Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Лабораторная работа №3 по курсу**

**«Операционные системы»**

**Управление потоками в ОС**

Студент: Гаврилов Максим Сергеевич

Группа: М8О–206Б–20

Вариант: 7

Преподаватель: Соколов Андрей Алексеевич

Оценка: \_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_

Москва, 2020.

**Постановка задачи**

## Цель работы

Целью является приобретение практических навыков в:

* Управление потоками в ОС
* Обеспечение синхронизации между потоками

## Задание

Составить программу на языке Си, обрабатывающую данные в многопоточном режиме. При обработке использовать стандартные средства создания потоков операционной системы (Windows/Unix). Ограничение потоков должно быть задано ключом запуска вашей программы.

Так же необходимо уметь продемонстрировать количество потоков, используемое вашей программой с помощью стандартных средств операционной системы.

В отчете привести исследование зависимости ускорения и эффективности алгоритма от входящих данных и количества потоков. Получившиеся результаты необходимо объяснить.

**Условия задачи**

Два человека играют в кости. Правила игры следующие: каждый игрок делает бросок 2-ух костей K раз; побеждает тот, кто выбросил суммарно большее количество очков. Задача программы экспериментально определить шансы на победу каждого из игроков. На вход программе подается K, какой сейчас тур, сколько очков суммарно у каждого из игроков и количество экспериментов, которые должна произвести программа

**Общие сведения о программе**

Программа компилируется из файлов main.c, child\_any.c, tramsformres.h. Используется заголовочные файлы: stdio.h, unistd.h, stdlib.h, string.h, fcntl.h, pthread.h, time.h. 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. **Open –** преобразует путь к файлу в файловый дескриптор.
4. **Close –** закрывает файловый дескриптор, который после этого не ссылается ни на один и файл и может быть использован повторно.
5. **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
6. **Fork –** порождает новый процесс (процесс-потомок), который почти идентичен порождающему процессу-родителю.

Синхронизация работы производится с помощью пересылки через pipe-ы специальных сообщений-маркеров. Таким образом родительский процесс может быть осведомлен о том, что ребенок завершил выполнение того или иного действия. Пересылка длинных чисел через pipe-ы осуществляется с помощью предварительного перевода их в формат строки, что крайне неэффективно и, по сути, бессмысленно.

**Общий метод и алгоритм решения**.

Для реализации поставленной задачи необходимо:

1. Найти способ синхронизации потоков  
   Я выбрал достаточно примитивный метод. Если (к примеру) родительский процесс хочет вывести на экран какое-то сообщение, то он должен подождать, пока дети не завершат свой вывод. Тогда родительский процесс выполняет вызов write. Ребенок же по завершении вывода записывает в файловый дескриптор pipe-а какой-либо символ. Когда родительский процесс прочитает по символу из pipe-ов всех детей, он может приступать к выводу.  
   Этот метод допускает некую модификацию (к примеру, в канал можно записывать маркер состояния), однако, для более сложных программ лучше использовать более высокоуровневые методы синхронизации
2. Найти способ удобно хранить в родителе все необходимые данные для взаимодействия с детьми.   
   при начале разделения родительский процесс выполняет n (n получено из аргумента) вызовов fork. При этом с ребенком, созданном на шаге I ассоциируется pipe под индексом I из заранее созданного набора pipe-ов. Также на всякий случай создается массив pid-ов детей.
3. Придумать оптимальную схему распределения вычислений между детьми.  
   Каждый ребенок проводит K/n (где K – общее число экспериментов, а n – число детей) экспериментов, и возвращает число экспериментов, в которых победил первый игрок.

**Основные файлы программы**

**Main.c**

|  |
| --- |
| #include <math.h>  #include <stdio.h>  #include <stdlib.h>  #include <time.h>  #include <unistd.h>  #include <string.h>  #include <pthread.h>  #include "tramsformres.h"  #define IN 1  #define OUT 0  #define TESTER\_LIMIT 1009  //система разделения процессов  //генерирует процесс родителя и n детей  //программы родителей и детей хранятнся отдельно  int main**(**int argc**,**char**\*\*** argv**)**  **{**  srand**(**time**(NULL));**  //получаем значение ключа  printf**(**"argc-> %d\n"**,**argc**);**  **if(**argc **!=** 2**)**  **{**  **return** **-**10**;** //неверное кол-во аргументов  **}**  **if(**strlen**(**argv**[**1**])<**2**)**  **{**  **return** **-**11**;** //невернаяя стурктура ключа  **}**  **if(**argv**[**1**][**0**]** **!=** '-'**)**  **{**  **return** **-**11**;** //неверная структура ключа  **}**  int i**=**strlen**(**argv**[**1**])** **-** 1**,** inv **=** 1**;**  int num **=** 0**;**  **while(**i**>**0**)**  **{**  **if(**argv**[**1**][**i**]** **<** '0' **||** argv**[**1**][**i**]** **>** '9'**)**  **{**  **return** 0**;** //в ключе не число  **}**  // printf("|%c|\n",argv[1][i]);  num **+=** inv**\*(**argv**[**1**][**i**]-**'0'**);** //сборка числового ключа  inv**\*=**10**;**  **--**i**;**  **}**  **if(**num**<=**0 **||** num**>=**256**)**  **{**  printf**(**"incorrect proc num\n"**);**  **return** **-**18**;**  **}**  printf**(**"procon-> %d\n"**,**num**);**  //значение ключа собрано  //создаем pipe-ы    int fdPC**[**num**][**2**];**  int fdCP**[**num**][**2**];**  **for(**i**=**0**;**i**<**num**;++**i**)**  **{**  pipe**(**fdPC**[**i**]);**  pipe**(**fdCP**[**i**]);**  **}**  //начинаем разделение. идея в том чтобы в итоге мы имели один родительский процесс и num ЕГО детей  int id**[**num**];** //это будет массив с id детей  **for(**i**=**0**;**i**<**num**;++**i**)**  **{**  id**[**i**]** **=** 0**;**  **}**  //!invalid! первый элемент уже проинициализирован чтобы можно было потом одним циклом записать все в этот массив  printf**(**"<-| initiating process separtion\n"**);**  **for(**i**=**0**;**i**<**num**;++**i**)**  **{**  **if(**i**!=** 0 **&&** id**[**i**-**1**]** **==** 0**)**  **{**  **break;**  **}**  id**[**i**]** **=** fork**();**  **}**  **if(**id**[**i**-**1**]** **==** 0**)** //вывод инфы по процессам, вывод синхронизирован  **{**  int localid **=** i**-**1**;** //обозначает, на каком шаге (при отсчете от нуля) разделения отделился этот процесс, используется при общении через pipe  printf**(**"[%d] child process started. local ID = %d\n"**,**getpid**(),**localid**);**  write**(**fdCP**[**localid**][**IN**],**"1"**,**1**);**  //место для запуска программы ребенка  //printf("[%d] sending: %d <-SEND|GET-> %d\n",getpid(),fdCP[localid][IN],fdPC[localid][OUT]);  execl**(**"child\_any.out"**,**transform**(**fdCP**[**localid**][**IN**]),**transform**(**fdPC**[**localid**][**OUT**]),NULL);**  //printf("[%d] process detached\n",getpid());  **}**  **else** //код родителя (может вынесу в отдельный файл еще)  **{**  char mark**=**'0'**;**  **for(**i**=**0**;**i**<**num**;++**i**)**//в этом цикле происходит чтение из pipe ов связи с детьми, т.е. родительский процесс паузится, пока все дети не пошлют сигнал  **{**  read**(**fdCP**[**i**][**OUT**],&**mark**,**1**);**  **}**//отчет процессов из этой проги, перед разделением  printf**(**"[%d] parent process operating, childs:\n"**,**getpid**());**  **for(**i**=**0**;**i**<**num**;++**i**)**  **{**  **if(**id**[**i**]==**0**)**  **{**  **break;**  **}**  printf**(**"[%d] child: [%d] lid: [%d]\n"**,**getpid**(),**id**[**i**],**i**);**  **}**  **for(**i**=**0**;**i**<**num**;++**i**)**  **{**  write**(**fdPC**[**i**][**IN**],**"2"**,**1**);**  **}**  **for(**i**=**0**;**i**<**num**;++**i**)**//в этом цикле происходит чтение из pipe ов связи с детьми, т.е. родительский процесс паузится, пока все дети не пошлют сигнал  **{**  read**(**fdCP**[**i**][**OUT**],&**mark**,**1**);**  **}**//отчет процессов из их отдельных программ, после разделения  //блок проверки вспомогательных функций  //особо не нужен, но при работе выглядит красиво :)  printf**(**"<-| testing auxiliary functions| "**);**  int slider **=** 0**;**  **while(**slider**<=**TESTER\_LIMIT**)**  **{**  **if(**back\_transform**(**transform**(**slider**))!=**slider**)**  **{**  printf**(**"incorrect work detected\n"**);**  **return(-**101**);**  **}**  **++**slider**;**  **}**  printf**(**"functions capability confirmed\n"**);**  //завершение проверки    long long K**;**  long long a**,**b**;**  long long C**;**  long long N**;**  printf**(**"input number of rouwnds\n|-> "**);**  scanf**(**"%lld"**,&**K**);**  printf**(**"input current round\n|-> "**);**  scanf**(**"%lld"**,&**C**);**  printf**(**"input current score\n|-> "**);**  scanf**(**"%lld %lld"**,&**a**,&**b**);**  printf**(**"input number of tries\n|-> "**);**  scanf**(**"%lld"**,&**N**);**  int count\_KmC **=** K**-**C**;**    long long count\_a **=** N**/**num **+** 1**;**  long long count\_b **=** N**%**num**;**  long long count\_c **=** count\_b**;**  // printf("a-> %d|%s|%d\n",count\_a,transform(count\_a),back\_transform(transform(count\_a)));  // printf("b-> %d|%s|%d\n",count\_b,transform(count\_b),back\_transform(transform(count\_b)));  **for(**i **=** 0**;**i**<**num**;++**i**)**  **{**  **if(**count\_b **==** 0**)** //распределение остатка по потокам (по 1 вычислению каждому, пока доп. вычислений не останется)  **{**  count\_a**-=**1**;**  count\_b**--;**  **}**  **else**  **{**  **if(**count\_b **>** 0**)**  count\_b**--;**  **}**  printf**(**"sent-> %lld\n"**,**count\_a**);**  write**(**fdPC**[**i**][**IN**],**transform**(**count\_a**),**strlen**(**transform**(**count\_a**))+**1**);** //пересылаем число попрыток, которые должен промоделировать процесс  write**(**fdPC**[**i**][**IN**],**transform**(**count\_KmC**),**strlen**(**transform**(**count\_KmC**))+**1**);** //пересылаем число попрыток, которые должен промоделировать процесс  write**(**fdPC**[**i**][**IN**],**transform**(**a**),**strlen**(**transform**(**a**))+**1**);** //пересылаем число попрыток, которые должен промоделировать процесс  write**(**fdPC**[**i**][**IN**],**transform**(**b**),**strlen**(**transform**(**b**))+**1**);** //пересылаем число попрыток, которые должен промоделировать процесс  **}**  int count\_recive **=** count\_a**+**1**;**  char getter**;**  int ittl **=** 0**;**  **for(**i**=**0**;**i**<**num**;++**i**)**//в этом цикле происходит чтение из pipe ов связи с детьми, т.е. родительский процесс паузится, пока все дети не пошлют сигнал  **{**  read**(**fdCP**[**i**][**OUT**],&**mark**,**1**);**  **}**  long long first\_wins\_total **=** 0**;**  printf**(**"--------------------\n"**);**  **for(**i **=** 0**;**i**<**num**;++**i**)**  **{**  long long j**=**0**;**  char inp **=** **-**1**;**  char**\*** wins\_in\_str **=** **(**char**\*)**malloc**(sizeof(**char**)\***1**);**  **while(**inp **!=** 0**)**  **{**  // printf("from [%d] reading... \n",i);  wins\_in\_str **=** **(**char**\*)**realloc**(**wins\_in\_str**,sizeof(**char**)\*(**j**+**1**));**  read**(**fdCP**[**i**][**OUT**],&**inp**,**1**);**  wins\_in\_str**[**j**]** **=** inp**;**  // printf("read -> %c|%d \n",inp,inp);  j**++;**  **}**  //wins\_in\_str[j] = 0;  long long wins\_normal **=** back\_transform**(**wins\_in\_str**);**  free**(**wins\_in\_str**);**  first\_wins\_total **+=** wins\_normal**;**  printf**(**"\n--------------------\ngot %s|%lld wins\n--------------------\n"**,**wins\_in\_str**,**wins\_normal**);**  **}**  printf**(**"total 1st wins: %lld of %lld games\n"**,**first\_wins\_total**,**N**);**  double chance **=** **(**double**)**first\_wins\_total**/(**double**)**N**;**  printf**(**"first wins chance: %lf\n"**,**chance**);**  **for(**i**=**0**;**i**<**num**;++**i**)**//закрытие всех pipe-ов  **{**  close**(**fdPC**[**i**][**OUT**]);**  close**(**fdCP**[**i**][**IN**]);**  **}**  **}**  **return** 0**;**  **}** |

**Child\_any.c**

|  |
| --- |
| #include <math.h>  #include <stdio.h>  #include <stdlib.h>  #include <time.h>  #include <unistd.h>  #include <string.h>  #include "tramsformres.h"  int main**(**int argc**,**char**\*\*** argv**)**  **{**  srand**(**time**(NULL));**  int fd\_to\_par **=** back\_transform**(**argv**[**0**]);**  int fd\_from\_par **=** back\_transform**(**argv**[**1**]);**  char mark**;**  read**(**fd\_from\_par**,&**mark**,**1**);**  printf**(**"[%d] process detached | %d <- SEND|GET -> %d\n"**,**getpid**(),**fd\_to\_par**,**fd\_from\_par**);**  write**(**fd\_to\_par**,**"1"**,**1**);**  char inp **=** **-**1**;**  int i**=**0**;**  char**\*** count\_in\_str **=** **(**char**\*)**malloc**(sizeof(**char**)\***1**);**  char**\*** rounds\_in\_str **=** **(**char**\*)**malloc**(sizeof(**char**)\***1**);**  **while(**inp **!=** 0**)**  **{**  count\_in\_str **=** **(**char**\*)**realloc**(**count\_in\_str**,sizeof(**char**)\*(**i**+**1**));**  read**(**fd\_from\_par**,&**inp**,**1**);**  count\_in\_str**[**i**]** **=** inp**;**  i**++;**  **}**  count\_in\_str**[**i**]** **=** 0**;**  inp **=** **-**1**;**i**=**0**;**  **while(**inp **!=** 0**)**  **{**  rounds\_in\_str **=** **(**char**\*)**realloc**(**rounds\_in\_str**,sizeof(**char**)\*(**i**+**1**));**  read**(**fd\_from\_par**,&**inp**,**1**);**  rounds\_in\_str**[**i**]** **=** inp**;**  i**++;**  **}**  rounds\_in\_str**[**i**]** **=** 0**;**    long long count\_of\_tries **=** back\_transform**(**count\_in\_str**);**  long long count\_of\_rounds **=** back\_transform**(**rounds\_in\_str**);**  free**(**count\_in\_str**);**  free**(**rounds\_in\_str**);**  i**=**0**;**inp **=** **-**1**;**  char**\*** a\_count **=** **(**char**\*)**malloc**(sizeof(**char**)\***1**);**  char**\*** b\_count **=** **(**char**\*)**malloc**(sizeof(**char**)\***1**);**  **while(**inp **!=** 0**)**  **{**  a\_count **=** **(**char**\*)**realloc**(**a\_count**,sizeof(**char**)\*(**i**+**1**));**  read**(**fd\_from\_par**,&**inp**,**1**);**  a\_count**[**i**]** **=** inp**;**  i**++;**  **}**  a\_count**[**i**]** **=** 0**;**  inp **=** **-**1**;**i**=**0**;**  **while(**inp **!=** 0**)**  **{**  b\_count **=** **(**char**\*)**realloc**(**b\_count**,sizeof(**char**)\*(**i**+**1**));**  read**(**fd\_from\_par**,&**inp**,**1**);**  b\_count**[**i**]** **=** inp**;**  i**++;**  **}**  b\_count**[**i**]** **=** 0**;**    long long a\_ **=** back\_transform**(**a\_count**);**  long long b\_ **=** back\_transform**(**b\_count**);**  free**(**a\_count**);**  free**(**b\_count**);**  printf**(**"[%d] %lld attts to generate %lld rounds | start values are %lld %lld\n"**,**getpid**(),**count\_of\_tries**,**count\_of\_rounds**,**a\_**,**b\_**);**  write**(**fd\_to\_par**,**"1"**,**1**);**  srand**(**time**(NULL)\***getpid**());**  long long first\_wins **=** 0**;**  **for(**long long j **=** 0**;** j**<**count\_of\_tries**;++**j**)**  **{**  long long a **=** a\_**,** b**=**b\_**;**  //printf("in loop, %lld\n",j);  **for(**long long i**=**0**;**i**<**count\_of\_rounds**;++**i**)**  **{**  a **+=** rand**()%**6 **+** 1 **+** rand**()%**6 **+** 1**;**  b **+=** rand**()%**6 **+** 1 **+** rand**()%**6 **+** 1**;**  **}**    //printf("%lld | %lld\n",a,b);  **if(**a**>**b**)**  first\_wins **+=** 1**;**  **}**    write**(**fd\_to\_par**,**transform**(**first\_wins**),**strlen**(**transform**(**first\_wins**))+**1**);**  char nul**[**1**];**nul**[**0**]** **=** 0**;**  //write(fd\_to\_par,nul,1);  printf**(**"loop finale\n"**);**  //write(fd\_to\_par,"1",1);  close**(**fd\_from\_par**);**  close**(**fd\_to\_par**);**  printf**(**"child: stop\n"**);**  **return** 0**;**  **}** |

**Transformers.h**

|  |
| --- |
| char**\*** transform**(**int inp**)**  **{**  int i**=**0**;**  char**\*** str **=** **(**char**\*)**malloc**(sizeof(**char**)\***1**);**  **while(**inp **!=** 0**)**  **{**  str **=** **(**char**\*)**realloc**(**str**,sizeof(**char**)\*(**i**+**1**));**  str**[**i**]** **=** **(**inp**%**10**)+**'0'**;**  **++**i**;**  inp**/=**10**;**  **}**  str**[**i**]** **=** 0**;**  **return** str**;**  **}**  int back\_transform**(**char**\*** inp**)**  **{**  int res **=** 0**;**  int mult **=** 1**;**  int i **=** 0**;**  **while(**inp**[**i**]!=** 0**)**  **{**  res **+=** **(**inp**[**i**]-**'0'**)\***mult**;**  **++**i**;**  mult**\*=**10**;**  **}**  //free(inp);  **return** res**;**  **}** |

**Пример работы**

**max@max-Swift:~/Рабочий стол/OS/lab3/prog$ make**

*gcc --std=c99 main.c -o exethr.out*

*gcc --std=c99 child\_any.c -o child\_any.out*

**max@max-Swift:~/Рабочий стол/OS/lab3/prog$ ./exethr.out -4**

*procon-> 4*

*<-| initiating process separtion*

*[6550] child process started. local ID = 0*

*[6551] child process started. local ID = 1*

*[6553] child process started. local ID = 3*

*[6552] child process started. local ID = 2*

*[6549] parent process operating, childs:*

*[6549] child: [6550] lid: [0]*

*[6549] child: [6551] lid: [1]*

*[6549] child: [6552] lid: [2]*

*[6549] child: [6553] lid: [3]*

*[6550] process detached | 6 <- SEND|GET -> 3*

*[6551] process detached | 10 <- SEND|GET -> 7*

*[6553] process detached | 18 <- SEND|GET -> 15*

*[6552] process detached | 14 <- SEND|GET -> 11*

*<-| testing auxiliary functions| functions capability confirmed*

*input number of rounds*

*|-> 100*

*input current round*

*|-> 10*

*input current score*

*|-> 56 11*

*input number of tries*

*|-> 100000*

*sent-> 25000*

*sent-> 25000*

*[6550] 25000 atts to generate 90 rounds | start values are 56 11*

*[6551] 25000 atts to generate 90 rounds | start values are 56 11*

*sent-> 25000*

*sent-> 25000*

*[6552] 25000 atts to generate 90 rounds | start values are 56 11*

*[6553] 25000 atts to generate 90 rounds | start values are 56 11*

*--------------------*

*got |22908 wins*

*--------------------*

*got |22939 wins*

*--------------------*

*got |22967 wins*

*--------------------*

*got |22858 wins*

*--------------------*

*total 1st wins: 91672 of 100000 games*

*first wins chance: 0.916720*

**max@max-Swift:~/Рабочий стол/OS/lab3/prog$ ./exethr.out -10**

*procon-> 10*

*<-| initiating process separtion*

*[6559] child process started. local ID = 0*

*[6560] child process started. local ID = 1*

*[6562] child process started. local ID = 3*

*[6563] child process started. local ID = 4*

*[6561] child process started. local ID = 2*

*[6565] child process started. local ID = 6*

*[6564] child process started. local ID = 5*

*[6568] child process started. local ID = 9*

*[6566] child process started. local ID = 7*

*[6567] child process started. local ID = 8*

*[6558] parent process operating, childs:*

*[6558] child: [6559] lid: [0]*

*[6558] child: [6560] lid: [1]*

*[6558] child: [6561] lid: [2]*

*[6558] child: [6562] lid: [3]*

*[6558] child: [6563] lid: [4]*

*[6558] child: [6564] lid: [5]*

*[6558] child: [6565] lid: [6]*

*[6558] child: [6566] lid: [7]*

*[6558] child: [6567] lid: [8]*

*[6558] child: [6568] lid: [9]*

*[6559] process detached | 6 <- SEND|GET -> 3*

*[6560] process detached | 10 <- SEND|GET -> 7*

*[6562] process detached | 18 <- SEND|GET -> 15*

*[6561] process detached | 14 <- SEND|GET -> 11*

*[6565] process detached | 30 <- SEND|GET -> 27*

*[6564] process detached | 26 <- SEND|GET -> 23*

*[6567] process detached | 38 <- SEND|GET -> 35*

*[6563] process detached | 22 <- SEND|GET -> 19*

*[6568] process detached | 42 <- SEND|GET -> 39*

*[6566] process detached | 34 <- SEND|GET -> 31*

*<-| testing auxiliary functions| functions capability confirmed*

*input number of rounds*

*|-> 50*

*input current round*

*|-> 40*

*input current score*

*|-> 66 67*

*input number of tries*

*|-> 100000*

*sent-> 10000*

*sent-> 10000*

*sent-> 10000*

*sent-> 10000*

*sent-> 10000*

*sent-> 10000*

*sent-> 10000*

*sent-> 10000*

*[6564] 10000 atts to generate 10 rounds | start values are 66 67*

*[6566] 10000 atts to generate 10 rounds | start values are 66 67*

*[6561] 10000 atts to generate 10 rounds | start values are 66 67*

*[6559] 10000 atts to generate 10 rounds | start values are 66 67*

*sent-> 10000*

*sent-> 10000*

*[6568] 10000 atts to generate 10 rounds | start values are 66 67*

*[6565] 10000 atts to generate 10 rounds | start values are 66 67*

*[6563] 10000 atts to generate 10 rounds | start values are 66 67*

*[6567] 10000 atts to generate 10 rounds | start values are 66 67*

*[6560] 10000 atts to generate 10 rounds | start values are 66 67*

*[6562] 10000 atts to generate 10 rounds | start values are 66 67*

*--------------------*

*got |4469 wins*

*--------------------*

*got |4434 wins*

*--------------------*

*got |4426 wins*

*--------------------*

*got |4532 wins*

*--------------------*

*got |4516 wins*

*--------------------*

*got |4459 wins*

*--------------------*

*got |4497 wins*

*--------------------*

*got |4420 wins*

*--------------------*

*got |4475 wins*

*--------------------*

*got |4429 wins*

*--------------------*

*total 1st wins: 44657 of 100000 games*

*first wins chance: 0.446570*

**max@max-Swift:~/Рабочий стол/OS/lab3/prog$ ./exethr.out -10**

*procon-> 10*

*<-| initiating process separtion*

*[6609] child process started. local ID = 0*

*[6610] child process started. local ID = 1*

*[6612] child process started. local ID = 3*

*[6614] child process started. local ID = 5*

*[6613] child process started. local ID = 4*

*[6615] child process started. local ID = 6*

*[6617] child process started. local ID = 8*

*[6611] child process started. local ID = 2*

*[6618] child process started. local ID = 9*

*[6616] child process started. local ID = 7*

*[6608] parent process operating, childs:*

*[6608] child: [6609] lid: [0]*

*[6608] child: [6610] lid: [1]*

*[6608] child: [6611] lid: [2]*

*[6608] child: [6612] lid: [3]*

*[6608] child: [6613] lid: [4]*

*[6608] child: [6614] lid: [5]*

*[6608] child: [6615] lid: [6]*

*[6608] child: [6616] lid: [7]*

*[6608] child: [6617] lid: [8]*

*[6608] child: [6618] lid: [9]*

*[6609] process detached | 6 <- SEND|GET -> 3*

*[6610] process detached | 10 <- SEND|GET -> 7*

*[6617] process detached | 38 <- SEND|GET -> 35*

*[6611] process detached | 14 <- SEND|GET -> 11*

*[6612] process detached | 18 <- SEND|GET -> 15*

*[6613] process detached | 22 <- SEND|GET -> 19*

*[6618] process detached | 42 <- SEND|GET -> 39*

*[6615] process detached | 30 <- SEND|GET -> 27*

*[6614] process detached | 26 <- SEND|GET -> 23*

*[6616] process detached | 34 <- SEND|GET -> 31*

*<-| testing auxiliary functions| functions capability confirmed*

*input number of rounds*

*|-> ^C*

*max@max-Swift:~/Рабочий стол/OS/lab3/prog$ ./exethr.out -2*

*procon-> 2*

*<-| initiating process separtion*

*[6624] child process started. local ID = 0*

*[6625] child process started. local ID = 1*

*[6623] parent process operating, childs:*

*[6623] child: [6624] lid: [0]*

*[6623] child: [6625] lid: [1]*

*[6624] process detached | 6 <- SEND|GET -> 3*

*[6625] process detached | 10 <- SEND|GET -> 7*

*<-| testing auxiliary functions| functions capability confirmed*

*input number of rounds*

*|-> 10*

*input current round*

*|-> 9*

*input current score*

*|-> 9 54*

*input number of tries*

*|-> 100*

*sent-> 50*

*sent-> 50*

*[6624] 50 atts to generate 1 rounds | start values are 9 54*

*[6625] 50 atts to generate 1 rounds | start values are 9 54*

*--------------------*

*got |0 wins*

*--------------------*

*got |0 wins*

*--------------------*

*total 1st wins: 0 of 100 games*

*first wins chance: 0.000000*

**max@max-Swift:~/Рабочий стол/OS/lab3/prog$**

**Оценка выигрыша по эффективности**

Проверим скорость выполнения с помощью команды time.

**max@max-Swift:~/Рабочий стол/OS/lab3/prog$ time ./exethr.out -1 < file\_inp > file\_res**  
real 0m2,410s

user 0m0,004s

sys 0m0,000s

**max@max-Swift:~/Рабочий стол/OS/lab3/prog$ time ./exethr.out -2 < file\_inp > file\_res**

real 0m1,241s

user 0m0,001s

sys 0m0,003s

**max@max-Swift:~/Рабочий стол/OS/lab3/prog$ time ./exethr.out -3 < file\_inp > file\_res**

real 0m1,120s

user 0m0,001s

sys 0m0,004s

**max@max-Swift:~/Рабочий стол/OS/lab3/prog$ time ./exethr.out -10 < file\_inp > file\_res**

real 0m1,142s

user 0m0,000s

sys 0m0,006s

Даже без детального анализа видно, что разделение на три (два дочерних) потока дает значительный (в 2 раза) прирост скорости выполнения. Разделение на 4 потока также дает прирост, однако уже небольшой. Так как всего в моем компьютере лишь 2 ядра по два потока на каждое, разделение на более чем четыре потока в принципе не может дать большего выигрыша.

Количество потоков, используемое программой можно продемонстрировать, к примеру, с помощью **ps**

**max@max-Swift:~/Рабочий стол/OS/lab3/prog$ ./exethr.out -4**

*procon-> 4*

*<-| initiating process separtion*

*[8522] child process started. local ID = 1*

*[8521] child process started. local ID = 0*

*[8523] child process started. local ID = 2*

*[8524] child process started. local ID = 3*

*[8520] parent process operating, childs:*

*[8520] child: [8521] lid: [0]*

*[8520] child: [8522] lid: [1]*

*[8520] child: [8523] lid: [2]*

*[8520] child: [8524] lid: [3]*

*[8521] process detached | 6 <- SEND|GET -> 3*

*[8523] process detached | 14 <- SEND|GET -> 11*

*[8522] process detached | 10 <- SEND|GET -> 7*

*[8524] process detached | 18 <- SEND|GET -> 15*

*<-| testing auxiliary functions| functions capability confirmed*

*input number of rounds*

*|-> ^Z*

*[1]+ Остановлен ./exethr.out -4*

**max@max-Swift:~/Рабочий стол/OS/lab3/prog$ ps**

*PID TTY TIME CMD*

*6495 pts/0 00:00:00 bash*

*8520 pts/0 00:00:00 exethr.out*

*8521 pts/0 00:00:00 child\_any.out* <child 1

*8522 pts/0 00:00:00 child\_any.out* <child 2

*8523 pts/0 00:00:00 child\_any.out* <child 3

*8524 pts/0 00:00:00 child\_any.out* <child 4

*8530 pts/0 00:00:00 ps*

**max@max-Swift:~/Рабочий стол/OS/lab3/prog$**Стоит также отметить, что в ходе работы процессы выводят много системной информации на экран, и каждому сообщению предшествует значение pid выполняющего вывод процесса, так что в многопоточности программы можно убедиться (и посчитать количество потоков) и без каких-либо специальных средств

**Вывод**

В моей программе полно всяческих неэффективных решений, к тому же система взаимодействия между процессами неустойчива к внештатным ситуациям. Несмотря на это, она справляется со своей основной задачей и успешно демонстрирует основные преимущества многопоточной обработки. В ходе работы над лабораторной работой я также пронаблюдал весьма очевидное, но все же достаточно занимательное явление, связанное с многопоточностью: если в созданных практически одновременно дочерних потоках проинициализировать генератор псевдослучайных чисел, как это обыкновенно принято, значением точного времени, то генераторы во всех потоках будут выдавать одни и те же значения. Во избежание этого я использовал для инициализации генератора значение точного времени, умноженное на pid.