Міністерство освіти і науки України

Харківський національний університет імені В. Н. Каразіна

Факультет комп'ютерних наук

Лабораторна робота №6

з навчальної дисципліни

«Операційні системи»

Виконав:
студент групи КУ-31
Нечипоренко Д.І.

Харків – 2022

Завдання

Задание №1

Написать функцию, которая выводит в стандартный поток вывода информацию о процессе. *Дополнительное задание*: предусмотреть аргумент, с помощью которого можно управлять той информацией, что выводится в стандартный поток вывода.

Задание №2

Создать один процесс — потомок. Продемонстрировать «непредсказуемость» алгоритма переключения процессов. С помощью функции time() (получение количества секунд, прошедших с начала эры *UNIX*, заголовочный файл time.h, вызов: time_t now = time(NULL);) заставить проработать программу заданное количество секунд (2, 3, 5, … можно указать с помощью макроопределения) и просчитать, сколько раз в каждом процессе выполнится тело цикла —

Задание №3 (или а, или б)

а). В программе функция fork может быть вызвана более одного раза. Ее можно вызывать как из родительского потока, так и из потоков — потомков. Родительский процесс содержит локальную переменную. Напишите программу, которая создает два под-процесса, а каждый из них, в свою очередь, свои два под-процесса. После каждого вызова fork в новом процессе увеличить значение локальной переменной, вывести на экран с помощью функции printf значение этой локальной переменной, ее адрес и идентификаторы родительского и дочернего процессов. Кроме того, каждый родительский процесс должен вывести идентификаторы своих дочерних процессов. Обратить внимание на очередность создания процессов при каждом запуске программы.

б). Програма принимает при запуске в командной строке натуральное число, означающее количество процессов-потомков, которые необходимо создать. В каждом дочернем процессе после запуска выводится сообщение и процесс (завершается или) уходит в бесконечный цикл. Основной процесс, закончив цикл создания процессов-потомков выводит на экран список работающих процессов и (при помощи последовательности вызовов команды ps выводит информацию о каждом из них или) удаляет их (kill). После работы программы проверьте работу процессов.

Задание № 5

С помощью функций fork, exec, wait создайте упрощенный аналог функции system() из прошлого занятия.

Задание №6

Напишите программу, создающую процесс — зомби и показывающую его наличие в системе, а также его исчезновение после вызова функции wait(). Для вызова команды ps с нужными опциями можно воспользоваться функцией system().

Задание №7

Рассмотрим простейший способ организовать взаимодействие процессов: родителя и потомка. Родительский процесс может передавать своему потомку некоторые данные с помощью аргументов одной из функций семейства exec(). В свою очередь дочерний процесс при нормальном завершении сообщает родителю свой код возврата. Рассмотрим задачу, в которой попробуем осуществить такое взаимодействие.

Написать программу, которая через аргументы командной строки получает натуральное число — количество «бросков», а через код возврата возвращает родителю количество точек, попавших в заданную область. Вторая программа запускает процесс — потомок, в него загружает эту программу, получает результат и с его помощью вычисляет требуемые характеристики.

Написать программу с одним родительским процессом и одним процессам — потомком.

Дополнительное задание — обобщить задачу на n потоков и усреднить результаты отдельных n «экспериментов». Параметры (количество бросков и количество потоков) передаются с помощью опций.

Код:

Task 1

```
#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <wait.h>


int main() {

    pid_t pid;

    pid = fork();


    if (pid == 0) {

        return 6;

    } else if (pid > 0) {

        system("ps -ax");

        wait(NULL);

        printf("After wait method call\n");

        system("ps -ax");

        return 0;

    }


    return 0;
```

}

Task 2

```
#include <stdio.h>
#include <time.h>
#include <unistd.h>

#define SECONDS 3

int main() {
    pid_t pid;

    printf("Forking started\n");
    pid = fork();

    int count = 0;
    time_t currentTime = time(NULL);

    if (pid == 0) {
        while (time(NULL) != (currentTime + SECONDS)) {
            count++;
        }
        printf("Child PID = %d. Count = %d\n", getpid(), count);
    } else if (pid > 0) {
        while (time(NULL) != (currentTime + SECONDS)) {
            count++;
        }
        printf("Parent PID = %d. Count = %d\n", getpid(), count);
```

```
    } else {
        printf("Error\n");
    }

    return 0;
}
```

Task 3

```
#include <stdio.h>
#include <unistd.h>
#include <wait.h>

int main() {
    setbuf(stdout, NULL);
    printf("Main PID %d\n", getpid());
    int local = 0;
    pid_t pid[2];
    pid[0] = fork();

    if (pid[0] == 0) {
        // child process
        local++;
        printf("\nLocal variable = %d\nAddress = %p\nParent PID = %d\nOwn PID = %d\n",
local, &local, getppid(), getpid());

        pid_t pidd[2];
        pidd[0] = fork();
```

```
        if (pidd[0] == 0) {
            // child of child
            local++;
            printf("\nLocal variable = %d\nAddress = %p\nParent PID = %d\nOwn PID = %d\n", local, &local, getppid(), getpid());
        } else if (pidd[0] > 0) {
            pidd[1] = fork();
            if (pidd[1] == 0) {
                local++;
                printf("\nLocal variable = %d\nAddress = %p\nParent PID = %d\nOwn PID = %d\n", local, &local, getppid(), getpid());
            } else if (pidd[1] > 0) {
                printf("Second Main IDs1 = %d IDs2 = %d\n", pidd[0], pidd[1]);
                wait(NULL);
            }
        }
    } else if (pid[0] > 0) {
        // parent
        pid[1] = fork();
        if (pid[1] == 0) {
            // child of parent
            local++;
            printf("\nLocal variable = %d\nAddress = %p\nParent PID = %d\nOwn PID = %d\n", local, &local, getppid(), getpid());
            pid_t pidd[2];
            pidd[0] = fork();
            if (pidd[0] == 0) {
```

```
            // parent of parent child
            local++;
            printf("\nLocal variable = %d\nAddress = %p\nParent PID = %d\nOwn PID = %d\n", local, &local, getppid(), getpid());
        } else if (pidd[0] > 0) {
            // child of parent child
            pidd[1] = fork();
            if (pidd[1] == 0) {
                local++;
                printf("\nLocal variable = %d\nAddress = %p\nParent PID = %d\nOwn PID = %d\n", local, &local, getppid(), getpid());
            } else if (pidd[1] > 0) {
                printf("Second Main IDs1 = %d IDs2 = %d\n", pidd[0], pidd[1]);
            }
        }
    } else if (pid[1] > 0) {
        printf("Main ID1 = %d ID2 = %d\n", pid[0], pid[1]);
        wait(NULL);
    }
  }

  return 0;
}
```

Task 5

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

```
#include <errno.h>
#include <unistd.h>

int my_system(const char *cmd_string) {
    pid_t pid;
    int status;

    if (cmd_string == NULL)
        return (1);

    if ((pid = fork()) < 0) {
        status = -1;
    } else if (pid == 0) {
        execl("/bin/sh", "sh", "-c", cmd_string, (char *)0);
        _exit(127);
    } else {
        while (waitpid(pid, &status, 0) < 0) {
            if (errno != EINTR) {
                status = -1;
                break;
            }
        }
    }

    return (status);
}



int main(void) {
    int status;
```

```
    if ((status = my_system("date")) < 0) {
        fprintf(stderr, "%s\n", "Error while invoking system()");
    }
    fprintf(stderr, "Exit code: %d\n", status);

    if ((status = my_system("nosuchcommand")) < 0) {
        fprintf(stderr, "%s\n", "Error while invoking system()");
    }
    fprintf(stderr, "Exit code: %d\n", status);

    if ((status = my_system("who; exit 44")) < 0)  {
        fprintf(stderr, "%s\n", "Error while invoking system()");
    }
    fprintf(stderr, "Exit code: %d\n", status);

    return 0;
}
```

Task 7

Parent

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

#define NUM_P_PROC 250

static int num_process = 0;
static int num_throws = 0;
static double radius = 1.0;

typedef unsigned long res_t;
```

```
typedef struct {
    pid_t pid;
    int res;
} RES;

typedef struct {
    char radius[16];
    char processes[16];
    char throws[16];
} PAR;

void analyze_options(int argc, char *argv[]) {
    int c;
    opterr = 0;
    while ((c = getopt(argc, argv, "r:p:t:")) != -1) {
        switch (c) {
            case 'r':
                radius = atof(optarg);
                break;
            case 'p':
                num_process = atoi(optarg);
                break;
            case 't':
                num_throws = atoi(optarg);
                break;
            case '?':
                if ((optopt == 'r') || (optopt == 'p') || (optopt == 't')) {
                    fprintf(stderr, "Warning: option '-%c' requires an argument\n", optopt);
                } else {
                    fprintf(stderr, "Warning: unknown option '-%c'\n", optopt);
                }
```

```
                break;
            default:
                exit(EXIT_FAILURE);
        }
    }
}


int validate(PAR *par) {
    printf("Flags value:\n");
    printf("\tr = %g,  processes = %d,  throws = %d\n", radius, num_process, num_throws);

    if (radius <= 0) radius = 1.0;
    if (num_throws < 0) num_throws = 0;
    if (num_process < 0) num_process = 0;

    sprintf(par->radius, "%g", radius);
    sprintf(par->throws, "%d", NUM_P_PROC);
    if (num_process > 0) {
        sprintf(par->processes, "%d", num_process);
    } else {
        num_throws = (num_throws > 0) ? num_throws : NUM_P_PROC;
        sprintf(par->processes, "%d", num_throws / NUM_P_PROC +
                                ((num_throws % NUM_P_PROC) ? 1 : 0));
    }


    printf("Score parameters\n");
    printf("Radius: %s\n", par->radius);
    printf("Processes: %s\n", par->processes);
    printf("Throws: %s\n", par->throws);
```

```
    return 0;
}


static void create_proc(RES *arr, int num, const PAR *p) {
    int i;
    pid_t pid;

    for (i = 0; i < num; i++) {
        usleep(1);
        pid = fork();
        if (pid < 0) {
            fprintf(stderr, "%s %d\n", "ERROR: Cannot create process", i);
            exit(EXIT_FAILURE);
        } else if (pid > 0) {
            arr[i].pid = pid;
        } else {
            execl("./child.out", "./child.out", p->radius, p->throws, NULL);
            fprintf(stderr, "%s%ld\n", "ERROR: Cannot execute program in ", (long) getpid());
            exit(EXIT_FAILURE);
        }
    }
}

static void wait_proc(RES *arr, int num) {
    int i, status;
    pid_t pid;

    while ((pid = wait(&status)) != -1) {
        for (i = 0; i < num; i++) {
```

```
            if (arr[i].pid == pid) {
                if (WIFEXITED(status)) {
                    arr[i].res = WEXITSTATUS(status);
                } else {
                    arr[i].res = -1;
                }
            }
        }
    }
}


static double score_proc(RES *arr, int num, const PAR *p) {
    int i;
    res_t count_all, count_in;
    double res;

    count_all = 0;
    count_in = 0;
    for (i = 0; i < num; i++) {
        if (arr[i].res > 0) {
            count_in += arr[i].res;
            count_all += 1;
        }
    }
    count_all *= atoi(p->throws);
    res = 4.0 * count_in / count_all;

    return res;
}
```

```
double score(const PAR *p) {
    RES *arr = NULL;
    int num;
    double res;

    num = atoi(p->processes);
    arr = (RES *) calloc(num, sizeof(RES));
    if (!arr) {
        fprintf(stderr, "%s\n", "ERROR: Cannot allocate memory\n");
        exit(EXIT_FAILURE);
    }

    create_proc(arr, num, p);
    wait_proc(arr, num);
    res = score_proc(arr, num, p);

    free(arr);

    return res;
}

int main(int argc, char *argv[]) {

    PAR params;
    double res;

    analyze_options(argc, argv);
    validate(&params);
    res = score(&params);

    printf("Approximate PI: %g\n", res);
```

```
    return EXIT_SUCCESS;
}
```

Child

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/time.h>

long long mtime() {
    struct timeval t;

    gettimeofday(&t, NULL);
    long long mt = (long long) t.tv_sec * 1000 + t.tv_usec / 1000;
    return mt;
}

double rnd(double left, double right) {
    return left + (right - left) * ((double) rand() / RAND_MAX);
}

int is_in(double r, double x, double y) {
    return x * x + y * y <= r * r;
}

int simulate(double r, int n) {
    int i, num;
    double x, y;

    srand(mtime());
    num = 0;

    for (i = 0; i < n; ++i) {
```

```
            x = rnd(-r, r);
            y = rnd(-r, r);

            if (is_in(r, x, y)) num += 1;
        }

        return num;
    }

    int main(int argc, char const *argv[]) {

        double r;
        int n;

        if (argc != 3) {
            fprintf(stderr, "Error: %s\n", "wrong number of arguments");
        }

        r = atof(argv[1]);
        n = atoi(argv[2]);

        return simulate(r, n);
    }
```

Результат:

Task 1

```
gcc main.c -o main.out
./main.out
PID 16471
parent PID 16463
group PID 16463
real user ID 1000
effective user ID 1000
group ID 1000
effective group ID 1000
```

Task 2

```
gcc main.c -o main.out
./main.out
Main PID 16521
Main ID1 = 16522 ID2 = 16523

Local variable = 1
Address = 0x7ffe2e468664
Parent PID = 16521
Own PID = 16523
Second Main IDs1 = 16524 IDs2 = 16525
nadia@nadia-VirtualBox:~/Documents/OS/labs/lab6/3$
Local variable = 1
Address = 0x7ffe2e468664
Parent PID = 1304
Own PID = 16522
Second Main IDs1 = 16526 IDs2 = 16527

Local variable = 2
Address = 0x7ffe2e468664
Parent PID = 1304
Own PID = 16525

Local variable = 2
Address = 0x7ffe2e468664
Parent PID = 1304
Own PID = 16524

Local variable = 2
```

```
Local variable = 2
Address = 0x7ffe2e468664
Parent PID = 16522
Own PID = 16527
```

Task 3

```
gcc main.c -o main.out
./main.out
п'ятниця, 26 листопада 2021 21:09:37 +0200
Exit code: 0
sh: 1: nosuchcommand: not found
Exit code: 32512
nadia    :0           2021-11-26 18:36 (:0)
Exit code: 11264
```

Task 5

```
gcc main.c -o main.out
./main.out
п'ятниця, 26 листопада 2021 21:12:18 +0200
Exit code: 0
sh: 1: nosuchcommand: not found
Exit code: 32512
nadia    :0           2021-11-26 18:36 (:0)
Exit code: 11264
```

Task 6

```
gcc main.c -o main.out
./main.out
    PID TTY      STAT   TIME COMMAND
      1 ?        Ss     0:01 /sbin/init splash
      2 ?        S      0:00 [kthreadd]
      3 ?        I<     0:00 [rcu_gp]
      4 ?        I<     0:00 [rcu_par_gp]
      6 ?        I<     0:00 [kworker/0:0H-events_highpri]
      9 ?        I<     0:00 [mm_percpu_wq]
     10 ?        S      0:00 [rcu_tasks_rude_]
     11 ?        S      0:00 [rcu_tasks_trace]
     12 ?        S      0:00 [ksoftirqd/0]
     13 ?        I      0:00 [rcu_sched]
     14 ?        S      0:00 [migration/0]
     15 ?        S      0:00 [idle_inject/0]
     16 ?        S      0:00 [cpuhp/0]
     17 ?        S      0:00 [kdevtmpfs]
     18 ?        I<     0:00 [netns]
     19 ?        I<     0:00 [inet_frag_wq]
     20 ?        S      0:00 [kauditd]
     21 ?        S      0:00 [khungtaskd]
     22 ?        S      0:00 [oom_reaper]
     23 ?        I<     0:00 [writeback]
     24 ?        S      0:00 [kcompactd0]
```

Task 7

```
gcc child.c -o child.out
gcc parent.c -o parent.out
./parent.out
Flags value:
        r = 1,  processes = 0,  throws = 0
Score parameters
Radius: 1
Processes: 1
Throws: 250
Approximate PI: 3.168
```