|  |  |  |
| --- | --- | --- |
|  | | |
| Федеральное государственное бюджетное  образовательное учреждение высшего образования «Новосибирский государственный технический университет» | | |
|  | | |
| Кафедра прикладной математики | | |
| Лабораторная работа № 2 | | |
| по дисциплине «Управление ресурсами в вычислительных системах» | | |
| **Порождение нового процесса и работа с ним. Запуск программы в рамках порожденного процесса.** | | |
|  | | |
|  | Бригада 7 | Грушев андрей |
| Группа ПМ-05 | болдырев сергей |
| Вариант 7 | хабарова анастасия |
|  |  |
|  |  |
|  |  |
| Преподаватели | стасышин владимир михайлович |
|  | сивак мария алексеевна |
| Новосибирск, 2023 | | |

# Условие

Разработать программу, реализующую действия, указанные в задании к лабораторной работе с учетом следующих требований:  
1) все действия, относящиеся как к родительскому процессу, так и к порожденным процессам, выполняются в рамках одного исполняемого файла;  
2) обмен данными между процессом-отцом и процессом-потомком предлагается выполнить посредством временного файла: процесс-отец после порождения процесса-потомка постоянно опрашивает временный файл, ожидая появления в нем информации от процесса-потомка;  
3) если процессов-потомков несколько, и все они подготавливают некоторую информацию для процесса-родителя, каждый из процессов помещает в файл некоторую структурированную запись, при этом в этой структурированной записи содержатся сведения о том, какой процесс посылает запись, и сама подготовленная информация.

4) Разработать программу, вычисляющую число размещений A(m,n)=n!/(n-m!). Для вычисления факториалов n!, (n-m)! должны быть порождены два параллельных процесса-потомка.

# Используемые программные средства

**strtol(const char \*nptr, char \*\*endptr, int base)** – функция конвертирует начальную часть строки nptr в длинное целое в соответствии с указанным base. Возвращает результат преобразования, если значение не вызвало «переполнения» или не было крайне маленьким.

**fork(void)** – системный вызов, порождающий новый дочерний процесс.

**wait(int \*status)** – системный вызов, с помощью которого выполняется ожидание завершения процесса-потомка родительским процессом.

**exit(int status)** – системный вызов, предназначенный для завершения функционирования процесса. Аргумент status является статусом завершения, который передается родительскому процессу, если он выполнял системный вызов wait.

**execl(const char \*path, const char \*arg, ...)** – функция, заменяющая текущий образ процесса новым образом процесса. Функция дублирует действия оболочки, относящиеся к поиску исполняемого файла.

**fprintf(FILE\* stream, const char \*format, …)** – форматированный вывод в файл на который указывает stream.

**open(const char \*pathname, int flags)** – преобразовывает путь к файлу в описатель файла. Возвращает файловый описатель, который не открыт процессом. Функция с флагом O\_TRUNC урезает длину файла до нуля, если файл существует, является обычным файлом и режим позволяет запись в этот файл.

**write(int fd, const void \*buf, size\_t count****)** – производит запись в описатель файла. Возвращает количество записанных байтов в случае успешного завершения, иначе возвращает -1.

**close(int fd)** – закрывает файловый дескриптор, все блокировки, находящиеся на соответствующем файле, снимаются.

**void exit(int status)** – функция приводит к обычному завершению программы. Стандарт C описывает два определения EXIT\_SUCCESS и EXIT\_FAILURE, которые могут быть переданы функции для обозначения соответственно успешного и неуспешного завершения.

# Алгоритм решения

1. Родительский процесс создает дочерний процесс для расчета (n-m)! и переходит в режим ожидания завершения дочернего процесса.
2. Процесс для расчета (n-m)! создает дочерний процесс для расчета n! и переходит в режим ожидания завершения дочернего процесса.
3. Дочерний процесс для расчета n! рассчитывает n!, записывает результат вычисления в файл data и завершается.
4. Дочерний процесс для расчета (n-m)! рассчитывает (n-m)!, записывает результат вычисления в файл data и завершается.
5. Родительский процесс считывает данные из файла data и выводит результат.

# Спецификация

Код программы расположен на сервере НГТУ в директории **/home/NSTU/pmi-b0507/upres/lab2**. Файлы с кодом на языке **C** – **main.c**, **factorial.c**.

Для корректной работы программы необходимо, чтобы в одной директории с основным исполняемым файлом находился исполняемый файл с именем **factorial.o**.

Получить данный исполняемый файл можно следующей командой:

**gcc factorial.c -o factorial.o**

Для получения основного исполняемого файла необходимо находясь в директории с файлом **main.c** выполнить команду:

**gcc main.с -o [имя\_исполняемого\_файла]**,

либо воспользоваться make-файлом при помощи команды:  
**make all**,

которая создаст исполняемые файлы **main.o и factorial.o**.

Запуск программы происходит при помощи команды:

**./[имя\_исполняемого\_файла] [значение\_n] [значение\_m]**,

например:

**./main.o 5 4**

Значения аргументов **n** и **m** должны быть натуральными числами.

Формат вывода результата:

Number of permutations without repetitions: **[результат]**.

# Перечень тестов

|  |  |  |  |
| --- | --- | --- | --- |
| № | Входные данные | Назначение | Результаты работы программы |
| 1 | ./main.o 5 | Запуск программы с недостаточным количеством параметров. | Error! Wrong number of arguments (expected 2, given 1). |
| 2 | ./main.o 5 4 8 | Запуск программы с излишним количеством параметров. | Error! Wrong number of arguments (expected 2, given 3). |
| 3 | ./main.o 5 a | Второй параметр не является целым числом. | Error! ProcessNM cannot convert second argument (a) to integer. |
| 4 | ./main.o 3.48 5 | Первый параметр не является целым числом. | Error! ProcessN cannot convert first argument (3.48) to integer. |
| 5 | ./main.o 5 4 | Запуск программы с правильным количеством параметров. | Number of permutations without repetitions: 120. |
| 6 | ./main.o 20 8 | Аргументы вызывают целочисленное переполнение. | Number of permutations without repetitions: -4. |
| 7 | ./main.o 5 4 | Отсутствие исполняемого файла factorial.o в директории с основным исполняемым файлом. | Process N unable to exec factorial.o (No such file or directory). |

# Make-файлы

Файл **makefile:**

main: main.c

gcc main.c -o main.o

factorial: factorial.c

gcc factorial.c -o factorial.o

all: main.c factorial.c

gcc main.c -o main.o

gcc factorial.c -o factorial.o

clean:

rm main.o

rm factorial.o

# Листинг программы

main.c:

#include <stdio.h>

#include <errno.h>

#include <string.h>

#include <unistd.h>

#include <stdlib.h>

#include <fcntl.h>

#include <sys/types.h>

#include <sys/stat.h>

#include <sys/wait.h>

int main**(**int argc**,** char**\*\*** argv**)**

**{**

**if** **(**argc **!=** 3**)**

**{**

fprintf**(**stderr**,** "Error! Wrong number of arguments (expected 2, given %d).\n"**,** **--**argc**);**

exit**(**EXIT\_FAILURE**);**

**}**

pid\_t pidNM **=** fork**();** //Процесс, который будет считать (n - m)!

**if** **(**pidNM **==** **-**1**)** //Если не удалось создать процесс.

fprintf**(**stderr**,** "Main process unable to fork.\n"**);**

**else** **if** **(**pidNM **>** 0**)** //Родительский процесс.

**{**

int status**;**

wait**(&**status**);** //Ожидает завершения процесса NM.

**if** **(**status **!=** EXIT\_SUCCESS**)** //Если процесс NM завершился с ошибкой.

exit**(**EXIT\_FAILURE**);**

int fd**,** nFactorial**,** nmFactorial**;**

fd **=** open**(**"data"**,** O\_RDONLY**);** //Открываем файл с данными для чтения.

**if** **(**read**(**fd**,** **&**nFactorial**,** **sizeof(**int**))** **!=** **sizeof(**int**)** **||**

read**(**fd**,** **&**nmFactorial**,** **sizeof(**int**))** **!=** **sizeof(**int**))** //Проверяем, чтобы считалось нужное количество байт (два раза по int).

**{**

close**(**fd**);** //Закрываем файл.

exit**(**EXIT\_FAILURE**);**

**}**

close**(**fd**);** //Закрываем файл.

printf**(**"Number of permutations without repetitions: %d.\n"**,** **(**nFactorial **/** nmFactorial**));** //Вывод результата

**}**

**else** //Процесс NM

**{**

pid\_t pidN **=** fork**();** //Процесс, который будет считать n!

**if** **(**pidN **==** **-**1**)** //Если не удалось создать процесс.

fprintf**(**stderr**,** "Process NM unable to fork.\n"**);**

**else** **if** **(**pidN **>** 0**)** //Родительский процесс.

**{**

int status**;**

wait**(&**status**);** //Ожидает завершения процесса N.

**if** **(**status **!=** EXIT\_SUCCESS**)** //Если процесс N завершился с ошибкой.

exit**(**EXIT\_FAILURE**);**

**if** **(**execl**(**"factorial.o"**,** argv**[**1**],** argv**[**2**],** **NULL)** **==** **-**1**)** //Запуск программы для расчёта (n - m)!

**{**

fprintf**(**stderr**,** "Process NM unable to exec factorial.o (%s).\n"**,** strerror**(**errno**));**

exit**(**EXIT\_FAILURE**);**

**}**

exit**(**EXIT\_SUCCESS**);**

**}**

**else** //Процесс N.

**{**

**if** **(**execl**(**"factorial.o"**,** argv**[**1**],** **NULL)** **==** **-**1**)** //Запуск программы для расчёта n!

**{**

fprintf**(**stderr**,** "Process N unable to exec factorial.o (%s).\n"**,** strerror**(**errno**));**

exit**(**EXIT\_FAILURE**);**

**}**

exit**(**EXIT\_SUCCESS**);**

**}**

**}**

**return** 0**;**

**}**

factorial.c:

#include <stdio.h>

#include <errno.h>

#include <string.h>

#include <unistd.h>

#include <stdlib.h>

#include <sys/types.h>

#include <sys/stat.h>

#include <sys/wait.h>

#include <fcntl.h>

/\*\*

\* Функция считает факториал целого числа.

\* **@param** n Число, факториал которого нужно рассчитать.

\* **@return** Значение факториала числа.

\*\*/

int factorial**(**int n**)**

**{**

int i**,** result **=** 1**;**

**for** **(**i **=** 2**;** i **<=** n**;** **++**i**)**

result **\*=** i**;**

**return** result**;**

**}**

int main**(**int argc**,** char**\*\*** argv**)**

**{**

**if** **(**argc **==** 1**)** //Для расчёта n!

**{**

int n**,** fd**,** result**;**

char **\***endptr**;**

n **=** strtol**(**argv**[**0**],** **&**endptr**,** 10**);** //Конвертирование строки с аргументом к типу integer.

**if** **(\***endptr **!=** '\0'**)**

**{**

fprintf**(**stderr**,** "Error! ProcessN cannot convert first argument (%s) to integer.\n"**,** argv**[**0**]);**

**return(**EXIT\_FAILURE**);**

**}**

result **=** factorial**(**n**);** //Расчёт n!

fd **=** open**(**"data"**,** O\_TRUNC **|** O\_WRONLY**);** //Открываем файл для записи с изменением длины файла до 0

write**(**fd**,** **&**result**,** **sizeof(**int**));** //Записываем в файл значение n!

close**(**fd**);** //Закрываем файл.

**}**

**else** **if** **(**argc **==** 2**)** //Для расчёта (n - m)!

**{**

int n**,** m**,** fd**,** result**;**

char **\***endptr**;**

n **=** strtol**(**argv**[**0**],** **&**endptr**,** 10**);** //Конвертирование строки с первым аргументом к типу integer.

**if** **(\***endptr **!=** '\0'**)**

**{**

fprintf**(**stderr**,** "Error! ProcessNM cannot convert first argument (%s) to integer.\n"**,** argv**[**0**]);**

exit**(**EXIT\_FAILURE**);**

**}**

m **=** strtol**(**argv**[**1**],** **&**endptr**,** 10**);** //Конвертирование строки со вторым аргументом к типу integer.

**if** **(\***endptr **!=** '\0'**)**

**{**

fprintf**(**stderr**,** "Error! ProcessNM cannot convert second argument (%s) to integer.\n"**,** argv**[**1**]);**

exit**(**EXIT\_FAILURE**);**

**}**

result **=** factorial**(**n **-** m**);** //Расчёт (n - m)!

fd **=** open**(**"data"**,** O\_WRONLY **|** O\_APPEND**);** //Открываем файл для записи в режиме добавления.

write**(**fd**,** **&**result**,** **sizeof(**int**));** //Записываем в файл значение (n - m)!

close**(**fd**);** //Закрываем файл.

**}**

**return** 0**;**

**}**