Assignment3 Report

20181016 KwonYongmin

Assignment3.c

In main function, first, it opens the text file and changes corresponding file descriptor to file pointer, to use 'fgets' function. Next, it sets the buffer with size 256, which is used to read and write the data in pipes. And then, it makes two pipes. One is for sending the data to its direct child process and the other is for receiving the data from other process.

```
int fd;

if(argc != 2){
    printf("Usage: assignment3 <text file>\n");
    return 0;
}

fd = open(argv[1], O_RDONLY|O_SYNC);
FILE *fp = fdopen(fd, "r");
int fp_size = sizeof(fp);

const int max = 256;
char line[max];
char* exit_quote = "I'm exiting...\n";

int init_pipefd[2];
pipe(init_pipefd);
int final_pipefd[2];
pipe(final_pipefd);
```

In my assignment, processes send and receive the information of "number of the line" through the pipes. Because it is useless to use file pointer because each process has isolated memory, and it is also useless to use real contents of text file directly, because there may exist the same line.

```
int first_line = 1;
void* ptr = &first_line;
write(final_pipefd[WRITE], ptr, max);
```

So, after making two pipes, it writes integer value '1' to "init_pipe". And then by "fork()", it makes 5 processes, which have parent and child relationship each other. Each process invokes the function "handle_pipe". After finishing it, each process waits until its child process exits by "wait()" function, and then exits printing exit quote.

```
pid_t pid;
pid = fork();

if(pid>0){
    handle_pipe(final_pipefd, init_pipefd, fp, line, max);

    wait(NULL);
    printf("%d %s", getpid(), exit_quote);
    exit(0);
}else if(pid == 0){
    int second_pipefd[2];
    pipe(second_pipefd);

    pid_t pid2 = fork();
    if(pid2 > 0){
        handle_pipe(init_pipefd, second_pipefd, fp, line, max);

        wait(NULL);
        printf("%d %s", getpid(), exit_quote);
        exit(0);
    }else if(pid2 == 0){
        int third_pipefd[2];
        pipe(third_pipefd);

        pid_t pid3 = fork();
        if(pid3 > 0){
            handle_pipe(second_pipefd, third_pipefd, fp, line, max);

            wait(NULL);
            printf("%d %s", getpid(), exit_quote);
            exit(0);
        }else if(pid3 == 0){
            int fourth_pipefd[2];
            pipe(fourth_pipefd);
```

```
                pid_t pid4 = fork();
                if(pid4 > 0){
                    handle_pipe(third_pipefd, fourth_pipefd, fp, line, max);

                    wait(NULL);
                    printf("%d %s", getpid(), exit_quote);
                    exit(0);
                }else if(pid4 == 0){
                    handle_pipe(fourth_pipefd, final_pipefd, fp, line, max);

                    printf("%d %s", getpid(), exit_quote);
                    exit(0);
                }
            }
        }
    }
```

Assignment3.h

There is only one function “handle pipe”. It reads the data from “read_pipe”.

If the data from “read_pipe” is equal to final quote, it checks whether the current process is most child process. If it is, it writes exit quote to “write_pipe”. If not, it just writes final quote to “write_pipe”.

If the data from “read_pipe” is equal to exit quote “exit₩n”, it writes exit quote to “write_pipe” and then breaks the while loop.

Else, it iterates the “fgets” function as much as the number from “read_pipe”. After reading, it prints its process id and reading result.

Finally, it checks whether the next line of the file pointer is the end of the file or not. If it is, it prints the final quote and its process id, and writes the final quote to “write_pipe”. If not, it writes the integer value which is just 1 bigger than the integer from “read_pipe” to “write_pipe”.

```
void handle_pipe(int* read_pipe, int* write_pipe, FILE* current_fp, char* buffer, int bf_size, int is_last_child){
    char* final_quote = "Read all data\n";
    char* exit_quote = "exit\n";
    close(read_pipe[WRITE]);
    close(write_pipe[READ]);

    while(1){
        read(read_pipe[READ], buffer, bf_size);
        int iteration = *(int*)(buffer);
        if(strncmp(buffer, final_quote, strlen(final_quote)) == 0){
            if(is_last_child == 1){
                write(write_pipe[WRITE], exit_quote, bf_size);
            }else{
                write(write_pipe[WRITE], final_quote, bf_size);
            }
        }else if (strncmp(buffer, exit_quote, strlen(exit_quote)) == 0){
            write(write_pipe[WRITE], exit_quote, bf_size);
            break;
        }else{
            memset(buffer, 0, bf_size);
            rewind(current_fp);
            for(int i = 0; i < iteration; i++){
                fgets(buffer, bf_size, current_fp);
            }
            printf("%d %s", getpid(), buffer);
            char* result = fgets(buffer, bf_size, current_fp);
            if(result != NULL){
                int next_iteration = iteration + 1;
                void* ptr = &next_iteration;
                write(write_pipe[WRITE], ptr, bf_size);
            }else{
                printf("%d %s", getpid(), final_quote);
                write(write_pipe[WRITE], final_quote, bf_size);
            }
        }
    }
}
```

Program Structure Figure