Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное учреждение высшего образования

«Новгородский государственный университет имени Ярослава Мудрого»

Институт «Электронных и информационных систем»

Кафедра «Информационных систем и технологий»

**ОЧЕРЕДЬ СООБЩЕНИЙ В UNIX**

Лабораторная работа №7

преподаватель:

\_\_\_\_\_\_\_\_\_ Ананьев В. В.

«\_\_\_» \_\_\_\_\_\_\_\_\_\_ .

студент группы 9091:

\_\_\_\_\_\_\_ Чалый С. М.

«\_\_\_» \_\_\_\_\_\_\_\_\_\_ .

Великий Новгород

2021

## Задание на лабораторную работу

В данной работе требуется использовать параллельные потоки (threads), а не процессы.

Основной поток должен сгенерировать и вывести на экран 4 случайных числа.

Далее эти числа должны быть переданы во второй процесс через очередь сообщений.

Второй поток должен сгенерировать все возможные перестановки из этих чисел без повторений («1 2 3 4», «2 1 3 4», «4 3 1 2» и т.д.) и передать их обратно.

Первый поток должен получить все эти перестановки и вывести их на экран. В конце требуется вывести количество полученных перестановок.

Первый поток должен самостоятельно выполнить освобождение всех выделенных ресурсов в конце своей работы.

В отчете привести исходный код программы, а также результаты, выведенные на экран.

Пример программы:

#include <stdio.h>#include <sys/types.h>#include <sys/ipc.h>#include <sys/msg.h>struct Strmsg{ long mtype; char str[256]; int islast;};void parentMainCode(int msgId){ struct Strmsg localmsg; int i; printf("Parent: 15 messages will be send.\n"); for (i = 1; i <= 15; ++i) { //prepare message localmsg.mtype = 7; sprintf(localmsg.str, "this is message number %i", i); localmsg.islast = !(i < 15); //send message msgsnd(msgId, &localmsg, sizeof(localmsg), 0); } printf("Parent: wait until child is finished.\n"); waitpid(0, 0, 0); printf("Parent: releasing the message queue.\n"); msgctl(msgId, IPC\_RMID, NULL); printf("Parent: Process is finished.\n");}void childMainCode(int msgId){ struct Strmsg childlocalmsg; int islast = 0; do { //read next message ssize\_t len = msgrcv(msgId, &childlocalmsg, sizeof(childlocalmsg), 0, 0); printf("Child: message with len = %i incoming.\n", len); islast = childlocalmsg.islast; printf("Child: the following string was received: %s\n", childlocalmsg.str); } while(!islast); printf("Child: the last message was read.\n"); printf("Child: Process is finished.\n");}int main(){ const size\_t semCount = 10; int msgId = msgget(IPC\_PRIVATE, 0600|IPC\_CREAT); if (msgId < 0) { perror("error with msgget()"); return -1; } else { printf("message id = %i\n", msgId); } pid\_t childId = fork(); if (childId < 0) { perror("error with fork()\n"); } else if (childId > 0) { parentMainCode(msgId); } else { childMainCode(msgId); } return 0;}

#include <sys/types.h>

#include <sys/ipc.h>

#include <sys/msg.h>

#include <pthread.h>

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

struct Strmsg

{

long mtype;

int digits[4];

};

struct args

{

int msgId;

};

int sort(const void \*x, const void \*y)

{

return \*(int \*)x - \*(int \*)y;

}

void swap(int \*digits, int x, int y)

{

int tmp = digits[x];

digits[x] = digits[y];

digits[y] = tmp;

}

int gen(int \*digits, size\_t n)

{

int j = n - 2;

while (j != -1 && digits[j] >= digits[j + 1])

{

j--;

}

if (j == -1)

{

return 0;

}

int k = n - 1;

while (digits[j] >= digits[k])

{

k--;

}

swap(digits, j, k);

int l = j + 1, r = n - 1;

while (l < r)

{

swap(digits, l++, r--);

}

return 1;

}

void \*mainThd(void \*thd\_args)

{

srand(time(NULL));

int digits[4];

for (size\_t i = 0; i < 4; ++i)

{

digits[i] = rand() % 100;

printf("%d ", digits[i]);

}

printf("\n");

struct args \*args\_var = (struct args \*)thd\_args;

int msgId = args\_var->msgId;

struct Strmsg message;

memcpy(message.digits, digits, 4 \* sizeof(int));

message.mtype = 1;

msgsnd(msgId, &message, sizeof(message), 0);

int count = 0;

do

{

size\_t size = msgrcv(msgId, &message, sizeof(message), 101, 0);

if (message.digits[0] != -1)

{

for (size\_t i = 0; i < 4; ++i)

{

printf("%d ", message.digits[i]);

}

count ++;

printf("\n");

}

} while (message.digits[0] != -1);

printf("count: %d\n", count);

msgctl(msgId, IPC\_RMID, NULL);

return NULL;

}

void \*childThd(void \*thd\_args)

{

struct args \*args = (struct args \*)thd\_args;

int msgId = args->msgId;

struct Strmsg message;

msgrcv(msgId, &message, sizeof(message), 1, 0);

qsort(message.digits, 4, sizeof(int), sort);

message.mtype = 101;

msgsnd(msgId, &message, sizeof(message), 0);

while (gen(message.digits, 4))

{

message.mtype = 101;

msgsnd(msgId, &message, sizeof(message), 0);

}

message.digits[0] = -1;

message.mtype = 101;

msgsnd(msgId, &message, sizeof(message), 0);

return NULL;

}

int main()

{

int msgId = msgget(IPC\_PRIVATE, 0600 | IPC\_CREAT);

if (msgId < 0) {

perror("error with msgget()");

return -1; }

else {

printf("message id = %i\n", msgId);

}

struct args \*args\_var;

args\_var->msgId = msgId;

pthread\_t mainThread, childThread;

pthread\_create(&mainThread, NULL, mainThd, (void \*)args\_var);

pthread\_create(&childThread, NULL, childThd, (void \*)args\_var);

pthread\_join(mainThread, NULL);

}