Министерство образования и науки Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ Робототехники и комплексной автоматизации

КАФЕДРА Системы автоматизированного проектирования (РК-6)

ОТЧЕТ О ВЫПОЛНЕНИИ ЛАБОРАТОРНОЙ РАБОТЫ

Студент	Гусаров Аркадий Андреевич
Группа	РК6-53Б
Тип задания	Лабораторная работа №4
Тема лабораторной работы	Конкуренция параллельных процессов OS UNIX

Студент _____ **Гусаров А.А.**
подпись, дата *фамилия, и.о.*

Оценка _____

Москва, 2021 г.

Задание на лабораторную работу

Модифицировать исходную программу так, чтобы происходило тиражирование меток процессов справа-налево. Диапазон тиражирования и число процессов передаются в качестве параметров командной строки. Процесс, который полностью заполнил своими метками свой диапазон тиражирования – прекращается.

Код программы

Файл main.c:

```
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/wait.h>
#include <sys/timeb.h>

void clear_screen()
{
    unsigned char esc[11];

    // Установить курсор в левый верхний угол окна
    // Escape последовательность "Escape[n;mH" перемещает курсор в
заданное положение. Без аргументов n=0, m=0
    esc[0] = 27;
    esc[1] = '[';
    esc[2] = 'H';
    write(1, esc, 3);

    // Очищаем содержимое экрана

    // Escape последовательность "Escape[ n J", где n=2, очищает экран от
курсор и до конца окна, либо может
    // добавить пустые строки, чтобы экран был чистым. Без аргументов
n=0, m=0
    esc[2] = '2';
    esc[3] = 'J';
    write(1, esc, 4);
}

void go_to_x_y(int tx, int ty, char c)
{
    unsigned char esc[16];         // Escape-последовательность
    static unsigned char x_str[3]; // Положение курсора по x
    static unsigned char y_str[3]; // Положение курсора по y
```

```
    int i = 0;                      // Текущий индекс
escape-последовательности
    int j = 0;                      // Текущий индекс положений по x и y

    // Конвертируем координаты в текстовый формат
    if ((tx > 99) || (ty > 99))
    {
        tx = ty = 99;
    }
    if ((tx < 1) || (ty < 1))
    {
        tx = ty = 1;
    }
    x_str[0] = x_str[1] = x_str[2] = '\0';
    y_str[0] = y_str[1] = y_str[2] = '\0';

    sprintf((char *)x_str, "%d", tx);
    sprintf((char *)y_str, "%d", ty);

    // Escape последовательность "Escape + [ n ; m H" перемещает курсор в
заданное положение. Без аргументов n=0, m=0
    esc[i++] = 27;
    esc[i++] = '[';

    // Вводим координату по y в нашу escape-последовательность
    while (y_str[j])
    {
        esc[i++] = y_str[j++];
    }
    esc[i++] = ';';

    // Вводим координату по x в нашу escape-последовательность
    j = 0;
    while (x_str[j])
    {
        esc[i++] = x_str[j++];
    }

    esc[i++] = 'H';

    // Затираем escape-последовательность и выводим букву соответствующую
процессу
    esc[i++] = '\b';
    esc[i++] = ' ';
    esc[i++] = c;
    esc[i++] = 27;
    esc[i++] = '[';
    esc[i++] = 'K';
    esc[i++] = '\b';
    esc[i] = '\0';
```

```
    write(STDIN_FILENO, esc, i);
}

int main(int argc, char *argv[])
{
    int x = 1;
    int i = 0;
    int j = 0;
    int status = 0;

    int PROC_NUM = atoi(argv[1]);
    int *p_id = sbrk((PROC_NUM + 1) * sizeof(int));
    char *lead = sbrk((PROC_NUM + 1) * sizeof(char));

    int start = atoi(argv[2]);
    int distance = atoi(argv[3]);

    if (start - distance < 0)
    {
        exit(-1);
    }

    int p = 0;
    char bell = '\007';
    struct timeb proc_time[1];

    clear_screen();

    while (j < PROC_NUM)
    {

        // Ответвляем(форкаем) процесс. В итоге создается отдельный
процесс
        if ((p_id[j] = fork()) == 0)
        {
            usleep(PROC_NUM - j);

            // Запускаем для процесса цикл, внутри которого буква процесса
будет перемещаться в консоли
            while (x < distance)
            {

                // Перемещение буквы
                go_to_x_y(start, j + 1, 'A' + j);

                // Смотрим время процесса, на основе чего определяем,
пойдет ли буква дальше или останется на месте
                ftime(proc_time);
                if ((proc_time[0].millitm % (j + 'A')) != j)
```

```
                {
                    continue;
                }
                x++;
                start--;
                // Фиктивный цикл, чтобы добавить небольшой таймаут
                for (i = 0; i < 1000000; i++)
                    ;
                usleep(2000);
            }
            // Выход из процесса
            exit('A' + j);
        }
        j++;
    }

    // Ожидание выполнения всех процессов и нахождение тех, что прибыли первые
    j = 0;
    // wait возвращает p_id дочернего процесса, а статус определяется тем, что вернули через exit(status)
    while ((p = wait(&status)) != (-1))
    {
        for (i = 0; i < PROC_NUM; i++)
        {
            if (p_id[i] == p)
            {
                lead[j++] = (char)(status >> 8);
            }
        }
        write(STDIN_FILENO, &bell, 1);
    }

    // Вывод списка лидеров
    lead[j] = '\n';
    sleep(0.5);
    go_to_x_y(1, PROC_NUM + 3, '\n');
    write(STDIN_FILENO, lead, PROC_NUM + 1);

    exit(0);
}
```