Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Лабораторная работа №6-8 по курсу**

**«Операционные системы»**

**Управление потоками в ОС**

Студент: Гаврилов Максим Сергеевич

Группа: М8О–206Б–20

Вариант: 8

Преподаватель: Соколов Андрей Алексеевич

Оценка: \_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_

Москва, 2020.

**Постановка задачи**

## Цель работы

## Целью является приобретение практических навыков в:

## • Управлении серверами сообщений (№6)

## • Применение отложенных вычислений (№7)

## • Интеграция программных систем друг с другом (№8)

## Задание

Реализовать распределенную систему по асинхронной обработке запросов. В данной распределенной системе должно существовать 2 вида узлов: «управляющий» и «вычислительный». Необходимо объединить данные узлы в соответствии с той топологией, которая определена вариантом. Связь между узлами необходимо осуществить при помощи технологии очередей сообщений. Также в данной системе необходимо предусмотреть проверку доступности узлов в соответствии с вариантом. При убийстве («kill -9») любого вычислительного узла система должна пытаться максимально сохранять свою работоспособность, а именно все дочерние узлы убитого узла могут стать недоступными, но родительские узлы должны сохранить свою работоспособность. Управляющий узел отвечает за ввод команд от пользователя и отправку этих команд на вычислительные узлы. Список основных поддерживаемых команд:

* Создание нового вычислительного узла
* Удаление существующего вычислительного узла
* Исполнение команды на вычислительном узле

**Условия задачи**

Топология – Все вычислительные узлы находятся в списке. Есть только один управляющий узел. Чтобы добавить новый вычислительный узел к управляющему, то необходимо выполнить команду: create id -1.

Набор команд – Формат команды: exec id n k1 … kn id – целочисленный идентификатор вычислительного узла, на который отправляется команда n – количество складываемых чисел (от 1 до 108) k1 … kn – складываемые числа

Команды проверки – Формат команды: ping id Команда проверяет доступность конкретного узла. Если узла нет, то необходимо выводить ошибку: «Error: Not found»

**Общие сведения о программе**

Программа компилируется из файлов [list.c](https://github.com/Megict/OS_lab_six/blob/main/src/list.c), [list.h](https://github.com/Megict/OS_lab_six/blob/main/src/list.h), [main.c](https://github.com/Megict/OS_lab_six/blob/main/src/main.c), [msgOPR.c](https://github.com/Megict/OS_lab_six/blob/main/src/msgOPR.c), [msgOPR.h](https://github.com/Megict/OS_lab_six/blob/main/src/msgOPR.h), [mutex.c](https://github.com/Megict/OS_lab_six/blob/main/src/mutex.c), [mutex.h](https://github.com/Megict/OS_lab_six/blob/main/src/mutex.h), [operator.c](https://github.com/Megict/OS_lab_six/blob/main/src/operator.c), [oprUI.h](https://github.com/Megict/OS_lab_six/blob/main/src/oprUI.h), [transformres.h](https://github.com/Megict/OS_lab_six/blob/main/src/transformres.h), [worker.c](https://github.com/Megict/OS_lab_six/blob/main/src/worker.c). Для взаимодействия управляющего узла с рабочими используется оператор сообщений, функционирующий в отдельном процессе. Для предачи сообщений он использует каналы (pipe). Оператор имеет лишь два канала – входной и выходной. На первый он принимает команды, на второй – отправляет ответ. Из-за такой архитектуры для корректного приема сообщений процессами необходимо использовать мьютекс (примитивный мьютекс на отображаемой памяти есть в файле mutex.c) Команда оператора включает в себя индекс действия, которое должен выполнить оператор и сообщение, которое он должен переслать. Действие может заключаться в помещении сообщения в очередь, либо же извлечение сообщения, адресованного процессу, pid которого указан во входном сообщении, из очереди и пересылке его на выходной канал.

Рабочие же реализованы как настоящие таймеры. Каждый из них крутится в бесконечном цикле, замирая на миллисекунду на каждом шаге. Если был получен сигнал от управляющего узла, то рабочий считывает сообщение, полученное им через оператор, определяет, какую команду нужно выполнить и либо останавливается, либо стартует, либо отправляет управляющему узлу данные о своем локальном времени.

В программе используются следующие системные вызовы:

1. **read –** пытается записать *n* байт из файлового дескриптора *fd* в буфер *buf*.
2. **Write –** пытается записать *n* байт из буфера *buf в ф*айловый дескриптор *fd*.
3. **Mmap –** производит отображение файла в память. После этого к нему можно обращаться по адресу в *heap*
4. **Open –** преобразует путь к файлу в файловый дескриптор.
5. **Close –** закрывает файловый дескриптор, который после этого не ссылается ни на один и файл и может быть использован повторно.
6. **Pipe –** создает пару файловых дескрипторов, указывающих на запись именованного канала, и помещает их в массив, на который указывает *filedes*. *filedes[0]* предназначен для чтения, а *filedes[1]* предназначен для записи
7. **Fork –** порождает новый процесс (процесс-потомок), который почти идентичен порождающему процессу-родителю.

**Общий метод и алгоритм решения**.

* + - 1. Создадим простой список для хранения данных о вычислительных узлах. Поместим его в list.c. Дополнительно сделаем точно такую же структуру, элементами которой будут уже сообщения. Она пригодится про организации оператора.
      2. Делаем основу программы управляющего узла – простой парсер команд и функции запуска вычислительных узлов.
      3. Для реализации оператора выполним небольшую надстройку над списком сообщений из п.1 – делаем структуру массива списков, а также добавляем функцию изъятия последнего элемента из списка. Добавив к каждой очереди поле с pid-ом получателя и получим массив очередей сообщений.
      4. Делаем функции помещения сообщения в массив очередей и извлечения из массива последнего сообщения для данного процесса. Добавляем простой алгоритм, читающий сообщения с входного пайпа и вызывающий соответствующую функцию. Оператор готов.
      5. Для ребенка нам понадобится установить обработчик сигналов, инкрементирующий переменную, которая показывает, что родитель отправил нам сообщение. Также реализуем функцию, паузящую процесс на 1 миллисекунду (почему на юниксе sleep(n) паузит процесс на n СЕКУНД, а не миллисекунд, как в винде – так еще загадка, конечно)
      6. Добавляем в программу управляющего узла все функци ддля работы с оператором и вычислительным узлом. Чтобы насколько процессов не читали выходной канал оператора одновременно добавим мьютекс.
      7. Для «переклички» процессов будем использовать обыкновенную переброску сигналов

**Основные файлы программы**

**Main.c** (управляющий узел)

|  |
| --- |
| //Гаврилов М.С. М8О-206Б-19 | Лр#6,7,8, вар. 21  //топология 1, команды 3,проверка 1.  // Все вычислительные узлы находятся в списке. Есть только один управляющий узел. Чтобы  // добавить новый вычислительный узел к управляющему, то необходимо выполнить команду:  // create id -1.  // Набора команд 3 (локальный таймер)  // Формат команды сохранения значения: exec id subcommand  // subcommand – одна из трех команд: start, stop, time.  // start – запустить таймерstop – остановить таймер  // time – показать время локального таймера в милисекундах  // Тип проверки доступности узлов  // Команда проверки 2  // Формат команды: ping  #include <string.h>  #include <unistd.h>  #include <stdlib.h>  #include <fcntl.h>  #include <math.h>  #include <signal.h>  #include <sys/types.h>  #include <sys/mman.h>  #include "list.h"  #include "transformres.h"  #include "mutex.h"  #include <time.h>  #include <stdio.h>  #include "oprUI.h"  #define Pi 3.1415926  #define MAP\_FILE\_SIZE 100  #define DEBUG 0  int readTag **=** 0**;**  int answTag **=** 0**;**  int killTag **=** 0**;**  void taghdl**(**int sig**)**  **{**  **if(**sig **==** SIGUSR1**)**  **{**  **if(**DEBUG **==** 1**)**printf**(**"MAINSIG(%d)\n"**,**readTag**);**  readTag**++;**  **}**  **else**  **if(**sig **==** SIGUSR2**)**  **{**  answTag **=** 1**;**  **}**  **if(**sig **==** SIGINT**)**  **{**  killTag **=**1**;**  **}**  **}**  char**\*** bitescan **(**char**\*** str**)** //принимает неинициализированный указатель, маллочит для него память и считывает строку  **{**  int counter **=** 0**,**size **=** 2**;**  str **=** **(**char**\*)**realloc**(**str**,**size**);**  char buf**;**  int mark**=**0**;**  **while(**read**(**0**,&**buf**,**1**)!=**0**)** **{**  **if(**buf **==** ' '**||**buf **==** '\n'**||**buf **==**'\t'**)** **{**  **if(**mark **==** 0**)** **{**  **continue;**  **}**  **else** **{**  **break;**  **}**  **}**  mark**=**1**;**  **if(**counter**+**1 **==** size**)** **{**  size**\*=**2**;**  str **=** **(**char**\*)**realloc**(**str**,**size**);**  **}**  str**[**counter**]** **=** buf**;**  **++**counter**;**  **}**  **if(**counter **==** 0**)** **{** //если ничего не считали  free**(**str**);**  str **=** **NULL;**  **return** str**;**  **}**  //str = (char\*)realloc(str,counter); //если раскомментить, то почему-то ругается валгринд  str**[**counter**]** **=** 0**;**  **return** str**;**  **}**  proc**\*** CreateWorker**(**int id**,**int fd\_to\_opr**,** int fd\_from\_opr**)** **{**  int curID **=** getpid**();**  proc**\*** crb**;**  crb **=** **(**proc**\*)**malloc**(sizeof(**proc**));**  crb**->**lid **=** id**;**  crb**->**pid **=** fork**();**    **if(**crb**->**pid **==** **-**1**)** **{**  printf**(**"unable to separate\n"**);**  **return** **NULL;**  **}**  **else**  **if(**crb**->**pid **==** 0**)** **{**  //printf("initing\n");  execl**(**"worker.exe"**,**"worker.exe"**,**transform**(**fd\_to\_opr**),**transform**(**fd\_from\_opr**),**transform**(**id**),**transform**(**curID**),NULL);**  **return** **NULL;**  **}**  **else** **{**  **return** crb**;**  **}**    **}**  int CreatePostman**(**int**\*** fd\_to\_opr**,**int**\*** fd\_from\_opr**)** **{**  int fd1**[**2**];**  int fd2**[**2**];**  pipe**(**fd1**);**  pipe**(**fd2**);**  **\***fd\_to\_opr **=** fd1**[**1**];**  **\***fd\_from\_opr **=** fd2**[**0**];**  int postmanID **=** fork**();**  **if(**postmanID **==** **-**1**)** **{**  **return** **-**1**;**  **}**  **else**  **if(**postmanID **==** 0**)** **{**  execl**(**"operator.exe"**,**transform**(**fd1**[**0**]),**transform**(**fd2**[**1**]),NULL);**  **return** 0**;**  **}**  **else** **{**  **return** postmanID**;** exit**(**0**);**  **}**  **}**  void destroy**(**proc**\*** dbl**)** **{**  kill**(**dbl**->**pid**,**SIGTERM**);**  free**(**dbl**);**  **}**  int pingWorker**(**proc**\*** pbl**)** **{**  kill**(**pbl**->**pid**,**SIGUSR2**);**  sleep**(**2**);**  **if(**answTag **==** 1**)** **{**  answTag **=** 0**;**  **return** 0**;**  **}**  **else** **{**  **return** **-**1**;**  **}**  **}**  void freze\_for\_1\_ms**()** **{**  struct timespec time1**,**time2**;**  clock\_gettime**(**CLOCK\_REALTIME**,&**time1**);**  clock\_gettime**(**CLOCK\_REALTIME**,&**time2**);**  **while(**/\*(time1.tv\_nsec/1000000 + 1)%1000 > time2.tv\_nsec/1000000 || \*/**(**time2**.**tv\_nsec**/**1000000 **==** time1**.**tv\_nsec**/**1000000**))** **{**  clock\_gettime**(**CLOCK\_REALTIME**,&**time2**);**  **}**  **}**  void freze**(**int t**)** **{**  **for(**int i**=**0**;**i**<**t**;++**i**)** **{**  freze\_for\_1\_ms**();**  **}**  **}**  int main**()** **{**  //задаем стурктуру для sigaction  struct sigaction act**;**  memset**(&**act**,** 0**,** **sizeof(**act**));**  act**.**sa\_handler **=** taghdl**;**  sigset\_t set**;**  sigemptyset**(&**set**);**  sigaddset**(&**set**,** SIGUSR1**);**  sigaddset**(&**set**,** SIGUSR2**);**  sigaddset**(&**set**,** SIGINT**);**  act**.**sa\_mask **=** set**;**  sigaction**(**SIGUSR1**,** **&**act**,** 0**);**  sigaction**(**SIGUSR2**,** **&**act**,** 0**);**  sigaction**(**SIGINT**,** **&**act**,** 0**);**  //обработчики сигналов готовы  **if(**createMutex**()!=**0**)** **{**  printf**(**"ERROR: create mutex error\n"**);**  **return** **-**1**;**  **}**  mutex mtx**;**  **if(**linkToMutex**(&**mtx**)!=**0**)** **{**  printf**(**"ERROR: mutex error\n"**);**  **return** **-**1**;**  **}**  List IndList**;**  LIinit **(&**IndList**);**    char**\*** inpstr **=** **(**char**\*)**malloc**(**20**);**  char**\*** subcommand **=** **(**char**\*)**malloc**(**10**);**  int fd\_to\_opr**,** fd\_from\_opr**;**  int work **=** CreatePostman**(&**fd\_to\_opr**,&**fd\_from\_opr**);** //запускаем оператор сообщений  **while(**0**==**0**)** **{**  strcpy**(**inpstr**,**""**);**  printf**(**"> "**);**  **if(**scanf**(**"%s"**,**inpstr**)==**EOF**)** **{**  **break;**  **}**//возвращает EOF при получении сигнала  **if(**strcmp**(**inpstr**,**"create"**)** **==** 0**)** **{**  int id**;**  scanf**(**"%d"**,&**id**);**    **if(**find**(&**IndList**,**id**)** **!=** **-**1**)** **{**  printf**(**"ERROR: already exists\n"**);**  **continue;**  **}**  proc**\*** ADB **=** CreateWorker**(**id**,**fd\_to\_opr**,**fd\_from\_opr**);**  add**(&**IndList**,**ADB**);**  printf**(**"OK %d\n"**,**id**);**  **}**  **else**  **if(**strcmp**(**inpstr**,**"remove"**)** **==** 0**)** **{**  int id**;**  scanf**(**"%d"**,&**id**);**  int ind **=** find**(&**IndList**,**id**);**  **if(**ind **==** **-**1**)** **{**  printf**(**"ERROR: no such node\n"**);**  **continue;**  **}**  proc**\*** dbl **=** getPointer**(&**IndList**,**id**);**  destroy**(**dbl**);**  Lremove**(&**IndList**,**ind**);**    printf**(**"OK %d\n"**,**id**);**  **}**  **else**  **if(**strcmp**(**inpstr**,**"exec"**)** **==** 0**)** **{**  int id**;**  scanf**(**"%d %s"**,&**id**,**subcommand**);**  proc**\*** curWc **=** getPointer**(&**IndList**,**id**);**  **if(**curWc **==** **NULL)** **{**  printf**(**"ERROR: node with index %d does not exist\n"**,**id**);**  **}**  **else**  **if(**pingWorker**(**curWc**)!=**0**)** **{**  printf**(**"ERROR: node with index %d is unavailable\n"**,**id**);**  **}**  **else** **{**  **if(**strcmp**(**subcommand**,**"start"**)** **==** 0**)** **{**  sendMSG**(**fd\_to\_opr**,**1**,**curWc**->**pid**);**  **if(**DEBUG **==** 1**)**printf**(**"ask sent\n"**);**  kill**(**curWc**->**pid**,**SIGUSR1**);**  printf**(**"OK\n"**);**  **}**  **else**  **if(**strcmp**(**subcommand**,**"stop"**)** **==** 0**)** **{**    sendMSG**(**fd\_to\_opr**,**2**,**curWc**->**pid**);**  **if(**DEBUG **==** 1**)**printf**(**"ask sent\n"**);**  kill**(**curWc**->**pid**,**SIGUSR1**);**  printf**(**"OK\n"**);**  **}**  **else**  **if(**strcmp**(**subcommand**,**"time"**)** **==** 0**)** **{**    sendMSG**(**fd\_to\_opr**,**3**,**curWc**->**pid**);**  **if(**DEBUG **==** 1**)**printf**(**"ask sent\n"**);**  kill**(**curWc**->**pid**,**SIGUSR1**);**  **}**  **else** **{**  printf**(**"ERROR: wrong subcommand \"%s\"\n"**,**subcommand**);**  **}**  **}**  **}**  **else**  **if(**strcmp**(**inpstr**,**"print"**)==**0**)** **{**  printf**(**"<|"**);**  **for(**int j**=**0**;**j**<**IndList**.**ListSize**;++**j**)** **{**  printf**(**"%d|"**,**IndList**.**Elem**[**j**]->**lid**);**  **}**  printf**(**"\n"**);**  **}**  **else**  **if(**strcmp**(**inpstr**,**"ping"**)==**0**)** **{**  int id**;**  scanf**(**"%d"**,&**id**);**  proc**\*** wc **=** getPointer**(&**IndList**,**id**);**  **if(**wc **==** **NULL)** **{**  printf**(**"ERROR: node with index %d does not exist\n"**,**id**);**  **continue;**  **}**    **if(**pingWorker**(**wc**)==**0**)** **{**  printf**(**"%d - available\n"**,**id**);**  **}**  **else** **{**  printf**(**"%d - unavailable\n"**,**id**);**  **}**  **}**  **else**  **if(**strcmp**(**inpstr**,**"pingall"**)==**0**)** **{**  printf**(**"nodes:\n"**);**  **for(**int j**=**0**;**j**<**IndList**.**ListSize**;++**j**)** **{**  kill**(**IndList**.**Elem**[**j**]->**pid**,**SIGUSR2**);**  sleep**(**2**);**  **if(**answTag **==** 1**)** **{**  answTag **=** 0**;**  printf**(**"%d - available\n"**,**IndList**.**Elem**[**j**]->**lid**);**  **}**  **else** **{**  printf**(**"%d - unavailable\n"**,**IndList**.**Elem**[**j**]->**lid**);**  **}**  **}**  **}**  **else**  **if(**strcmp**(**inpstr**,**"q"**)** **==** 0 **||** killTag **==** 1**)** **{**  **for(**int j**=**0**;**j**<**IndList**.**ListSize**;++**j**)** **{**  kill**(**IndList**.**Elem**[**j**]->**pid**,**SIGTERM**);**  **}**  kill**(**work**,**SIGTERM**);**  **return** 0**;**  **}**  **else**  **if(**strcmp**(**inpstr**,**"freze"**)** **==** 0**)** **{**  int t**;**  scanf**(**"%d"**,&**t**);**  printf**(**"frost (%d ms)\n"**,**t**);**  freze**(**t**);**  **}**  **else**  **if(**strcmp**(**inpstr**,**""**)!=**0**)** **{**//да, это костыль | ps. хотя даже не совсем и костыль...  printf**(**"ERROR: no such command \"%s\"\n"**,**inpstr**);**  **}**  freze**(**20**);**  //if(readTag > 0) {  **if(**DEBUG **==** 1**)**printf**(**"main hasMSG\n"**);**    **while((**lock**(&**mtx**,**10**)!=**0**))** **{};**  **if(**DEBUG**==**1**)**printf**(**"par--|%c|l\n"**,**mtx**.**memloc**[**0**]);**    unsigned long long time **=** 0**;**  int chPid **=** 0**;**  **if(**getMSG**(**fd\_to\_opr**,**fd\_from\_opr**,&**time**,&**chPid**)** **==** **-**1**)** **{**  **if(**DEBUG**==**1**)**printf**(**"[main] got no msgs\n"**);**  **}**  **else** **{**  int lid **=** findLID**(&**IndList**,**chPid**);**  printf**(**"time of %d is: %llu\n"**,**lid**,**time**);**  **}**  unlock**(&**mtx**);**  **if(**DEBUG**==**1**)**printf**(**"par--|%c|u\n"**,**mtx**.**memloc**[**0**]);**    //readTag--;  //}  **}**  **for(**int j**=**0**;**j**<**IndList**.**ListSize**;++**j**)** **{**  kill**(**IndList**.**Elem**[**j**]->**pid**,**SIGTERM**);**  **}**  kill**(**work**,**SIGTERM**);**  detachFromMutex**(&**mtx**);**  free**(**inpstr**);**  free**(**subcommand**);**  **}** |

**Operator.c** (опратор очереди сообщений)

|  |
| --- |
| #include <string.h>  #include <unistd.h>  #include <stdlib.h>  #include <fcntl.h>  #include <math.h>  #include <signal.h>  #include <sys/types.h>  #include <sys/mman.h>  #include "list.h"  #include "transformres.h"  #include <stdio.h>  #include "msgOPR.h"  #define MAP\_FILE\_SIZE 100  #define DEBUG 0  void taghdl**(**int sig**)**  **{**  **if(**sig **==** SIGTERM**)**  **{**  exit**(**0**);**  **}**  **}**  int main**(**int argc**,**char**\*\*** argv**)** **{**  struct sigaction act\_sync**;**  memset**(&**act\_sync**,** 0**,** **sizeof(**act\_sync**));**  act\_sync**.**sa\_handler **=** taghdl**;**  sigset\_t set\_sync**;**  sigemptyset**(&**set\_sync**);**  sigaddset**(&**set\_sync**,** SIGUSR1**);**  sigaddset**(&**set\_sync**,** SIGTERM**);**  sigaddset**(&**set\_sync**,** SIGUSR2**);**  act\_sync**.**sa\_mask **=** set\_sync**;**  sigaction**(**SIGUSR1**,** **&**act\_sync**,** **NULL);**  sigaction**(**SIGTERM**,** **&**act\_sync**,** **NULL);**  sigaction**(**SIGUSR2**,** **&**act\_sync**,** **NULL);**  int fd\_com\_from **=** back\_transform**(**argv**[**0**]);**//откуда читать  int fd\_com\_to **=** back\_transform**(**argv**[**1**]);**//куда писать    **if(**DEBUG **==** 1**)**printf**(**"operator activatrd\n"**);**    Qmas allmsges**;**  allmsges**.**size **=** 0**;**  allmsges**.**queues **=** **NULL;**    **for(;;)** **{**  ask cur\_ask**;**  cur\_ask**.**action **=** **-**1**;**  cur\_ask**.**mess**.**content **=** 0**;**    read**(**fd\_com\_from**,&**cur\_ask**,sizeof(**cur\_ask**));**  **if(**DEBUG **==** 1**)**printf**(**"operator| got ask\n"**);**  **if(**DEBUG **==** 1**)** **{**  **for(**int i**=**0**;**i**<**allmsges**.**size**;++**i**)** **{**  printf**(**"|%d|-|"**,**allmsges**.**queues**[**i**]->**recipient**);**  **for(**int j**=**0**;**j**<**allmsges**.**queues**[**i**]->**ListSize**;++**j**)** **{**  printf**(**"%llu|"**,**allmsges**.**queues**[**i**]->**Elem**[**j**]->**content**);**  **}**  printf**(**"(%d)\n"**,**allmsges**.**queues**[**i**]->**ListSize**);**  **}**  **}**  **if(**DEBUG **==** 1**)**printf**(**"operator| %d | %d - %d |%llu\n"**,**cur\_ask**.**action**,** cur\_ask**.**mess**.**from**,** cur\_ask**.**mess**.**to**,**cur\_ask**.**mess**.**content**);**  **if(**cur\_ask**.**action **==** 1**)** **{**  **if(**DEBUG **==** 1**)**printf**(**"operator| sending...\n"**);**  msg**\*** mesSD **=** **(**msg**\*)**malloc**(sizeof(**msg**));**  mesSD**->**to **=** cur\_ask**.**mess**.**to**;**  mesSD**->**from **=** cur\_ask**.**mess**.**from**;**  mesSD**->**content **=** cur\_ask**.**mess**.**content**;**  **if(**DEBUG **==** 1**)**printf**(**"operator| msg is: %llu\n"**,**mesSD**->**content**);**  sendTO**(&**allmsges**,**mesSD**->**to**,**mesSD**);**  //kill(cur\_ask.mess.to,SIGUSR1);  **}**  **else**  **if(**cur\_ask**.**action **==** 0**)** **{**  **if(**DEBUG **==** 1**)**printf**(**"operator| cheking...\n"**);**  msg**\*** mesSD **=** reciveBY**(&**allmsges**,**cur\_ask**.**mess**.**to**,**mesSD**);**  **if(**mesSD**!=NULL)** **{**  cur\_ask**.**mess**.**from **=** mesSD**->**from**;**  cur\_ask**.**mess**.**content **=** mesSD**->**content**;**  free**(**mesSD**);**  **if(**DEBUG **==** 1**)**printf**(**"operator| msg is: %llu\n"**,**cur\_ask**.**mess**.**content**);**  write**(**fd\_com\_to**,&**cur\_ask**,sizeof(**cur\_ask**));**  **}**  **else** **{**  **if(**DEBUG **==** 1**)**printf**(**"operator| no messenges\n"**);**  cur\_ask**.**action **=** 2**;** //это значит, что не удалось прочитать  write**(**fd\_com\_to**,&**cur\_ask**,sizeof(**cur\_ask**));**  **}**  **}**    **}**  **}** |

**Worker.c**

|  |
| --- |
| #include <string.h>  #include <unistd.h>  #include <stdlib.h>  #include <fcntl.h>  #include <math.h>  #include <signal.h>  #include <stdio.h>  #include <sys/types.h>  #include <sys/mman.h>  #include "transformres.h"  #include "oprUI.h"  #include "mutex.h"  #include <time.h>  #define DEBUG 0  #define MAP\_FILE\_SIZE 100  char readTag **=** 0**;**  char answTag **=** 0**;** //тег для переклички  void taghdl**(**int sig**)**  **{**  **if(**sig **==** SIGUSR2**)**  **{**  answTag **=** 1**;**  **}**  **else**  **if(**sig **==** SIGUSR1**)**  **{**  **if(**DEBUG **==** 1**)**printf**(**"WS(%d)\n"**,**readTag**);**  readTag**++;**  **}**  **else**  **if(**sig **==** SIGTERM**)**  **{**  exit**(**0**);**  **}**  **}**  void freze\_for\_1\_ms**()** **{**  struct timespec time1**,**time2**;**  clock\_gettime**(**CLOCK\_REALTIME**,&**time1**);**  clock\_gettime**(**CLOCK\_REALTIME**,&**time2**);**  **while(**/\*(time1.tv\_nsec/1000000 + 1)%1000 > time2.tv\_nsec/1000000 || \*/**(**time2**.**tv\_nsec**/**1000000 **==** time1**.**tv\_nsec**/**1000000**))** **{**  clock\_gettime**(**CLOCK\_REALTIME**,&**time2**);**  **}**  **}**  int main**(**int argc**,**char**\*\*** argv**)** **{**  //аргументы - пайпы  //ставим обработчик  struct sigaction act\_sync**;**  memset**(&**act\_sync**,** 0**,** **sizeof(**act\_sync**));**  act\_sync**.**sa\_handler **=** taghdl**;**  sigset\_t set\_sync**;**  sigemptyset**(&**set\_sync**);**  sigaddset**(&**set\_sync**,** SIGUSR1**);**  sigaddset**(&**set\_sync**,** SIGTERM**);**  sigaddset**(&**set\_sync**,** SIGUSR2**);**  act\_sync**.**sa\_mask **=** set\_sync**;**  sigaction**(**SIGUSR1**,** **&**act\_sync**,** **NULL);**  sigaction**(**SIGTERM**,** **&**act\_sync**,** **NULL);**  sigaction**(**SIGUSR2**,** **&**act\_sync**,** **NULL);**  //парисм аргументы  int fd\_to\_operator **=** back\_transform**(**argv**[**1**]);**  int fd\_from\_operator **=** back\_transform**(**argv**[**2**]);**  int parId **=** back\_transform**(**argv**[**4**]);**  int wf **=** 0**;**  unsigned long long time **=** 0**;**  mutex mtx**;**  **if(**linkToMutex**(&**mtx**)!=**0**)** **{**  printf**(**"ERROR: mutex error\n"**);**  exit**(-**1**);**  **}**  **for(;;)** **{**  **if** **(**wf **==** 1**)** **{**  freze\_for\_1\_ms**();**  **++**time**;**  **}**  **if(**readTag **>** 0**)** **{**  **if(**DEBUG **==** 1**)**printf**(**"reading\n"**);**    **while((**lock**(&**mtx**,**10**)** **!=** 0**))** **{}**  **if(**DEBUG **==** 1**)**printf**(**"chil--|%c|l\n"**,**mtx**.**memloc**[**0**]);**  unsigned long long act **=** 0**;**  int from**;**  **if(**getMSG**(**fd\_to\_operator**,**fd\_from\_operator**,&**act**,&**from**)** **==** **-**1**)** **{**  unlock**(&**mtx**);**  **if(**DEBUG **==** 1**)**printf**(**"ERROR: msg lost\n"**);**  readTag **=** 0**;**  **continue;**  **}**  **else** **{**  **if(**from**!=**parId**)** **{**  unlock**(&**mtx**);**  **if(**DEBUG **==** 1**)**printf**(**"ERROR: msg incorrect\n"**);**  readTag **=** 0**;**  **continue;**  **}**  **if(**DEBUG **==** 1**)**printf**(**"%llu\n"**,**act**);**  **}**  unlock**(&**mtx**);**  **if(**DEBUG **==** 1**)**printf**(**"chil--|%c|u\n"**,**mtx**.**memloc**[**0**]);**  **if(**DEBUG **==** 1**)**printf**(**"got post\n"**);**  **if(**act **==** 1**)** **{**  wf **=** 1**;**  **}**  **else**  **if(**act **==** 2**)** **{**  wf **=** 0**;**  **}**  **else**  **if(**act **==** 3**)** **{**  sendMSG**(**fd\_to\_operator**,**time**,**parId**);**  **}**  **else** **{**  printf**(**"ERROR: child got unknown command\n"**);**  **}**  readTag**--;**  **}**  **if(**answTag **==** 1**)** **{**  kill **(**parId**,**SIGUSR2**);** //если процесс спрашивают, жив лион, он отвечает родителю, что жив  answTag **=** 0**;**  **}**  **}**  // munmap(adr,MAP\_FILE\_SIZE);  //close(ftmp);  detachFromMutex**(&**mtx**);**  **return** 0**;**  **}** |

**Пример работы**

**max@max-Swift:~/Рабочий стол/OS/lab6/prog$ make**

*gcc -Wall -std=c99 -D\_POSIX\_C\_SOURCE=199309L -o prog.exe main.c list.c msgOPR.c mutex.c*

*gcc -Wall -std=c99 -D\_POSIX\_C\_SOURCE=199309L -o worker.exe worker.c msgOPR.c mutex.c*

*gcc -Wall -std=c99 -D\_POSIX\_C\_SOURCE=199309L -o operator.exe operator.c msgOPR.c*

Запуск 1. Заморозка управляющего процесса нужна чтобы отмерять точное время. Небольшой убегание часов вперед, – скорее всего, время на пересылку команды.

**max@max-Swift:~/Рабочий стол/OS/lab6/prog$ ./prog.exe < test/test3**

*> OK 1*

*> frost (100 ms)*

*> OK*

*> frost (1000 ms)*

*> OK*

*> time of 1 is: 1042*

*> OK 2*

*> frost (100 ms)*

*> OK*

*> frost (1000 ms)*

*> OK*

*> frost (2000 ms)*

*> OK*

*> time of 2 is: 2042*

*> time of 1 is: 4165*

> frost (100 ms)

**> max@max-Swift:~/Рабочий стол/OS/lab6/prog$ ./prog.exe < test/test1**

> OK 1

> frost (1 ms)

> OK

> frost (1 ms)

> time of 1 is: 43

> OK

> time of 1 is: 64

> time of 1 is: 64

> OK 2

> frost (1 ms)

> OK

> OK

> time of 1 is: 86

> time of 2 is: 63

> time of 1 is: 128

> time of 2 is: 105

> time of 1 is: 170

> time of 2 is: 147

> OK

> OK

> 1 - available

> 2 - available

> OK 1

**> max@max-Swift:~/Рабочий стол/OS/lab6/prog$ ./prog.exe < test/test2**

> OK 1

> frost (1 ms)

> OK

> frost (1 ms)

> OK

> time of 1 is: 43

**> max@max-Swift:~/Рабочий стол/OS/lab6/prog$ ./prog.exe < test/test4**

> OK 1

> frost (10 ms)

> OK

> frost (1000 ms)

> OK

> time of 1 is: 1042

> OK 2

> frost (10 ms)

> OK

> frost (1000 ms)

> OK

> OK 3

> frost (100 ms)

> OK

> frost (10 ms)

> ERROR: node with index 4 does not exist

> time of 3 is: 106

> time of 3 is: 127

> time of 2 is: 370

> frost (2000 ms)

> OK

> time of 2 is: 2382

> time of 1 is: 4262

> frost (100 ms)

Запуск с оператором в режиме отладки. Можно наблюдать процесс пересылки сообщений.

**max@max-Swift:~/Рабочий стол/OS/lab6/prog$ ./prog.exe < test/test3**

> OK 1

operator activatrd

operator| got ask

operator| 0 | -1 - 93252 |1

operator| cheking...

operator| no messenges

> frost (100 ms)

operator| got ask

operator| 0 | -1 - 93252 |1

operator| cheking...

operator| no messenges

> OK

operator| got ask

operator| 1 | 93252 - 93254 |1

operator| sending...

operator| msg is: 1

operator| got ask

|93254|-|1|(1)

operator| 0 | -1 - 93254 |1

operator| cheking...

operator| msg is: 1

operator| got ask

|93254|-|(0)

operator| 0 | -1 - 93252 |1

operator| cheking...

operator| no messenges

> frost (1000 ms)

(Это кусок вывода, там этих логов страниц на 5 было)

**Вывод**

Эта лабораторная работа столь обширна и многообразна, что я, несомненно, о чем-то да забуду здесь написать. В ходе ее выполнения я закрепил свои знания о взаимодействиях процессов, впервые написал свой мьютекс, очередь сообщений, функцию работы со временем. Как нетрудно заметить, в итоге я применил в ней все приемы из предыдущих лабораторных работ (кроме, разве что, динамических библиотек). Как я уже заметил выше, меня несколько удивило, что в unix функция sleep работает с целыми секундами. Вроде бы я где-то читал, что более низкоуровневые функции могут принимать даже наносекунды, так что отчего у sleep столь низкая точность мне решительно непонятно. Работая же с оператором очереди, я немного расстроился оттого что в итоге мне пришлось его лочить при каждом обращении, и я подумал, что неплохо было бы написать такую версию, которая бы на каждый процесс, что с ней работает, открывала свои коммуникационные каналы. Так я и выбрал тему для курсовой. Что ж, посмотрим, что у меня в итоге выйдет.