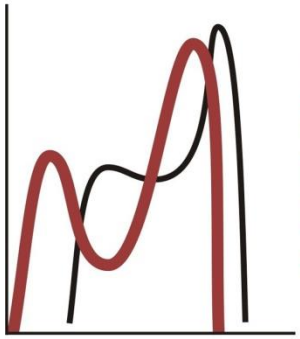


ИНСТИТУТ
МАТЕМАТИКИ
МЕХАНИКИ
КОМПЬЮТЕРНЫХ
НАУК

имени И.И. Воровича —

# Архитектура компьютера и операционные системы

## Лекция 15. Управление процессами и потоками

Андреева Евгения Михайловна
доцент кафедры информатики и вычислительного эксперимента


# План лекции

- Потоки
- Реализация потоков
- Механизмы взаимодействия
- Разделяемая память
- Каналы

ИНСТИТУТ МАТЕМАТИКИ МЕХАНИКИ КОМПЬЮТЕРНЫХ НАУК имени И.И. Воровича

# Зачем нужны потоки

Ввести массив A

Ожидание ввода A

Ввести массив B

Ожидание ввода B

Ввести массив C

Ожидание ввода C

A=A+B

C=A+C

Вывести массив C

Ожидание вывода C

Ожидание ввода A и B

A=A+B

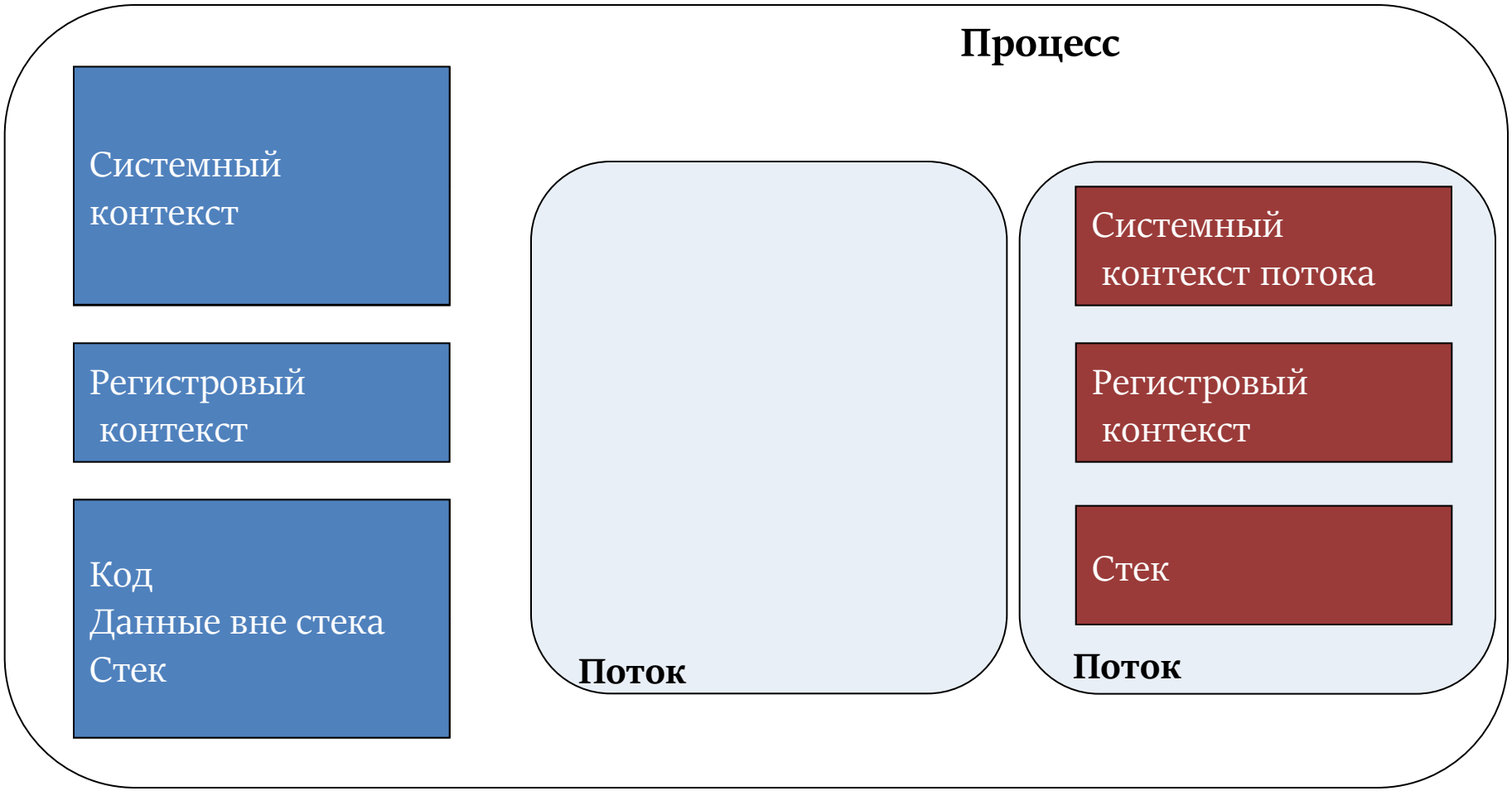
ИНСТИТУТ МАТЕМАТИКИ МЕХАНИКИ КОМПЬЮТЕРНЫХ НАУК имени И.И. Воровича

# Решение 1

| Процесс 1 | | Процесс 2 |
|---|---|---|
| Создание процесса 2 | Переключение контекста | |
| | | Создание общей памяти |
| | Переключение контекста | |
| | | Ожидание ввода A и B |
| Создание общей памяти | | |
| Ввести массив A | | |
| Ожидание ввода A | | |
| Ввести массив B | | |
| Ожидание ввода B | | |
| | Переключение контекста | |
| Ввести массив C | | |
| | | A=A+B |
| Ожидание ввода C | | |
| | Переключение контекста | |
| C=A+C | | |
| Вывести массив C | | |
| Ожидание вывода C | | |

ИНСТИТУТ МАТЕМАТИКИ МЕХАНИКИ КОМПЬЮТЕРНЫХ НАУК имени И.И. Воровича

# Потоки



ИНСТИТУТ МАТЕМАТИКИ МЕХАНИКИ КОМПЬЮТЕРНЫХ НАУК имени И.И. Воровича

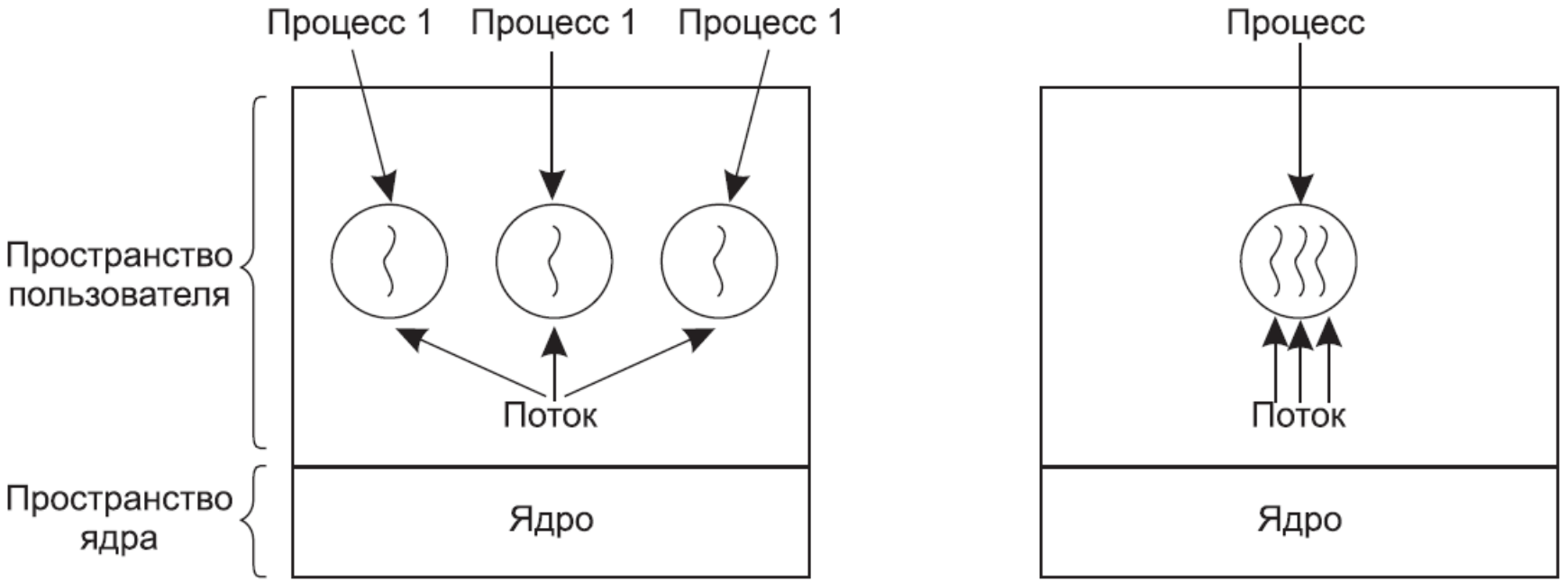
# Формальное определение

- Поток выполнения (англ. thread — нить) — наименьшая единица обработки, исполнение которой может быть назначено ядром операционной системы.

- Потоки (облегченные процессы) – «механизм» параллельного или квазипараллельного выполнения фрагментов кода, использующих общие ресурсы с целью эффективной реализации задачи

ИНСТИТУТ МАТЕМАТИКИ МЕХАНИКИ КОМПЬЮТЕРНЫХ НАУК имени И.И. Воровича

# Многопроцессорный и многопоточный режим



# Процесс vs Поток

- Атрибуты процесса
  - Адресное пространство
  - Глобальные переменные
  - Открытые файлы
  - Дочерние процессы
  - Необработанные аварийные сигналы
  - Сигналы и обработчики сигналов
  - Учетная информация
- Атрибуты потока
  - Счетчик команд
  - Регистры
  - Стек
  - Состояние

ИНСТИТУТ МАТЕМАТИКИ МЕХАНИКИ КОМПЬЮТЕРНЫХ НАУК имени И.И. Воровича

# Решение 2

| Поток 1 | | Поток 2 |
|---|---|---|
| Создание потока 2 | | |
| | Переключение контекста | |
| | | Ожидание ввода A и B |
| | Переключение контекста | |
| Ввести массив A | | |
| Ожидание ввода A | | |
| Ввести массив B | | |
| Ожидание ввода B | | |
| | Переключение контекста | |
| Ввести массив C | | |
| | | A=A+B |
| Ожидание ввода C | | |
| | Переключение контекста | |
| C=A+C | | |
| Вывести массив C | | |
| Ожидание вывода C | | |

ИНСТИТУТ МАТЕМАТИКИ МЕХАНИКИ КОМПЬЮТЕРНЫХ НАУК имени И.И. Воровича

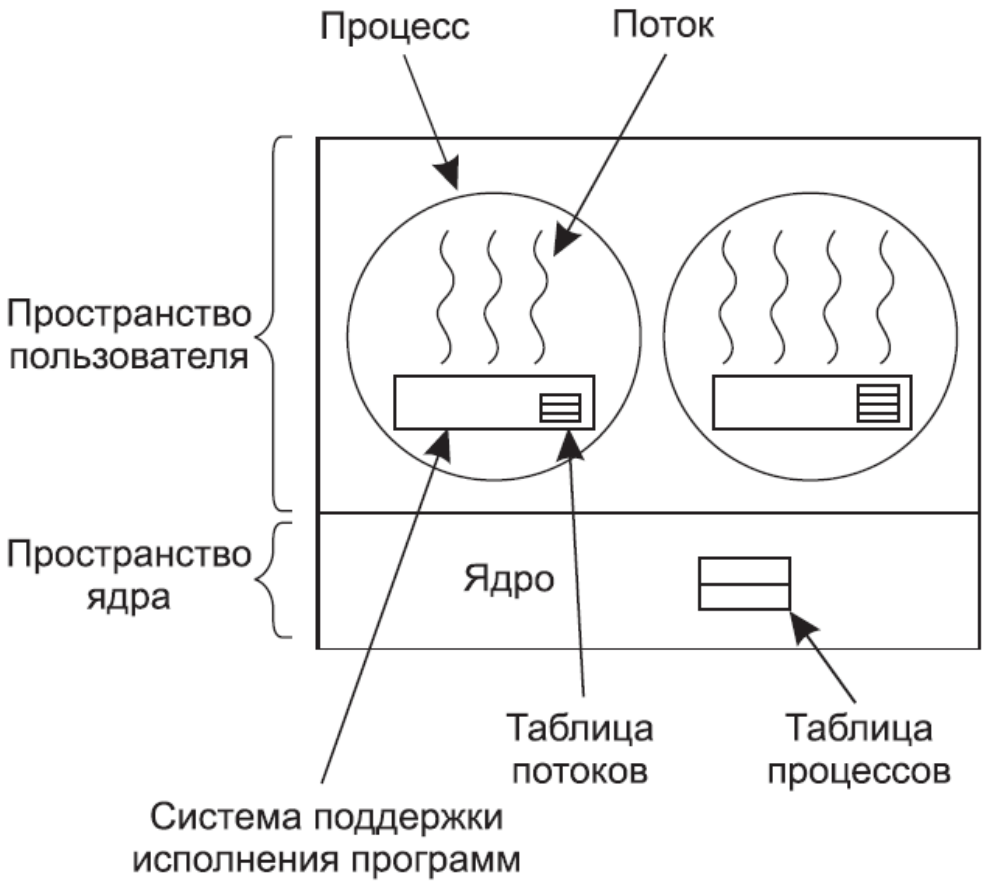
# Состояния процесса

- Исполнение:
  - хотя бы один поток находится в состоянии «Исполнение»
- Готовность:
  - хотя бы один поток находится в состоянии «Готовность»
  - никто не находится в состоянии «Исполнение»
- Ожидание:
  - хотя бы один поток находится в состоянии «Ожидание»
  - никто не находится в состоянии «Исполнение» и «Готовность»
- Завершение:
  - Все потоки находятся в состоянии «Завершение»

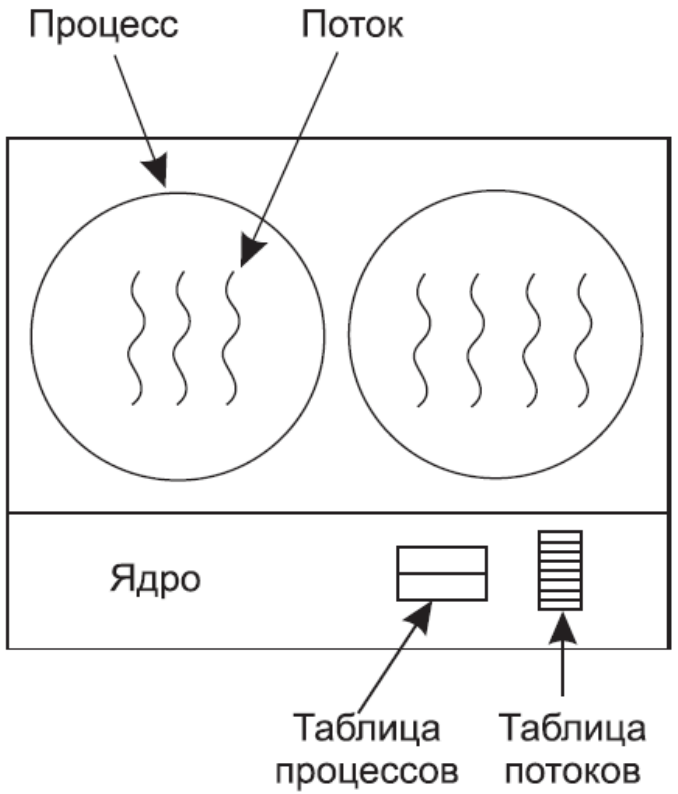
ИНСТИТУТ МАТЕМАТИКИ МЕХАНИКИ КОМПЬЮТЕРНЫХ НАУК имени И.И. Воровича

# Реализация потоков

- **В пользовательском пространстве**




- **В ядре**



ИНСТИТУТ МАТЕМАТИКИ МЕХАНИКИ КОМПЬЮТЕРНЫХ НАУК имени И.И. Воровича

# Реализация потоков в пользовательском пространстве

- Ядро о потоках ничего не знает
- Процедура, сохраняющая информацию о потоке, и планировщик являются локальными процедурами, и их вызов существенно более эффективен, чем вызов ядра
- При запуске одного потока ни один другой поток не будет запущен, пока первый поток добровольно не отдаст процессор
- Проблема реализации блокирующих системных запросов.

# Реализация потоков в ядре

- Все вызовы, способные заблокировать поток, реализованы как системные
- Когда поток блокируется, ядро запускает другой поток
- Некоторые системы используют повторное использование потоков (потоки помечаются как нефункционирующие)
- Управление потоками в ядре не требует новых не блокирующих системных вызовов
- Основным недостатком управления потоками в ядре является существенная цена системных вызовов

# Создание потоков в Posix

- #include <pthread.h>
- int pthread_create(

```
    pthread_t *pThread,
    const pthread_attr_t *pAttr,
    void *(* pStart)(void *),
    void *pArg);
```

ИНСТИТУТ МАТЕМАТИКИ МЕХАНИКИ КОМПЬЮТЕРНЫХ НАУК имени И.И. Воровича

# Другие функции

- Ожидания потока

  int pthread_join(pthread_t thread, void ** pValue);

- Завершение потока

  void pthread_exit(void *retval);

- Досрочное завершение потока

  int pthread_cancel (pthread_t THREAD_ID);

ИНСТИТУТ МАТЕМАТИКИ МЕХАНИКИ КОМПЬЮТЕРНЫХ НАУК имени И.И. Воровича

# Пример

```
//thread_demo.c
#include <stdio.h>
#include <pthread.h>
void *my_th(void *arg) {
        char c;
        int  i;
        i=0;
        c=*((char *)arg);
        while (i++<1000)
                printf(" %c%d",c, i);
        return (NULL);
}
```

# Пример (окончание)

```
int main(void) {
        pthread_t th1, th2;
        char c1='A', c2='B', c3='C';
        pthread_create(&th1, NULL, my_th, (void *)&c1);
        pthread_create(&th2, NULL, my_th, (void *)&c2);
        my_th((void *)&c3);
        return(0);
}
```

ИНСТИТУТ МАТЕМАТИКИ МЕХАНИКИ КОМПЬЮТЕРНЫХ НАУК имени И.И. Воровича

# Результат

```
mmcs@lubuntu-vm:~/2018/My decisions/OS$ gcc -pthread  -o thread_demo thread_demo.c
mmcs@lubuntu-vm:~/2018/My decisions/OS$ ./thread_demo
 C1 C2 C3 C4 C5 C6 C7 C8 C9 C10 C11 C12 C13 C14 C15 C16 C17 C18 C19 C20 C21 C22 C23 C24 C25 C26 C27 C28 C
29 C30 C31 C32 C33 C34 C35 C36 C37 C38 C39 C40 C41 C42 C43 C44 C45 C46 C47 C48 C49 C50 C51 C52 C53 C54 C5
5 C56 C57 C58 C59 C60 C61 C62 C63 C64 C65 C66 C67 C68 C69 C70 C71 C72 C73 C74 C75 C76 C77 C78 C79 C80 C81
 C82 C83 C84 C85 A1 A2 A3 A4 A5 A6 A7 A8 A9 A10 A11 A12 A13 A14 A15 A16 A17 A18 A19 A20 A21 A22 A23 A24 A
25 A26 A27 A28 A29 A30 A31 A32 A33 A34 A35 A36 A37 A38 A39 A40 A41 A42 A43 A44 A45 A46 A47 A48 A49 A50 A5
1 A52 A53 A54 A55 A56 A57 A58 A59 A60 A61 A62 A63 A64 A65 A66 A67 A68 A69 A70 A71 A72 A73 A74 A75 A76 A77
 A78 A79 A80 A81 A82 A83 A84 A85 A86 A87 A88 A89 A90 A91 A92 A93 A94 A95 A96 A97 A98 A99 A100 A101 A102 A
103 A104 A105 A106 A107 A108 A109 A110 A111 A112 A113 A114 A115 A116 A117 A118 A119 A120 A121 A122 A123 A
124 A125 A126 A127 A128 A129 A130 A131 A132 A133 A134 A135 A136 A137 A138 A139 A140 A141 A142 A143 A144 A
145 A146 A147 A148 A149 A150 A151 A152 A153 A154 A155 A156 A157 A158 A159 A160 A161 A162 A163 A164 A165 A
166 A167 A168 A169 A170 A171 A172 A173 A174 A175 A176 A177 A178 A179 A180 A181 A182 A183 A184 A185 A186 A
187 A188 A189 A190 A191 A192 A193 A194 A195 A196 A197 A198 A199 A200 A201 A202 A203 A204 A205 A206 A207 A
208 A209 A210 A211 A212 A213 A214 A215 A216 A217 A218 A219 A220 A221 A222 A223 A224 A225 A226 A227 A228 A
229 A230 A231 A232 A233 A234 A235 A236 A237 A238 A239 A240 A241 A242 A243 A244 A245 A246 A247 A248 A249 A
250 A251 A252 A253 A254 A255 A256 C86 C87 C88 C89 C90 B1 A257 B2 C91 C92 B3 C93 B4 C94 B5 C95 B6 C96 B7 C
97 B8 C98 B9 C99 B10 C100 B11 C101 B12 C102 B13 C103 C104 B14 C105 B15 C106 B16 B17 C107 B18 C108 B19 C10
9 C110 B20 C111 B21 C112 B22 C113 B23 C114 B24 C115 B25 C116 B26 C117 B27 C118 B28 C119 B29 C120 B30 C121
 B31 C122 B32 C123 B33 C124 B34 C125 B35 C126 B36 C127 B37 C128 B38 C129 B39 C130 B40 C131 B41 C132 B42 C
133 B43 C134 B44 C135 B45 C136 B46 C137 B47 C138 B48 C139 B49 C140 B50 C141 B51 C142 B52 C143 B53 C144 B5
4 C145 B55 C146 B56 C147 B57 C148 B58 C149 B59 C150 B60 B61 B62 B63 B64 B65 B66 B67 B68 B69 B70 B71 B72 B
73 B74 B75 B76 B77 B78 B79 B80 B81 B82 B83 B84 B85 B86 B87 B88 B89 B90 B91 B92 B93 B94 B95 B96 B97 B98 B9
9 B100 B101 B102 B103 B104 B105 B106 B107 B108 B109 B110 B111 B112 B113 B114 B115 B116 B117 B118 B119 B12
0 B121 B122 B123 B124 B125 B126 B127 B128 B129 B130 B131 B132 B133 B134 B135 B136 B137 B138 B139 B140 B14
1 B142 B143 B144 B145 B146 B147 B148 B149 B150 B151 B152 B153 B154 B155 B156 B157 B158 B159 B160 B161 B16
2 B163 B164 B165 B166 B167 B168 B169 B170 B171 B172 B173 B174 B175 B176 B177 B178 B179 B180 B181 B182 B18
3 B184 B185 B186 B187 B188 B189 B190 B191 B192 B193 B194 B195 B196 B197 B198 B199 B200 B201 B202 B203 B20
4 B205 B206 B207 B208 B209 B210 B211 B212 B213 B214 B215 B216 B217 B218 B219 B220 B221 B222 B223 B224 B22
5 B226 B227 B228 B229 B230 B231 B232 B233 B234 B235 B236 B237 B238 B239 B240 B241 B242 B243 B244 B245 B24
6 B247 B248 B249 B250 B251 B252 B253 B254 B255 B256 B257 B258 B259 B260 B261 B262 B263 B264 B265 B266 B26
7 B268 B269 B270 B271 B272 B273 B274 B275 B2                                                B285 B286 B287 B28
8 B289 B290 B291 B292 B293 B294 B295 B296 B2                                                B306 B307 B308 B30
9 B310 B311 B312 B313 B314 B315 B316 B317 B3                                                B327 B328 B329 B33
```

переключения потоков

# Взаимодействие процессов

- *Кооперативные* или *взаимодействующие* процессы - это процессы, которые влияют на поведение друг друга путем обмена информацией

- Причины
  - Повышение скорости решения задач
  - Совместное использование данных
  - Модульная конструкция какой-либо системы
  - Для удобства работы пользователя

- Кооперация = взаимодействие процессов = межпроцессное взаимодействие = InterProcess Communication (IPC).

ИНСТИТУТ МАТЕМАТИКИ МЕХАНИКИ КОМПЬЮТЕРНЫХ НАУК имени И.И. Воровича

# Категории средств обмена информацией

- Сигнальные
- Разделяемая память
- Канальные (самые распространённые)

ИНСТИТУТ МАТЕМАТИКИ МЕХАНИКИ КОМПЬЮТЕРНЫХ НАУК имени И.И. Воровича

# Сигналы в POSIX

- Сигнал в Unix-подобных операционных системах — асинхронное уведомление процесса о каком-либо событии, один из основных способов взаимодействия между процессами.
- Таблица с перечнем сигналов в POSIX

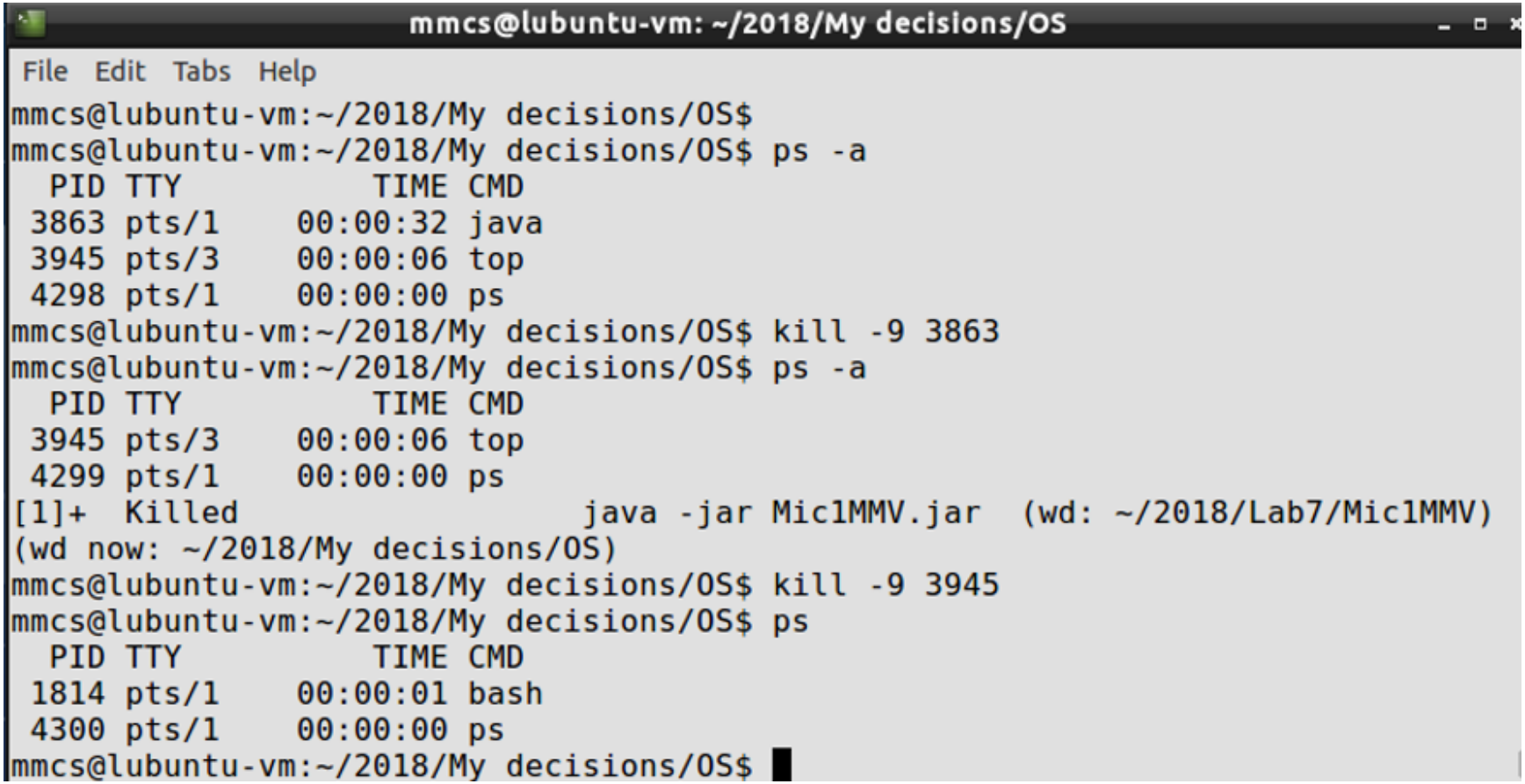
ИНСТИТУТ МАТЕМАТИКИ МЕХАНИКИ КОМПЬЮТЕРНЫХ НАУК имени И.И. Воровича

# Сигналы посылаются

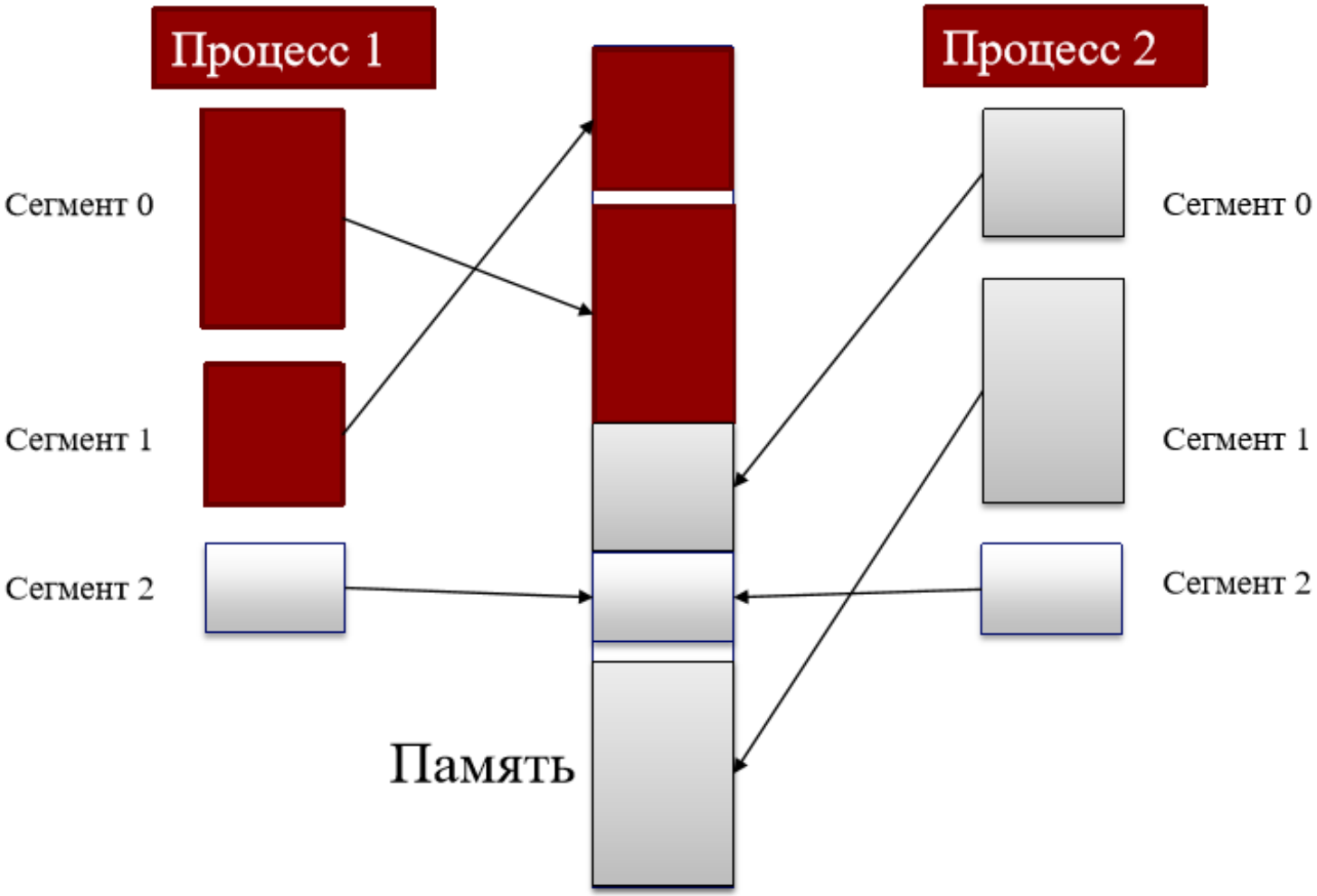
- из терминала, нажатием специальных клавиш или комбинаций (например, нажатие Ctrl-C генерирует SIGINT, Ctrl-\ SIGQUIT, а Ctrl-Z SIGTSTP);
- ядром системы:
  - при возникновении аппаратных исключений;
  - ошибочных системных вызовах;
  - для информирования о событиях ввода-вывода;
- одним процессом другому, с помощью системного вызова kill(), в том числе:
  - из shell, утилитой /bin/kill.

ИНСТИТУТ МАТЕМАТИКИ МЕХАНИКИ КОМПЬЮТЕРНЫХ НАУК имени И.И. Воровича

# Сигнал SIGKILL



# Разделяемая память (shared memory)



# Особенности работы с разделяемой памятью

- Обработка данных, находящихся в разделяемой памяти, выполняется обычными средствами языков программирования, которые используются для работы с памятью.
- Поддерживается произвольный порядок доступа к данным в режимах чтения и записи, что позволяет организовать обмен данными в двух направлениях.
- Обеспечивается наивысшая скорость обмена данными и наибольшие объемы передачи данных.
- Считается надежным средством передачи данных.
- Разделяемую память могут использовать только процессы работающие в рамках одной вычислительной системы.

# Разделяемая память POSIX

```
#include <sys/types.h>
#include <sys/shm.h>
int shmget(key_t nKey, int nSize, int nShmFlg);
```

предназначена для выполнения операции доступа к сегменту разделяемой памяти и, в случае ее успешного завершения, возвращает дескриптор для этого сегмента

| nKey | nShmFlg |
|---|---|
| IPC_PRIVATE | IPC_CREAT |
| | IPC_EXCL |

Возможные значения флагов параметров функции shmget()

ИНСТИТУТ МАТЕМАТИКИ МЕХАНИКИ КОМПЬЮТЕРНЫХ НАУК имени И.И. Воровича

# shmat(), shmdt()

```
#include <sys/types.h>
#include <sys/shm.h>
void *shmat(int nShmId, const void *pvShmAddr, int nShmFlg);
int shmdt(const void *pvShmAddr);
```

**shmat** предназначен для включения, а **shmdt** для исключения области разделяемой памяти в/из адресного пространства текущего процесса.

| nShmFlg |
| --- |
| 0 |
| SHM_RDONLY |

Возможные значения флагов параметров функции shmat()

ИНСТИТУТ МАТЕМАТИКИ МЕХАНИКИ КОМПЬЮТЕРНЫХ НАУК имени И.И. Воровича

# Решение проблемы дублирования ключей

- Использование в качестве ключа константы IPC_PRIVATE.
- Генерирование ключа при помощи функции ftok().

```
#include <sys/types.h>
#include <sys/ipc.h>

key_t ftok(const char *pcszPathName, int nProjId);
```

# Пример

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <stdio.h>
#include <errno.h>
int main(){
    int *array;
    int shmid;
    int new = 1;
    key_t key
    if((key = ftok("06-1a.c",0)) < 0)
        { printf("Can\'t generate key\n"); exit(-1); }
```

# Пример (продолжение)

```
if((shmid = shmget(key, 3*sizeof(int),
0666|IPC_CREAT|IPC_EXCL)) < 0){
    if(errno != EEXIST){
        printf("Can\'t create shared memory\n"); exit(-1); }
    else {
        if((shmid = shmget(key, 3*sizeof(int), 0)) < 0){
            printf("Can\'t find shared memory\n"); exit(-1); }
        new = 0; }
        }
```

# Пример (окончание)

```
if((array = (int *)shmat(shmid, NULL, 0)) == (int *)(-1)){
    printf("Can't attach shared memory\n"); exit(-1); }
if(new){
    array[0] = 1; array[1] = 0; array[2] = 1; }
else {
    array[0] += 1; array[2] += 1; }
printf("%d  %d  %d \n", array[0], array[1], array[2]);
if(shmdt(array) < 0){
    printf("Can't detach shared memory\n"); exit(-1); }
return 0; }
```

# Виды каналов

- Неименованные
  - используются для организации передачи данных между родительскими, дочерними процессами и их потомками
- Именованные
  - при создании присваивается имя, которое доступно для других процессов

ИНСТИТУТ МАТЕМАТИКИ МЕХАНИКИ КОМПЬЮТЕРНЫХ НАУК имени И.И. Воровича

# Использование каналов в POSIX

- ls | sort | lp


- Когда процесс создает канал, ядро устанавливает два файловых дескриптора для пользования этим каналом.
- Один дескриптор используется, чтобы открыть путь ввода в канал (запись – fd1), другой применяется для получения данных из канала (чтение – fd0).

ИНСТИТУТ МАТЕМАТИКИ МЕХАНИКИ КОМПЬЮТЕРНЫХ НАУК имени И.И. Воровича

# Использование каналов в POSIX


- Два процесса взаимно согласовываются и "закрывают" неиспользуемый конец канала.


- Чтобы получить прямой доступ к каналу, можно применять системные вызовы, подобные тем, которые нужны для ввода/вывода в файл или из файла на низком уровне

# Закрытие канала

- Оставлять оба дескриптора незакрытыми плохо по двум причинам:
  - Родитель после записи может считать часть собственных данных, которые станут недоступными для потомка.
  - Если один из процессов завершился или закрыл свои дескрипторы, то второй этого не заметит, так как pipe на его стороне по-прежнему открыт на чтение и на запись.
- Если надо организовать двунаправленную передачу данных, то можно создать два канала.

# Неименованные каналы (pipe)

```
#include <unistd.h>
int pipe(int FD[2]);            // FD[0] – чтение,
                                // FD[1] – запись

int close(int FD);

ssize_t read(int FD, void *pvBuf, size_t uCount);
ssize_t write(int FD, const void *pcvBuf, size_t uCount);
int eof(int FD);
```

ИНСТИТУТ МАТЕМАТИКИ МЕХАНИКИ КОМПЬЮТЕРНЫХ НАУК имени И.И. Воровича

# Пример 1

```
#include <sys/types.h>
#include <unistd.h>
#include <stdio.h>
int main(){
   int fd[2];
   size_t size;
   char string[] = "Hello, world!";
   char resstring[14];

   if(pipe(fd) < 0){     printf("Can\'t create pipe\n");       exit(-1); }
   size = write(fd[1], string, 14);     if(size != 14){     printf("Can\'t write all string\n");   exit(-1); }
   size = read(fd[0], resstring, 14);   if(size < 0){       printf("Can\'t read string\n");        exit(-1); }

   printf("%s\n",resstring);

   if(close(fd[0]) < 0){  printf("Can\'t close input stream\n");  }
   if(close(fd[1]) < 0){  printf("Can\'t close output stream\n"); }
   return 0;
}
```

ИНСТИТУТ МАТЕМАТИКИ МЕХАНИКИ КОМПЬЮТЕРНЫХ НАУК имени И.И. Воровича

# Пример 2

```
#include <sys/types.h>
#include <unistd.h>
#include <stdio.h>
int main(){
    int fd[2], result;
    size_t size;
    char resstring[14];

    if(pipe(fd) < 0){ printf("Can\'t create pipe\n");  exit(-1); }

    result = fork();

    if(result < 0){ printf("Can\'t fork child\n"); exit(-1);  }
    else if (result > 0) {
        close(fd[0]);
        size = write(fd[1], "Hello, world!", 14);
```

# Пример 2 (окончание)

```
        if(size != 14){   printf("Can\'t write all string\n"); exit(-1); }
        close(fd[1]);
        printf("Parent exit\n");
    }
    else {       close(fd[1]);
                 size = read(fd[0], resstring, 14);
                 if(size < 0){   printf("Can\'t read string\n"); exit(-1); }

                 printf("%s\n",resstring);

                 close(fd[0]);
    }
    return 0;
}
```

# Именованные каналы (FIFO)

```
#include <sys/types.h>
#include <sys/stat.h>
int mkfifo(const char *pcszPathName, mode_t ulMode);
```

# Особенности

- Именованные каналы существуют в виде специального файла устройства в файловой системе.
- Процессы различного происхождения могут разделять данные через такой канал.
- Именованный канал остается в файловой системе для дальнейшего использования и после того, как весь ввод/вывод сделан.

ИНСТИТУТ МАТЕМАТИКИ МЕХАНИКИ КОМПЬЮТЕРНЫХ НАУК имени И.И. Воровича

# Пример fifo_server.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define PIPE_NAME_1 "to-server"
#define PIPE_NAME_2 "from-server"
#define RESPONSE_ST "Response from server"
#define BUFFER_SIZE 50
int main(){
        int nResult, nFD1, nFD2;
        ssize_t i, nLength;
        char achBuffer[BUFFER_SIZE];
        const char *pcszResponse = RESPONSE_ST;

        printf("Starting server...\n");
        if (unlink(PIPE_NAME_1) != 0){perror("Remove pipe 1");exit(-1);}

        if (unlink(PIPE_NAME_2) != 0){perror("Remove pipe 2");exit(-1);}
```

# Пример (продолжение)

```
nResult = mkfifo(PIPE_NAME_1, S_IWUSR | S_IRUSR);
if (nResult != 0){perror("Create pipe 1");exit(-1);}

nResult = mkfifo(PIPE_NAME_2, S_IWUSR | S_IRUSR);
if (nResult != 0){perror("Create pipe 2");exit(-1);}

printf("Opening...\n");
nFD1 = open(PIPE_NAME_1, O_WRONLY);
if (nFD1 < 0){perror("Open pipe 1");exit(-1);}

nFD2 = open(PIPE_NAME_2, O_RDONLY);
if (nFD2 < 0){perror("Open pipe 2");exit(-1);}
printf("Opened!\n");
```

# fifo_server.c (окончание)

```
do{     nLength = read(nFD2, achBuffer, BUFFER_SIZE);
        if (nLength < 0){perror("read");exit(-1);}
        for (i = 0; i < nLength; ++ i)
                putchar(achBuffer[i]);
}while (nLength == BUFFER_SIZE);
putchar('\n');

printf("Writing server...\n");
write(nFD1, pcszResponse, strlen(pcszResponse));
printf("Wrote server!\n");
close(nFD1);
close(nFD2);}
```

# Пример fifo_client.c

```
#include <sys/stat.h>
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define PIPE_NAME_1 "to-server"
#define PIPE_NAME_2 "from-server"
#define RESPONSE_ST "Response from client"
#define BUFFER_SIZE 50
int main(){
        int nResult, nFD1, nFD2;
        ssize_t i, nLength;
        char achBuffer[BUFFER_SIZE];
        const char *pcszResponse = RESPONSE_ST;

        printf("Starting client...\n");
        printf("Opening...\n");
        nFD2 = open(PIPE_NAME_1, O_RDONLY);
        if (nFD2 < 0){perror("Open pipe 2");exit(-1);}
```

# fifo_client.c (окончание)

```
nFD1 = open(PIPE_NAME_2, O_WRONLY);
if (nFD1 < 0){perror("Open pipe 1");exit(-1);}
printf("Opened!\n");
printf("Writing client...\n");
write(nFD1, pcszResponse, strlen(pcszResponse));
printf("Wrote client!\n");
do{
        nLength = read(nFD2, achBuffer, BUFFER_SIZE);
        if (nLength < 0){perror("read");exit(-1);}
        for (i = 0; i < nLength; ++ i)
                putchar(achBuffer[i]);
}while (nLength == BUFFER_SIZE);
putchar('\n');
close(nFD1);
close(nFD2);}
```

ИНСТИТУТ МАТЕМАТИКИ МЕХАНИКИ КОМПЬЮТЕРНЫХ НАУК имени И.И. Воровича

# Домашнее задание

- Подготовка к тстированию по материалам лекций.
- Читать книгу Таненбаум Э., Бос Х. Современные операционные системы, стр.80-82, стр. 111-123. стр. 178-198.
- Подготовка к лабораторной работе 11
- Рекомендованные курсы на платформе Intuit:
  - https://intuit.ru/studies/courses/1088/322/info
  - https://intuit.ru/studies/courses/2249/52/info