

# CSE-344 System Programming Homework-2

Author: Hacı Hasan Savan  
1901042704

## Algorithm Design:

This is the algorithm that shows how the code functions step by step, ensuring seamless communication and synchronization between processes:

The algorithm begins by generating an array of random numbers based on the size of the command line argument provided by the user. Two FIFOs, named `fifo1` and `fifo2`, are then created for communication between Child1 and Child2 processes. Subsequently, Child1 and Child2 processes are spawned. Child1 opens `fifo1` for reading and waits for 10 seconds using the `sleep` function. Similarly, Child2 opens `fifo2` for reading and waits for 10 seconds. Meanwhile, the main process opens `fifo1` and `fifo2` for writing and sequentially writes the arrays of random numbers into them. Then, the main process writes the "multiply" command into `fifo2`. Child1 receives the array from `fifo1`, performs summation, and sleep 1 second here for synchronization and after that writes the result into `fifo2`. Child2 reads the array and the "multiply" command from `fifo2`, calculates the product, and waits for Child1's sum value to be written into `fifo2`. Upon receiving the sum value, Child2 prints the sum, product, and sum+product values to screen. Simultaneously, the main process runs in the background, printing "proceeding" every 2 seconds. The signal handler function in the background monitors the termination of child processes. Once the child processes finish their tasks, they exit.

## Interrupt Handling:

The `sigchld_handler` function is a signal handler designed to handle the `SIGCHLD` signal, which is emitted when a child process terminates. It enters a loop to continuously reap terminated child processes using the `waitpid()` function with the `WNOHANG` option, allowing it to return immediately if there are no terminated child processes. When a child process is reaped, it prints out the child process ID and its exit status. If the exit status indicates an error, the handler attempts to determine the other child process's ID and sends a `SIGTERM` signal to it to terminate it gracefully. The function keeps track of the number of terminated child processes by incrementing the `child_count` variable. If an error occurs during the termination signal sending process, it prints an error message and exits the program with a failure status. Overall, this handler ensures proper cleanup and management of child processes in a multi-process environment. If a child process exits with a failure (syscall fails like `read`, `close` etc.) the other child process will be exited as well because the two processes depend on each other.

### Bonus Sections:

- Zombie protection achieved by using `waitpid` in `sigchld_handler` function. If a function ends its job, it will absolutely be reaped.

```
// Function to handle SIGCHLD signal
void sigchld_handler(int signo)
{
    pid_t pid;
    int status;

    while ((pid = waitpid(-1, &status, WNOHANG)) > 0)
    {
        printf("Child process %d exited with status: %d\n", pid, WEXITSTATUS(status));
        if (WEXITSTATUS(status) != 0)
        { // it means that a child exited with an error
            if (pid == pid1) pid = pid2;
            else pid = pid1;
            if (kill(pid, SIGTERM) == -1)
            {
                perror("kill");
                exit(EXIT_FAILURE);
            }
        }
        child_count++;
    }
}
```

- Exit status of all process are printed in to the screen in all scenarios either success or failure:

```
result (sum + product): 20
Child process 8078 exited with status: 0
Child process 8079 exited with status: 0
proceeding...
```

```
read: Bad file descriptor
Child process 8113 exited with status: 1
proceeding...
Child process 8112 exited with status: 0
```

### Memory Leak Checks:

The code checked with **valgrind** if there is a memory leak or not. There is no memory leak:

These are the ss from valgrind for each processes (if you want to control it by self just run code with a valgrind keyword):

```
==28903== HEAP SUMMARY:
==28903==    in use at exit: 0 bytes in 0 blocks
==28903==   total heap usage: 2 allocs, 2 frees, 1,044 bytes allocated
==28903==
==28903== All heap blocks were freed -- no leaks are possible
```

```

==28906== HEAP SUMMARY: 1 heap blocks were freed -- no leaks are possible
==28906==    in use at exit: 0 bytes in 0 blocks
==28906== total heap usage: 4 allocs, 4 frees, 1,069 bytes allocated
==28906==
==28906== All heap blocks were freed -- no leaks are possible

```

```

==28905== HEAP SUMMARY: total heap usage: 2 allocs, 2 frees, 1,044 bytes allocated
==28905==    in use at exit: 0 bytes in 0 blocks
==28905== total heap usage: 3 allocs, 3 frees, 1,064 bytes allocated
==28905==
==28905== All heap blocks were freed -- no leaks are possible
==28905==

```

## Error Scenarios:

- All error scenarios that described handled.

## Example Runs:

with a 1000 random numbers (result is correct):

```

Array: 5 6 4 1 2 4 5 0 3 1 3 1 0 8 3 8 1 1 3 1 8 8 5 9 4 4 4 8 2 2 7 8 8 1 1 2 6 6 2 1 9 8 3 0 6 8 8 0 0 3 3 8 1 8 9 7 4 5 6 7 7
5 5 7 9 8 9 5 4 2 8 5 2 1 5 0 0 5 0 2 8 5 2 1 6 3 9 0 8 7 7 5 2 4 4 1 4 3 8 0 7 7 6 9 0 1 2 0 9 4 4 9 0 8 1 6 1 2 6 1 9 6 6 1 2 2
3 7 6 3 7 3 0 3 5 1 7 7 3 8 3 0 7 3 8 0 1 0 2 0 1 3 6 0 5 8 2 0 7 0 3 5 6 6 8 1 7 7 0 0 5 5 0 3 9 1 5 2 3 0 2 6 3 8 8 0 9 3 0 6
3 6 1 9 4 2 2 1 1 4 3 7 0 6 2 1 7 7 3 2 7 6 8 3 6 7 3 5 0 6 2 5 4 5 5 8 9 9 9 1 6 4 0 6 0 2 9 9 1 2 3 1 8 2 4 7 1 9 2 3 5 6 8 9 4
5 7 3 5 8 6 3 3 6 9 5 0 8 7 2 2 0 5 3 4 1 0 5 0 2 8 6 1 9 7 5 4 5 8 1 5 5 4 8 3 5 6 4 5 3 8 8 5 5 1 0 6 3 7 6 7 8 4 8 7 2 3 3 9
4 5 4 1 1 3 4 7 1 0 4 6 0 2 1 5 5 3 1 0 1 0 8 1 4 6 8 6 2 3 5 8 0 0 9 2 5 5 1 6 6 5 4 8 0 5 4 7 9 7 0 2 7 8 3 2 6 3 0 0 6 6 8 7 8
7 1 5 3 4 3 1 1 7 1 1 4 5 1 3 3 1 7 0 1 2 4 7 7 5 8 6 3 8 5 3 6 6 8 1 2 3 4 3 0 5 7 4 3 8 0 6 1 9 8 2 4 5 1 1 0 1 7 5 0 2 0 8 0
8 1 2 3 5 6 2 3 9 5 3 1 3 4 3 4 8 7 9 1 8 1 5 6 8 7 0 8 5 3 8 8 5 1 3 3 6 7 6 7 5 1 1 0 5 4 6 5 1 7 9 1 8 4 9 8 1 0 8 8 3 6 8 0
9 1 4 5 0 2 5 7 4 6 8 1 2 4 7 5 2 8 6 2 2 6 3 5 8 1 5 3 8 3 5 7 6 1 5 8 4 2 6 0 0 6 1 4 0 0 9 4 8 7 9 2 3 2 9 3 5 4 8 3 9 4 3 7
5 0 6 1 4 2 1 4 8 3 8 0 5 9 7 4 6 8 6 2 0 6 5 7 2 6 3 2 2 6 1 7 8 7 1 2 1 2 8 1 7 6 4 3 7 1 9 3 9 5 5 1 3 3 8 8 9 1 2 1 9 3 0 9 1
1 1 4 6 1 6 5 9 0 0 6 3 9 0 4 7 5 5 2 0 5 0 9 9 2 2 0 8 3 0 1 6 3 5 4 7 1 0 6 3 0 3 6 2 5 0 9 0 7 1 3 5 4 2 6 8 5 6 6 0 8 7 8 4
3 3 1 6 5 7 2 7 2 8 9 7 1 0 0 0 4 3 7 8 5 3 6 2 2 5 4 2 2 3 6 7 6 9 6 3 7 8 0 9 8 2 9 1 2 9 2 8 4 1 6 1 5 5 4 9 0 8 1 2 3 0 2 1 9
8 4 6 8 7 8 6 1 9 8 5 8 2 4 2 3 0 3 0 5 9 1 7 0 3 0 3 5 4 7 4 2 1 3 0 0 3 8 1 2 8 7 2 0 1 6 6 3 1 8 1 3 0 8 3 5 0 8 0 4 5 6 8 9
9 0 9 2 9 1 6 9 8 8 2 9 4 8 4 8 8 5 1 0 4 4 7 4 4 9 1 0 6 9 9 5 0 0 0 1 1 6 0 1 7 2 0 3 2 5 1 3 2 2 5 6 8 3 1 3 2 4 5 0 3 6 8 5 6
8 6 0 6 9 1 5 3 4 9 8 1 2 1 3 5 6 0 5 9 3 0 4 7 5 4 2 3 2 8 2 2 6 2 1 7 3 6 1 7 7 9 8 0 2 2 7 8 4 2 0 7 5 6 6 0 2 8 6 7 6 8 9 5
0 2 2 5 1 3 3 0 2 3 0 6 7 9 5 1 4 7 0 9 3 6 9 5 5 5 2 3 5 4 8 5 8 1 1 9
proceeding... 124
proceeding... 125 // printf("done!\n", sizeof(arr2)* ArrLength);
proceeding... 126 // close(fd);
proceeding... 127 // printf("child2 multiply commutative!\n");
proceeding... 128 if (close(fd) == -1)
Sum: 4250 129 {
Product: 0 130 perror("close");
Result (sum + product): 4250 exit(4);
Child process 9150 exited with status: 0
Child process 9152 exited with status: 0

```

```

hsnsvn@hsnsvn:~/Desktop/System Programming/hmw2/hmw2/SystemProgramming/hmw2$ ./hmw 5
Array: 6 0 1 7 0
proceeding... Median: 3
proceeding... Mode: 1
proceeding...
proceeding...
proceeding...
proceeding...
Sum: 14
Product: 0
Result (sum + product): 14
Child process 9465 exited with status: 0
Child process 9466 exited with status: 0

```

```

hsnsvn@hsnsvn:~/Desktop/System Programming/hmw2/hmw2/SystemProgramming/hmw2$ ./hmw 3
Array: 8 3 7
proceeding... 19 pid_t pid1, pid2;
proceeding... 20
proceeding... 21 // Function to handle SIGCHLD signal
proceeding... 22 void sigchild_handler(int signo)
proceeding... 23 {
Sum: 18 24 pid_t pid;
Product: 168 25 int status;
Result (sum + product): 186
Child process 9843 exited with status: 0 (id = waitpid(-1, &status, WNOHANG) > 0)
proceeding... 28 {
Child process 9844 exited with status: 0 ("Child process %d exited with status: %d\n", pid, WEXITSTATUS(status));
proceeding... 30 if (WEXITSTATUS(status) != 0)

```