCHLD, and prevent zombie processes. Also there was not any memory leaks.

Solution #5:

Waitpid done its job well with a do-while loop for zombue processes. I used signal blocks in critical sections which changing global pointers. With this method, if a memory location already freed, code knows it.

```
void signalHandlerChild(int signo)
{
    int errnoBackup = errno;
    canExit = CHILD_EXIT;
    errno = errnoBackup;
}
```

```
signalBlock();
C = multiplyMatrices(M, side);
if (-1 == close(wtFD)) {
    errExit("childMain, close, wtFD");
}
for (uint8_t i = 0; i < 4; ++i) {
    freeMatrix8(M[i], side);
    M[i] = NULL;
}
signalUnblock();
sendMatrix64(wrFD, C, side);
signalBlock();
freeMatrix64(C, side);
C = NULL;
```

Problem #6:

When SIGINT comes, all the processes exit gracefully.

Solution #6:

SIGINT already comes all of the processes, pointers all global, and not changed because of signal blocks; so if a pointer is not NULL it is goint to be freed.

```
for (uint8_t i = 0; i < 4; ++i) {
    if (NULL != M[i]) {
        freeMatrix8(M[i], mm->qSide);
        M[i] = NULL;
    }
}
if (NULL != C) {
    freeMatrix64(C, mm->qSide);
    C = NULL;
}
_exit(EXIT_SUCCESS);
```

```
static struct MatrixMult *mm = NULL;
static uint8_t **M[4] = {NULL, NULL, NULL, NULL};
static uint64_t **C = NULL;
static volatile sig_atomic_t isChildAlive = 0;
static volatile sig_atomic_t canExit = 0;
static volatile sig_atomic_t waitFD = -1;
```

Notes:

There is a commented define *__DEBUG__* in *funcs.h*. If it is uncommented, the program will print A, B and C matrices to the screen before Singular Values.

```
[myndos@halicarnassus:hw3]$ make && ./hw3 -i input1 -j input2 -n 2
make: Nothing to be done for 'all'.

--- BEGIN MATRIX ---
  92   49  224   62
  39  150  235  160
 140  233  122  215
 244  255   60  237
---  END  MATRIX ---


--- BEGIN MATRIX ---
 134   56  175  203
 202  245  107   75
  16   80   49  247
 217  243   62   18
---  END  MATRIX ---


--- BEGIN MATRIX ---
   39264    50143    36163    78795
   74006    96614    44310    80092
  114433   126930    68739    79899
  136595   138530    87619    87743
---  END  MATRIX ---

0 -> 354853.812
1 -> 46462.340
2 -> 104.230
3 -> 14879.018
```