CSE-344 System Programming Homework-2

Author: Hacı Hasan Savan
1901042704

Algorithm Design:

This 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 exit with a failure (syscall fails like read,close etc.) the other child process will be exited aswell because the two process depends each others.

Bonus Sections:

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

```c
// 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:

```
Child process 8078 exited with status: 0
Child process 8079 exited with status: 0
```

```
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:
==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:
==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):