МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ
РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ
БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
«САМАРСКИЙ ГОСУДАРСТВЕННЫЙ АЭРОКОСМИЧЕСКИЙ
УНИВЕРСИТЕТ ИМЕНИ АКАДЕМИКА С.П. КОРОЛЕВА
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)» (СГАУ)

Архитектура современных операционных систем

Электронный учебно-методический комплекс
по дисциплине в LMS Moodle

Работа выполнена по мероприятию блока 1 «Совершенствование
образовательной деятельности» Программы развития СГАУ
на 2009 – 2018 годы по проекту «Разработка образовательных программ со
сквозной документацией процессов по информатике, математике и физике.»
Соглашение № 1/3 от 03.06.2013 г.

САМАРА
2013

УДК 004.451
А 878

Автор-составитель: **Климентьев Константин Евгеньевич**

Архитектура современных операционных систем [Электронный ресурс] : электрон. учеб.-метод. комплекс по дисциплине в LMS Moodle / Мин-во образования и науки РФ, Самар. гос. аэрокосм. ун-т им. С. П. Королева (нац. исслед. ун-т); авт.-сост. К.Е. Клименетьев - Электрон. текстовые и граф. дан. - Самара, 2013. – 1 эл. опт. диск (CD-ROM).

В состав учебно-методического комплекса входят:

1. АСОС.Курс лекций.Часть 1.pdf
2. АСОС.Курс лекций.Часть 2.pdf
3. АСОС.Курс лекций.Часть 3.pdf
4. АСОС.Курс лекций.Часть 4.pdf
5. АСОС.Методические указания и задания к лабораторной работе 1.pdf
6. АСОС.Методические указания и задания к лабораторной работе 2.pdf
7. АСОС.Тесты к зачету.pdf
8. Рабочая программа - Архитектура современных операционных систем.pdf

УМКД «Архитектура современных операционных систем» предназначен для студентов факультета информатики, обучающихся по направлению подготовки магистров 230100.68 «Информатика и вычислительная техника».

УМКД разработан на кафедре Информационных систем и технологий.

Самарский аэрокосмический университет им. акад. С.П. Королева

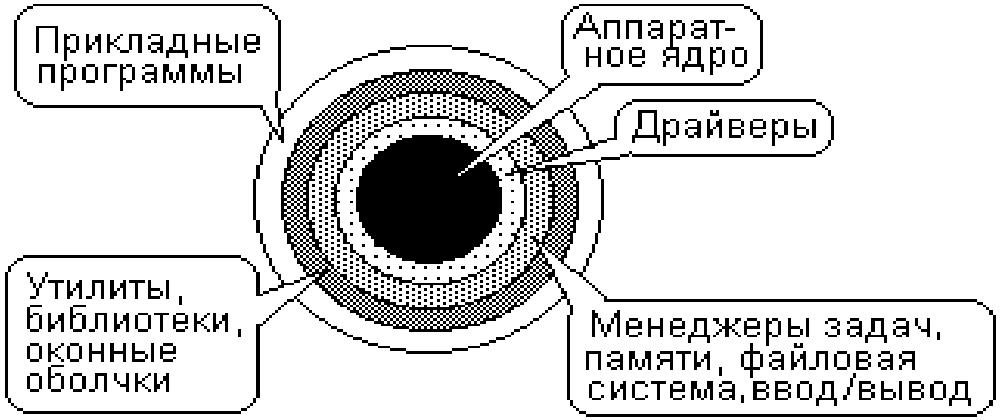
Архитектура современных ОС
Часть I

Составитель: к.т.н., доц. К.Е. Климентьев

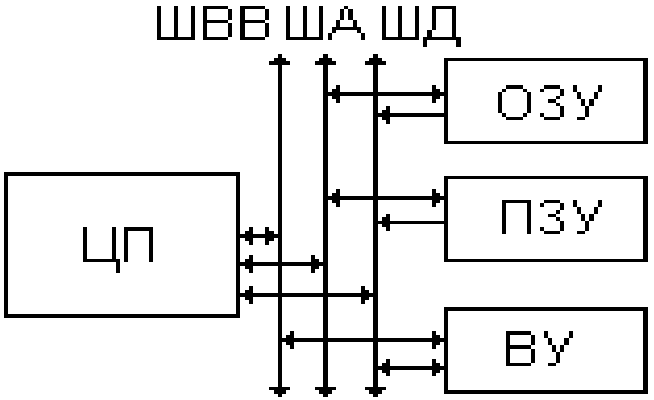
Самара 2013

1. Обобщенная модель вычислительного устройства

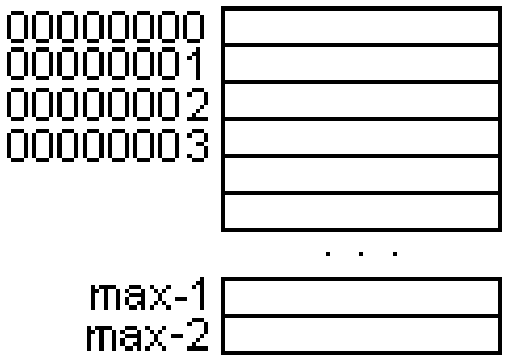
1.1. Общая схема



1.2. Архитектура вычислительной системы



1.3. Схема адресного пространства



2. Процессоры Intel

2.1. Хронология

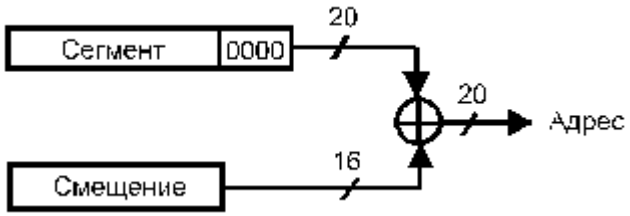
2.1.1. 16-битовые процессоры
• 8086, 8088 – 1978-79 гг., 1 Мб адресного пространства, 5-10 МГц
• 80186 – 1982 г., использовался в контроллерах
• 80286 – 1982 г., до 20 МГц, сегментный защищенный режим

2.1.2. 32-битовые процессоры
• 80386 – 1985 г., до 40 МГц, 2^{32}=4 Гб адресного пространства, страничный защищенный режим
• 80486 – 1994 г., до 75 МГц
• Pentium/Pentium Pro – 1993-95 гг., до 200 МГц
• Pentium MMX – 1997 г., до 300 МГц,, «векторные» и «матричные» команды
• Pentium II – 1997 г., до 533 МГц
• Pentium III – 1999-2001 гг.., до 1400 МГц
• Pentium IV – 2001-2002, до 3400 МГц
• Intel Core Solo, Duo, Quattro – 2006 г., 1, 2 или 4 ядра, до 2600 МГц

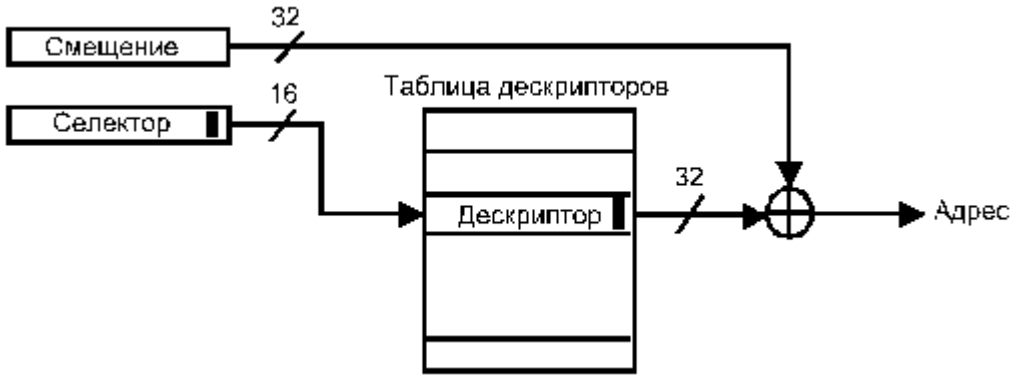
2.1.3. 64-битовые процессоры
• Itanium – 2001-2007 гг. г., до 1666 МГц,, 2^{64} адресов (точнее, 2^{48})
• Xeon – 2006 г., до 3000 МГц
• Intel Core 2 Solo, Duo, Quattro – 2006 г., 1, 2 или 4 ядра, до 1600 МГц
• Intel Core i3, i5, i7 – до 3300 МГц, интегрированный графический процессор

3. Процессоры Intel – режимы работы

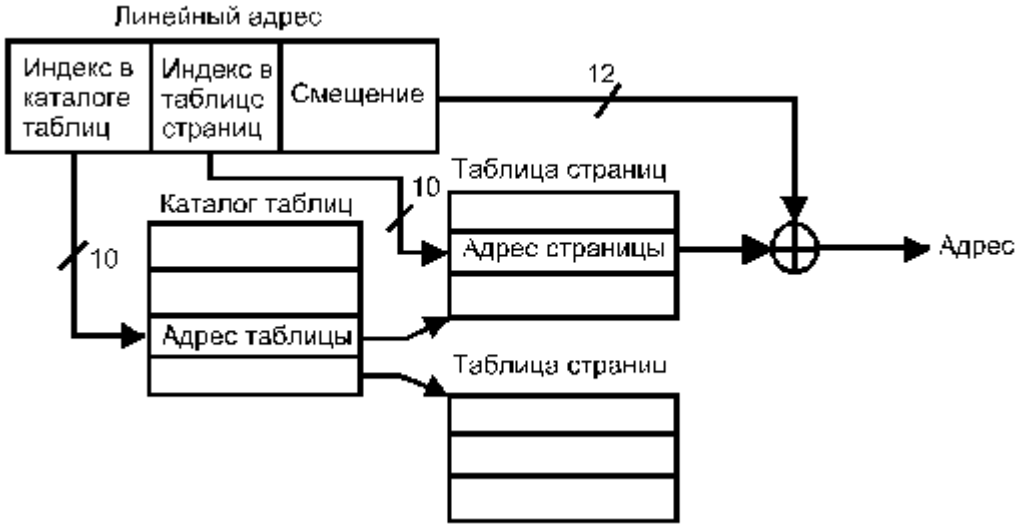
3.1. Реальный режим



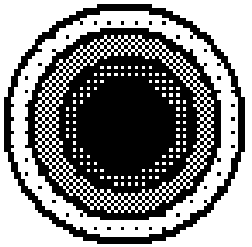
3.2. Сегментный защищенный режим



3.3. Страничный защищенный режим



3.4. Кольца защиты



Уровни привилегий:
- CPL – Current privelege level
- RPL – Requested privelege level
- DPL – Descriptor privelege level

Правило: CPL ≤RPL ≤ DPL

4. Процессоры ARM

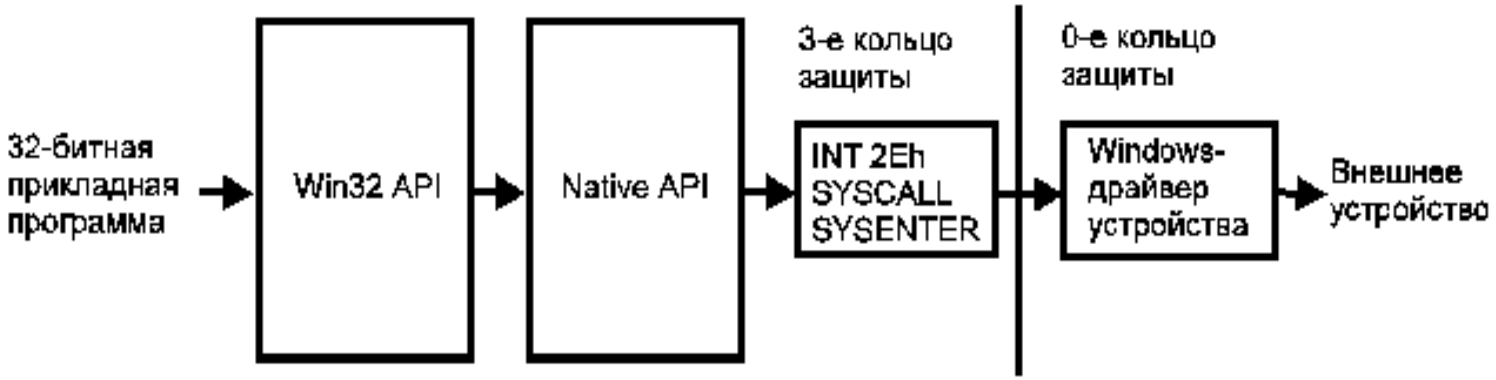
4.1. Режимы работы:
• User – режим выполнения прикладных программ;
• System – режим выполнения операционной системы;
• FIQ, IRQ, Abort, Undef – режимы обработки прерываний и исключений;
• Supervisor – при включении питания.

4.2. Регистры:
•r0-r12 – регистры общего назначения;
• r13 – указатель стека;
• r14 – регистр ссылк (link register);
• r15 – счетчик команд;
• cpsr – регистр статуса (флагов).

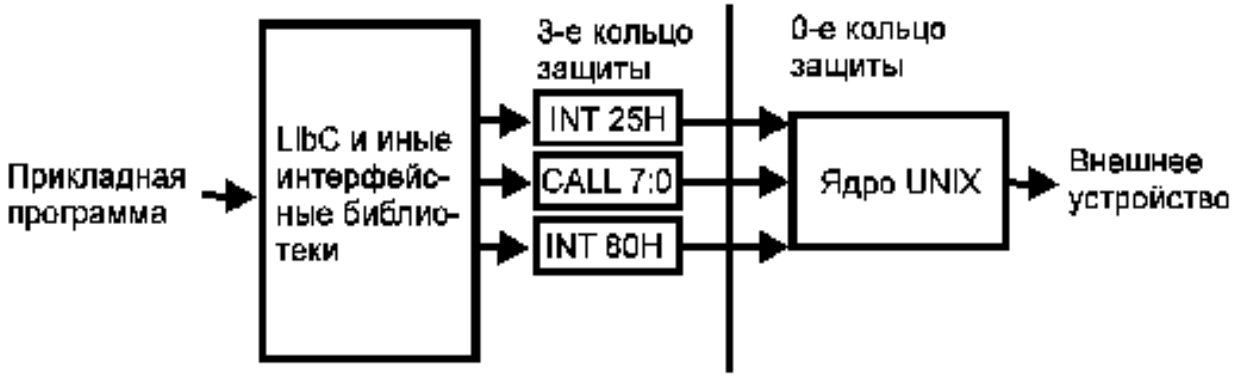
4.3. Наборы команд:
• ARM – 32 бита, пример: **ADD r0, r1, r2** – трехместная!!!;
• Thumb – 16 битов;
• Jazelle – 8 битов.

5. Архитектуры офисных ОС

6.1. Архитектура Windows

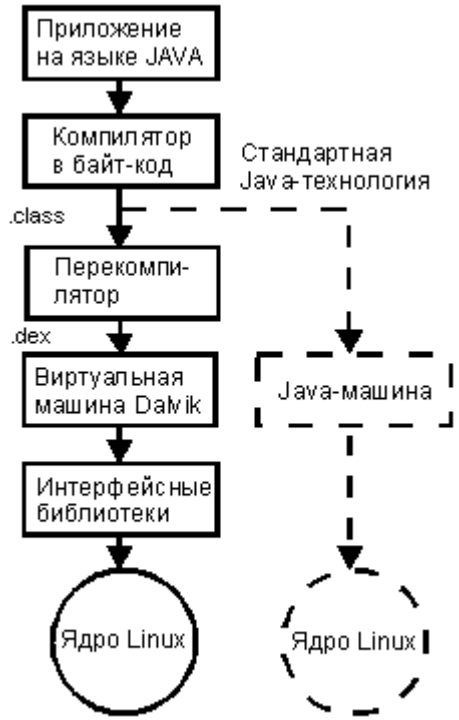


6.2. Архитектура UNIX

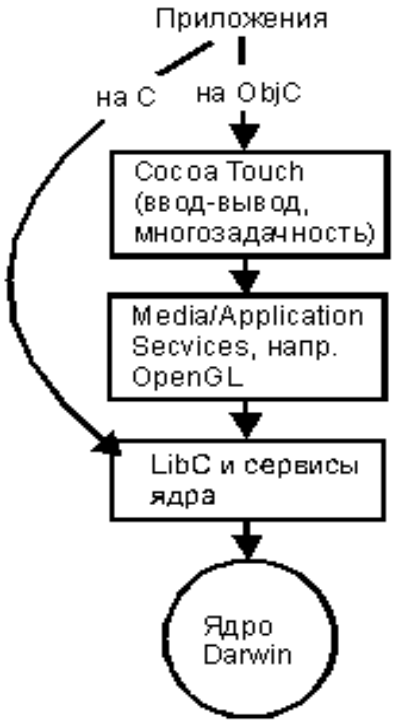


6. Архитектуры мобильных ОС

6.1. Архитектура Android



6.2. Архитектура iOS



Самарский аэрокосмический университет им. акад. С.П. Королева

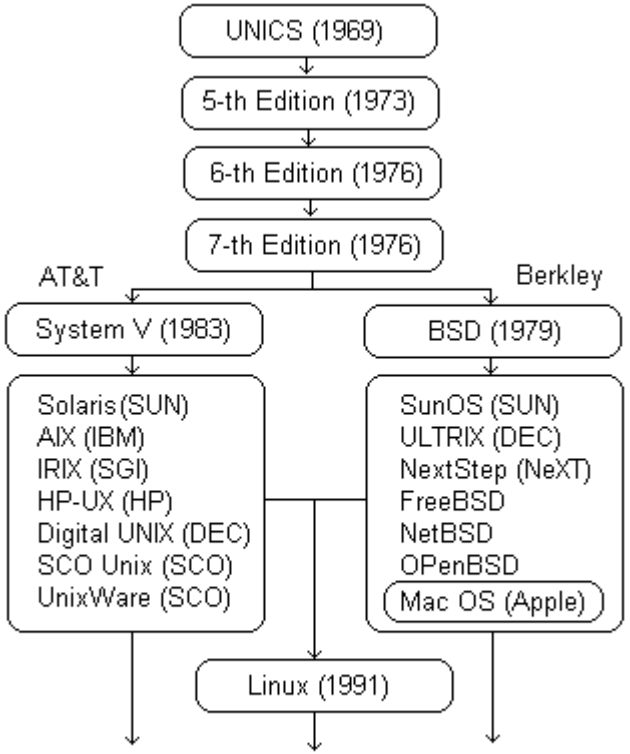
Архитектура современных ОС
Часть II

Составитель: к.т.н., доц. К.Е. Климентьев

Самара 2013

1. История UNIX и предпосылки POSIX

1.1. Основные линии развития UNIX



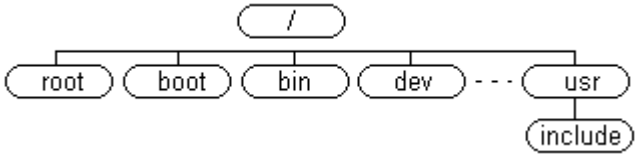
1.2. Упрощенная архитектура ОС

1.3. Составляющие элементы стандартизации:

- ANSI C – стандартизация языка;
- X Window – стандартизация графики;
- SVID и POSIX – стандартизация программного и пользовательского интерфейса.

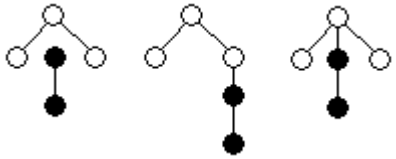
2. Общая характеристика UNIX

2.5.1. Дерево каталогов

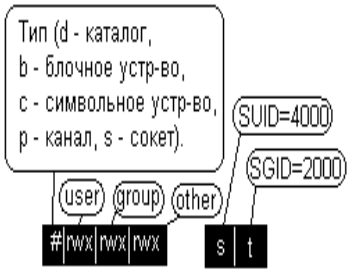


/root – домашний каталог суперпользователя
/boot – файлы ядра, загрузчика и пр.
/home – домашние подкаталоги пользователей
/bin – программы и утилиты
/sbin – системные утилиты
/usr – приложения, исходники и т.п.
/opt – большие приложения
/dev – файлы устройств
/etc – конфигурационные файлы
/lib – библиотеки
/mnt – сюда монтировать съемные устройства
/tmp – временные файлы

2.5.2. Монтируемость файловых систем



2.5.3. Биты доступа



Примеры:

- ls –l *.* – посмотреть биты;
- chmod –R 755 *.* – рекурсивная установка битов
- chmod 444 myfile.txt – индивидуальная установка.

3. Общая характеристика POSIX

POSIX (Portable Operating System Interface for Unix) – набор стандартов на программный интерфейс между ядром ОС и прикладными программами.

Официальное наименование: ISO/IEC 9945.
Разработчик: комитет 1003 IEEE.
Дата 1-ой публикации: 1988 г.
Дата последнего изменения: 2008 г.

Содержимое:
- имена интерфейсных функций;
- имена типов данных;
- имена и состав заголовочных файлов;
- имена глобальных переменных;
- символьные имена кодов ошибок (но не числовые значения!);
- имена стандартных констант;
- символьные имена номеров сигналов (но не числовые значения!);
- символьные имена аргументов функций (но не числовые значения!);
- имена макросов, констант, битовых флагов и переменных среды окружения.

Состав:
- POSIX.1 – сервисы ядра;
- POSIX.1b – расширения реального времени;
- POSIX.1c – расширенные возможности потоков;
- POSIX.2 – Shell, команды и утилиты.

4. Понятие «многозадачности»

Многозадачность – параллельная работа нескольких программ (задач, вычислительных процессов):
- «истинная», когда процессоров не меньше, чем программ;
- «псевдопараллельная», когда процессоров меньше, чем программ.

Предпосылки многозадачности:
- разделение времени между пользователями;
- работа в реальном времени со внешними объектами в темпе процессов, протекающих в объектах;
- повышение общей производительности вычислительной системы.

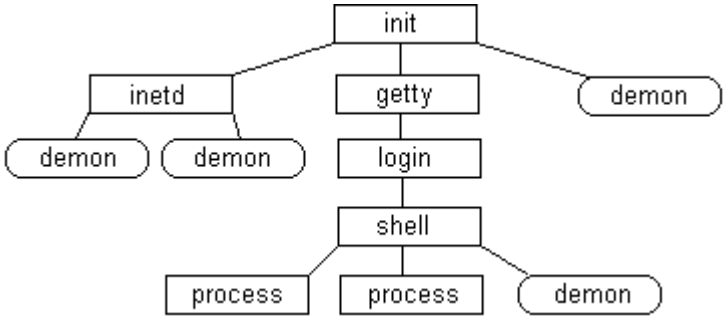
Ресурсы, необходимые для организации виртуальной сессии:
1) терминал (возможно, виртуальный);
2) набор задач (программ) пользователя;
3) задача (программа), осуществляющая диалог – shell.

Типы задач:
- процессы – программы, монопольно владеющие выделенным адресным пространством;
- потоки – программы, разделяющие общее адресное пространство (т.е. код, данные, стек и т.п.).

Контекст задачи – «личность» задачи (содержимое регистров процессора, положение стека, адрес выполняемой команды, запись в системных таблицах операционной системы и т.п.).

5. Процессы

Процесс – исторически первый и основной тип задачи в Unix.



Идентифицирующая информация:

- id процесса PID;
- id родителя PPID;
- id хозяина UID;
- id группы хозяина GUID.

Способы запуска процессов:

- «обычный»: `$proc`
- «конвейер»: `$proc1|proc2|proc3…`
- Список параллельных процессов: `$proc1&proc2&proc3&…`
- Список последовательных процессов: `$proc1;proc2;proc3`

6. Процессы (продолжение)

Просмотр информации о запущенных процессах: команда ps (ключ -e - показать всех).

```
root@slax:#ps
  PID TTY          TIME CMD
 4076 pts/0    00:00:00 bash
 4543 pts/0    00:00:00 ps
```

Завершение запущенных процессов: команда kill, примеры:

- kill SIGTERM 1234 или просто kill 1234 – посылает сигнал SIGTERM процессу с PID=1234 ;
- kill -s SIGKILL 1234 – посылает указанный сигнал процессу с PID=1234.

```
# ps
  PID TTY          TIME CMD
 4074 tty1     00:00:00 bash
 4309 tty1     00:00:00 ps
```
Текущий список процессов

```
# ./dull&
```
Запуск процесса dull в фоне

```
[1] 4310
```
Сообщение о запуске

```
# ps
  PID TTY          TIME CMD
 4074 tty1     00:00:00 bash
 4310 tty1     00:00:07 dull
 4311 tty1     00:00:00 ps
```
Список процессов с dull

```
# kill 4310
```
Посылка сигнала завершения

```
# ps
  PID TTY          TIME CMD
 4074 tty1     00:00:00 bash
 4312 tty1     00:00:00 ps
```
Текущий список процессов без dull

```
[1]+  Terminated ./dull
```
Сообщение о завершении

7. Процессы (продолжение)

Сигнал – сообщение, посылаемое процессу. Процессу доступен факт прихода сигнала и номер сигнала.

Согласно POSIX.1, их 28; в ANSI C определены только 6:

- SIGABRT – процесс посылает сам себе для аварийного останова при помощи abort();
- SIGFPE – признак некорректной арифметической операции;
- SIGILL – признак некорректной машинной команды;
- SIGINT – сигнал для остановки процесса пользователем с терминала (Ctrl-C);
- SIGSEGV- признак общей ошибки защиты;
- SIGTERM – запрос на завершение процесса (обычно, 15).

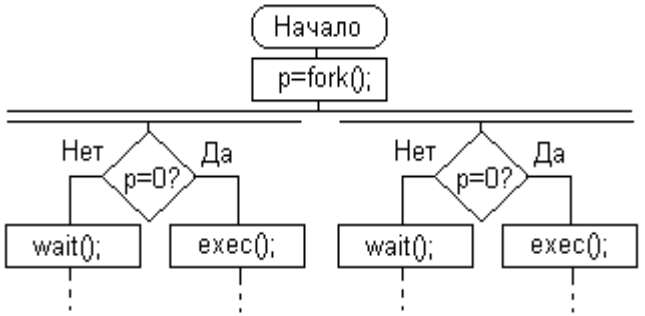
Некоторые «остальные» сигналы:

- SIGKILL – немедленного прекращения процесса (обычно, 9);
- SIGSTP и SIGCONT – приостановка (Ctrl-Z) и продолжение процесса;
- SIGSTOP – не игнорируемая версия SIGSTP;
- SIGRTMIN и SIGRTMAX – диапазон «пользовательских» сигналов.

8. Программная работа с процессами

Системные вызовы (обращения к ядру) возвращают:

- либо признак удачи/неудачи: 0/-1;
- либо код ошибки, который можно посмотреть в глобальной переменной errno.




```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>

pid_t p, q; char *e[]={"",""};

main() {
 p = fork(); /*Копирование адресного пространства и процесса*/
 if (p)      /* Родитель получает PID  ребенка*/
   wait(&q); /* Ждать завершения потомков */
 else        /* Ребенок получает 0 */
   execv("./hello", e); /* Ребенок замещает себя другой программой*/
}
```

Вновь созданный процесс наследует терминал родителя, в нем автоматически открываются дескрипторы ввода-вывода: stdin=0, stdout=1, stderr=2.

«Зомби» - процесс, который завершается раньше, чем родитель вызвал wait().

9. Программная работа с процессами (продолжение)

Программное удаление процесса: функция kill().

```
#include <stdio.h>
#include <unistd.h>
#include <signal.h>
#include <sys/types.h>
pid_t p, q; char *e[]={"",""};
main() {
 p = fork(); /*Копирование адресного пространства и процесса*/
 if (p) {    /* Родитель получает PID  ребенка*/
  sleep(10); /* Задержка на 10 сек */
  kill (p, SIGKILL);
  }
 else        /* Ребенок получает 0 */
   execv("./dull", e); /* Ребенок замещает себя другой программой*/
}
```

Структура «демона» – процесса, не привязанного к терминалу и к/л сессии.

```
 .  .  .
 setsid();      /* Перестать быть членом к/л группы ???*/
 chdir("/");   /* Текущий каталог /root */
 fclose(stdin);
 fclose(stdout);
 fclose(stderr);
 .  .  .
```

Как его запускать:

```
 .  .  .
 p = fork();
 if (p) exit(0); else execv("./demon", e);
 .  .  .
```

Самарский аэрокосмический университет им. акад. С.П. Королева

Архитектура современных ОС
Часть III

Составитель: к.т.н., доц. К.Е. Климентьев

Самара 2013

1. Потоки. Моделирование средствами процессов

Потоки (нити, треды) – программы, разделяющие общее адресное пространство (т.е. код, данные, стек и т.п.). Достоинства: быстрота переключения, простота передачи данных.

Функция **clone()** создает точную копию текущего процесса, включая:
- CLONE_VM – адресное пространство;
- CLONE_FS – файловую систему;
- CLONE_FILES – дескрипторы файлов;
- CLONE_SIGHAND – обработчики сигналов;
- CLONE_PID – идентификатор процесса (**этого делать не надо!!!**).

```
#include <sched.h>
#include <pthread.h>
#include <stdio.h>
#include <unistd.h>

/*Глобальные данные*/
char stack[10000], i, j;

/*Код и данные потока*/
int mythread(void * thread_arg) {
 for(int i=0;i<10;i++){ sleep(1);  printf("\nMythread: i=%d",i);}
 printf("Thread1 finished\n");
}
/*Код и данные «главного» потока*/
int main() {
 clone(mythread,(void*)(stack+10000-1),CLONE_VM|CLONE_FS|CLONE_FILES|LONE_SIGHAND,NULL);
 for(j=0;j<10;j++){  sleep(1);  printf("Main thread: j=%d\n",j); }
}
```

2. POSIX Threads - фреймворк pthreads

Основные функции:

- pthread_create() – создать поток;
- pthread_exit() – завершить работу потока изнутри;
- pthread_cancel() – завершить работу потока снаружи;
- pthread_kill() – послать сигнал потоку;
- pthread_join() – ждать завершения потока;
- pthread_detach() – отказаться от сохранения служебной информации о потоке.

```
/*Пример с общим адресным пространством и кодом, компиляция: gcc primer.c -lpthreads */
#include <stdlib.h>
#include <stdio.h>
#include <errno.h>
#include <pthread.h>

void *func(void *arg) {
 int loc_id = * (int *) arg;
 while (1) {
  printf("Thread %i is running\n", loc_id); sleep(1);
 }
}

int main() {
 int id1, id2, result;
 pthread_t thread1, thread2;
 id1 = 1; result = pthread_create(&thread1, NULL, func, &id1); pthread_detach(thread1);
 id2 = 2; result = pthread_create(&thread2, NULL, func, &id2); pthread_detach(thread2);
}
```

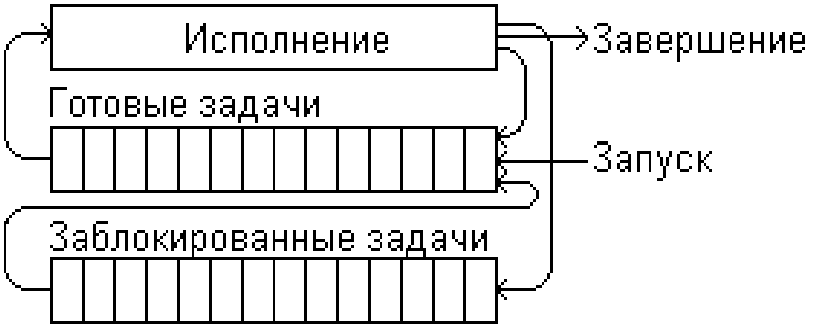
3. POSIX Threads (продолжение)

```
/*Пример с разным кодом*/
#include <stdlib.h>
#include <stdio.h>
#include <errno.h>
#include <pthread.h>

void *func1() { /* Поток 1 */
 int i; for (i=0;i<10;i++) { printf("Thread 1 is running\n"); sleep(1); }
}

void *func2() { /* Поток 2 */
 int i; for (i=0;i<10;i++) { printf("Thread 2 is running\n"); sleep(1); }
}

int main() {
 int result, status1, status2;
 pthread_t thread1, thread2;
 result = pthread_create(&thread1, NULL, func, NULL);
 result = pthread_create(&thread2, NULL, func, NULL);
 pthread_join(thread1, &status1);
 pthread_join(thread2, &status2);
 printf(”\nПотоки завершены с %d и %d”, status1, status2); // Например, PTHREAD_CANSELLED
}
```

Статусы потоков сохраняются после завершения. Если выполнить pthread_detach(), то они будут недоступны.

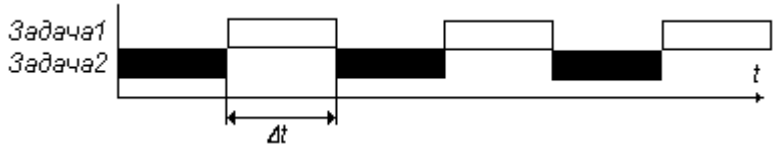
4. Организация параллельного выполнения задач

4.1. Граф многозадачности

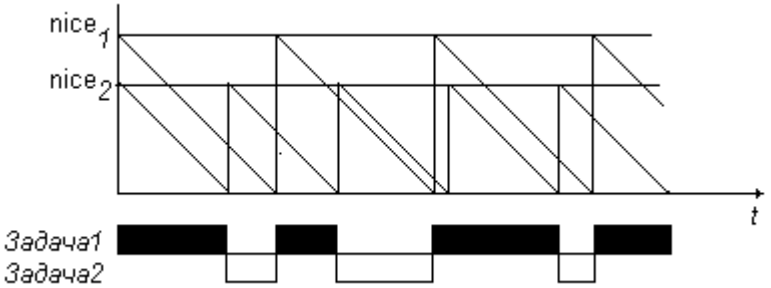


4.2. Алгоритмы диспетчеризации в UNIX

4.2.1. SCHED_RR (карусель) и SCHED_FIFO (очередь) – в pthreads не реализованы.

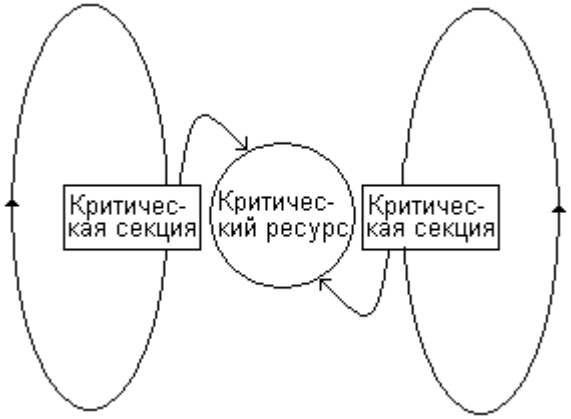


4.2.2. SCHED_OTHER (старение

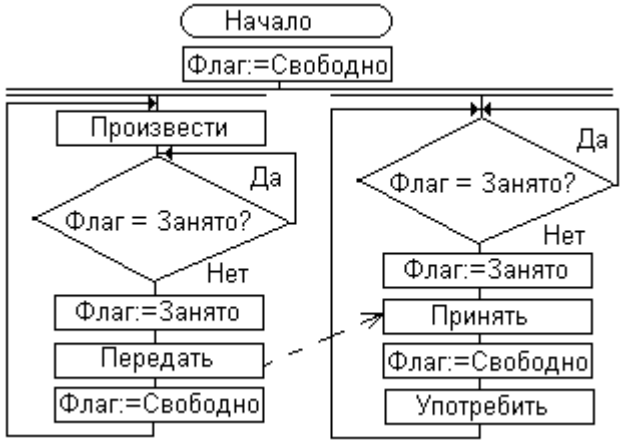


5. Проблема синхронизации задач

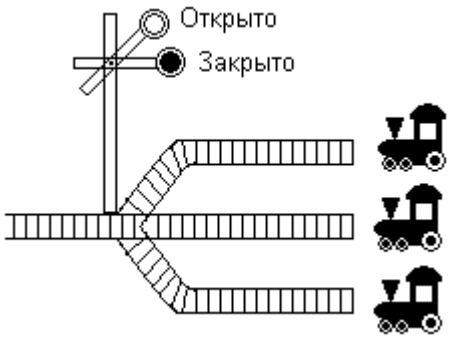
5.1. Проблема доступа к общему ресурсу



5.2. Решение при помощи блокирующего флага



5.3. Принцип работы и определение семафора

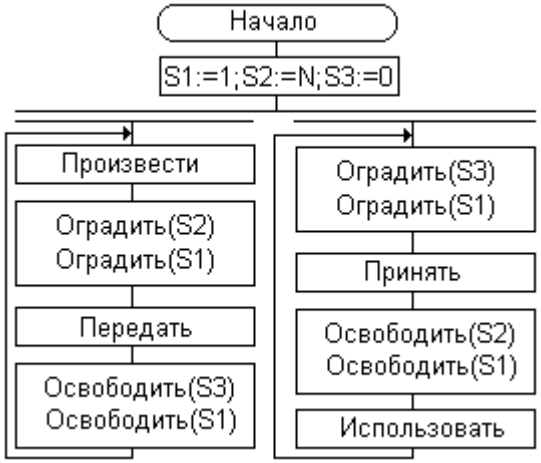


1. Целая переменная S

2. P(S) - операция «Оградить»
   - S:=S-1
   - Если S<0, то текущая задача встает в очередь

3. V(S) – операция «Освободить»
   - S:=S+1
   - Если S>0, то 1-я в очереди задача продолжает работу

5.4. Решение при помощи семафоров



6. Семафоры

Основные функции:

- sem_open() – открыть именованный семафор
- sem_close() – закрыть именованный семафор
- sem_unlink() – удалить именованный семафор
- sem_init() – инициализировать неименованный семафор
- sem_destroy() – удалить неименованный семафор
- sem_wait() – если S>0 то S:=S-1 и продолжить; иначе встать в очередь.
- sem_trywait() – возвращается с неудачей, если >0, то S:=S-1
- sem_post() – если очередь пуста, то S:=S+1 и продолжить; иначе первый в очереди продолжает работу

```
sem_t empty,full;
int data;

int main(){
  sem_init(&empty,SHARED,1); // Установлен в 1
  sem_init(&full,SHARED,0); // Установлен в 0
  ...
}
void *Producer(void *arg){ // Сначала empty=1 full=0
  while(1)  {
    sem_wait(&empty);    // empty:=empty-1
    data=Expression();
    sem_post(&full);    // full:=full+1
  }
}
void *Consumer(void *arg){ // Сначала empty=1 full=0
while(1)  {
    sem_wait(&full);    // full:=full-1
    Use(data);
    sem_post(&empty);    // empty:=empty+1
  }
 }
```

7. Мьютексы (мутексы)

Мьютекс – двоичный семафор, принимающий значения S=0 или S=1. Основные функции:

- init_mutex_init() – инициализирует мьютекс
- pthread_mutex_lock() – попытка захвата мьютекса (при неудаче ждать)
- pthread_mutex_trylock()– попытка захвата мьютекса (при неудаче вернуть ошибку)
- pthread_mutex_unlock() – освобождение мьютекса
- pthread_mutex_destroy() – удаляет мьютекс

```
int data; pthread_mutex_t mutex;

void *Producer() {
  while(1)  {
   pthread_mutex_lock(&mutex);
   printf("\nPing: %d", ++data); sleep(1);
   pthread_mutex_unlock(&mutex);
   }
  }
 void *Consumer() {
 while(1)  {
  pthread_mutex_lock(&mutex);
  printf("\nPong: %d", --data);  sleep(1);
  pthread_mutex_unlock(&mutex);
 }
}
int main() {
  pthread_t thread1, thread2;
  pthread_mutex_init(&mutex, NULL);  data = 0;
  pthread_create(&thread1, NULL, &Producer, NULL);
  pthread_create(&thread2, NULL, &Consumer, NULL);
  sleep(60);
  pthread_cancel(thread1); pthread_cancel(thread2);
 }
```

8. Условные переменные

Условные переменные – расширение мьютексов. Основные функции:

- pthread_cond_init(&cond, NULL) - инициализация
- pthread_cond_wait(&cond,&mutex) - разблокировать mutex + ждать + дождаться + заблокировать mutex
- pthread_cond_signal(&cond) - послать сигнал разблокировки первому в очереди
- pthread_cond_broadcast(&cond) - послать сигнал всем

```
int data;
pthread_mutex_t mutex;
pthread_cond_t cond;

void *Producer() {
 while(1)  {
  pthread_mutex_lock(&mutex);
  printf("\nPing: %d", ++data);
  pthread_cond_signal(&cond);
  sleep(1);
  pthread_mutex_unlock(&mutex);
 }
}

void *Consumer() {
 while(1)  {
   pthread_mutex_lock(&mutex);
   pthread_cond_wait(&cond, &mutex);  // Чтобы не делать холостой цикл, если условие
не наступило
   printf("\nPong: %d", --data);
   pthread_mutex_unlock(&mutex);
  }
 }
```

```
int main()  {
  pthread_t thread1, thread2;
  pthread_cond_init(&cond, NULL);
  pthread_mutex_init(&mutex, NULL);
  data = 0;
  pthread_create(&thread1, NULL, &Producer,
NULL);
  pthread_create(&thread2, NULL, &Consumer,
NULL);
  …
}
```

Самарский аэрокосмический университет им. акад. С.П. Королева

Архитектура современных ОС
Часть IV

Составитель: к.т.н., доц. К.Е. Климентьев

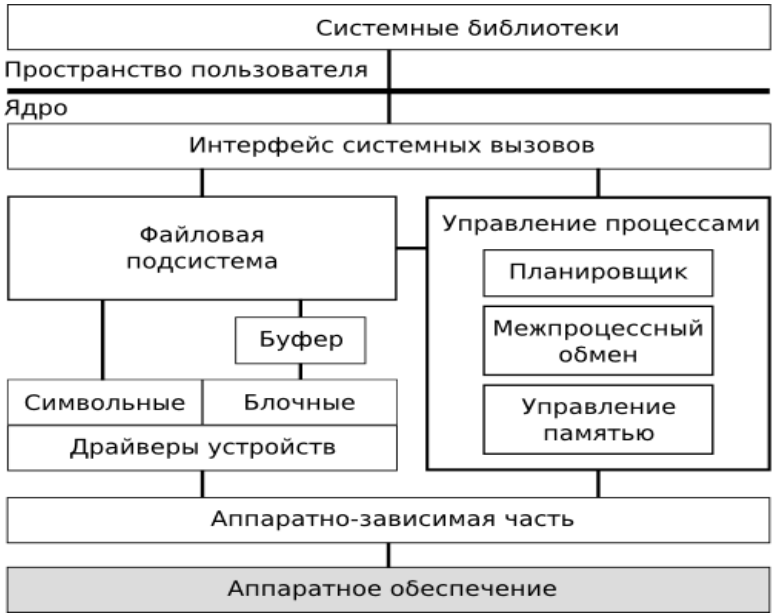
Самара 2013

1. Дисковые файлы и файлы устройств

Файл – именованный набор данных все зависимости от его месторасположения и формы представления. Базовые операции над файлом: 1) чтение; 2) запись. Вспомогательные операции: перемещение указателя. Служебные операции: 1) открытие; 2) закрытие.

Примеры:
- /etc/passwd – дисковый файл (описание пользователей);
- /dev/sda – носитель или /dev/hda1 – раздел носителя;
- /dev/tty1 – консоль, /dev/ttys0 – com-порт и пр.;
- /dev/mem – физическая память;
- /dev/random – ДСЧ, /dev/urandom – ДПСЧ;
- /dev/null, /dev/zero, /dev/full и пр. – «спецфайлы»;
- канал (будет дальше);
- сокет (будет дальше).

1.1. Архитектура UNIX



2. «Буферизованный» или «стандартный» ввод-вывод

Буфер располагается в адресном пространстве (в хипе или стеке) задачи. Основные функции:

- fopen() и fclose() – открыть и закрыть файл;
- fread() и fwrite() – читать и писать файл;
- fseek() – перемещение указателя в файле;
- fflush() – форсировать передачу из буфера и в буфер;
- fgetc(), fputc(), fgets(),fputs(), fprintf(), fscanf() – операции форматированного ввода-вывода;
- getc(), putc(), gets(),puts(), printf(), scanf() – операции форматированной работы с консолью;
- sgetc(), sputc(), sgets(),sputs(), sprintf(), sscanf() – форматированная работа со строкой.

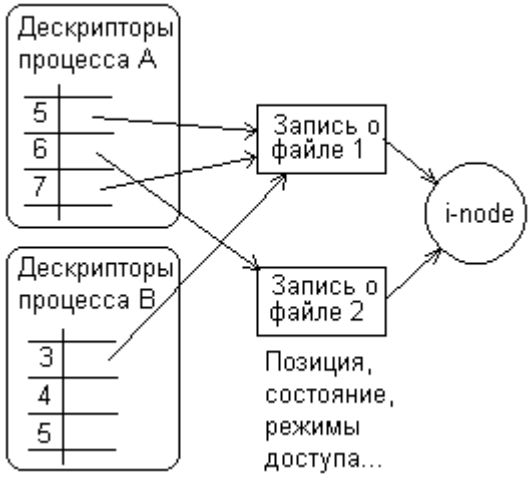
```
#include <stdio.h>
main() { // Побайтовое копирование файла описания пользователей
 FILE *f1, *f2; unsigned char c;
 f1 = fopen("/etc/passwd", "r+b"); // Readonly + binary
 f2 = fopen("./file.txt", "w");
while (1) {
  if (feof(f1)) break;
  fread( &c, sizeof(unsigned char), 1, f1  );
  fwrite( &c, sizeof(unsigned char), 1, f2 );
  fprintf(stdout, "%c", c);  // stdin, stdout, stderr
 }
fclose(f1); fclose(f2);
}
```

3. «Простой» или «базовый» ввод-вывод

Основные функции:
- int open (const char * filename, int flags, mode_t mode) – открытие или создание файла; флаги в fcntl.h: O_RDONLY, O_WRONLY, O_RDWR, O_APPEND, O_TRUNC, O_CREAT и пр.; режимы доступа в /sys/stat.h: S_IREAD, S_IWRITE,S_IEXEC и прочие биты.
- int creat(const char *filename, mode_t mode) эквивалентно int open(…, O_WRONLY|O_CREAT|O_TRUNC,…);
- ssize_t read (int fd, void * buffer, size_t count) – чтение файла;
- ssize_t write (int fd, const void * buffer, size_t count) – запись в файл;
- int close (int fd) – закрытие файла;
- off_t lseek (int fd, ott_t offset, int against) – перемещение указателя в файле, режимы в fcntl.h: SEEK_SET=0, SEEK_CUR=1, SEEK_END=2;
- pread() и pwrite() – чтение и запись без перемещения указателей.

3.1. Файловые дескрипторы

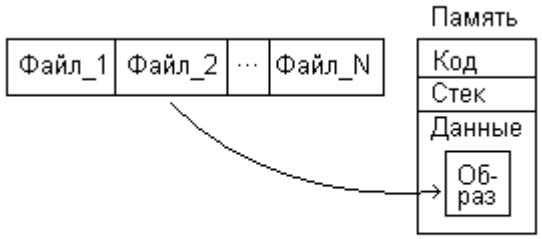
```
#include <stdio.h> #include <fcntl.h>
#include <errno.h>

main() { // Чтение даты создания BIOS
 int r, f, i; unsigned char c;
 f = open("/dev/mem", O_RDONLY, 0);
 lseek(f, 0xFFFF5, SEEK_SET);
 for (i=0;i<8;i++) {
  r = read(f, &c, sizeof(char));
  write(1, &c, sizeof(char)); // STDIN_FILENO=0,

            STDOUT_FILENO=1, STDERR_FILENO=2
 }
 close(f);
}
```



4. Проецирование файлов в память

Основные функции:
- mmap() – отображение файла в память, доступ как к массиву байтов;
- munmap() – отмена отображения.

4.1. Принцип отображения

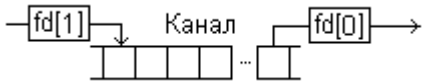



```
#include <stdio.h>
#include <fcntl.h>
#include <sys/mman.h>
#include <sys/types.h>
#include <sys/stat.h> main() {
 int r, f, i; char *m;
 f = open("/dev/mem", O_RDONLY, 0);
 m = (char *) mmap( 0, 0xFFFFF, PROT_READ, MAP_SHARED, f, 0);
 for (i=0xFFFF5;i<0xFFFFD;i++)  write(1, &m[i], 1);
 munmap( m, 0xFFFFF);
 close(f);
}
```

5. Каналы ввода-вывода (“пайпы”)

Каналы (пайпы) – файлы с последовательным доступом. Назначение: IPC. Функции:
- pipe() – создать и открыть неименованный канал с 2 дескрипторами: для ввода и вывода;
- mkfifo() – создать именованный канал;
- open(), read(), write() – назначение таково же, как для обычных файлов.

5.1. Принцип доступа

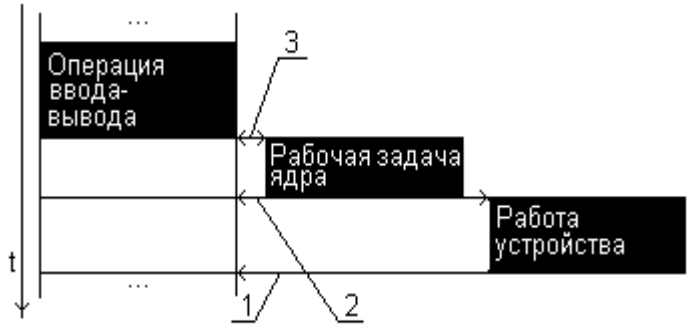



```
#include <stdlib.h> #include <stdio.h>

int pd[2]; // Pipe descriptors
int main()  {
  int d1=1234, d2;
  pipe(pd); // create pipe
  write(pd[1], &d1, sizeof(int));
  read(pd[0], &d2, sizeof(int));
  printf("%u", d2);
  close(pd[0]);  close(pd[1]);
}
```

```
#include <sys/stat.h>
#include <fcntl.h>
#include <string.h>
#include <stdio.h>
main () {
 int f1, f2, d1=1234, d2, q;
 q=mkfifo("/tmp/tralala", 0x777); // Create
 f2 = open("/tmp/tralala",
O_RDONLY|O_NONBLOCK); // !!!
 f1 = open("/tmp/tralala", O_WRONLY);
q = write(f1, &d1, sizeof(int));
close(f1);
 q = read(f2, &d2, sizeof(int));
 close(f2);
 printf("%u", d2);
}
```

6. Синхронный и асинхронный ввод-вывод

6.1. Модели ввода-вывода



Модели ввода-вывода:
1) Блокирующий синхронизированный;
2) Блокирующий синхронный;
3) Асинхронный.

Модели оповещения о завершении асинхронного ввода-вывода:
A) проверка признака готовности;
B) ожидание готовности;
C) обработка сигнала готовности SIGIO.

7. Мультиплексированный» ввод-вывод

Функции и макросы:

- select(maxdesc, fd_set *rd, fd_set *wd, fd_set *ed, timeval *t) – ожидание завершения + разрушение списка;
- select(maxdesc, fd_set *rd, fd_set *wd, fd_set *ed, timeval *t, sigset_t *sigmask) + реакция на сигналы.
- FD_ZERO() – очищает список;
- FD_SET() – добавляет к списку дескриптор;
- FD_CLR() – удаляет дескриптор;
- FD_ISSET() – проверяет готовность дескриптора.

```
#include <stdio.h>
#include <sys/time.h>
#include <sys/types.h>
#include <unistd.h>
main() {
 int len, ret;
fd_set readfds; // Список дескрипторов ввода
struct timeval tv;  // Параметры тайм-аута
unsigned char buf[16];
 FD_ZERO(&readfds); // Очистка списка
 FD_SET( STDIN_FILENO, &readfds); // Добавление операции ввода
 tv.tv_sec=5;  // Тайм-аут ожидания
 tv.tv_usec=0; // Микросекунды
 ret = select(STDIN_FILENO+1, &readfds, NULL, NULL, &tv); // Ждем хотя бы одного
 if (ret==0) {
  printf("\nGame over\n");
 }
 else  {
  len=read(STDIN_FILENO, buf, 16);
  write (STDOUT_FILENO, buf, len);
   }
}
```

8. Асинхронный ввод-вывод

Функции:
- aio_read() – инициализация ввода;
- aio_write() – инициализация вывода;
- lio_listio() – инициализация списка операций;
- aio_error() – возвращает статус асинхронной операции;
- aio_fsync() – ждет завершения всех операций вывода;
- aio_cancel() – прерывает незавершенную операцию ввода-вывода.

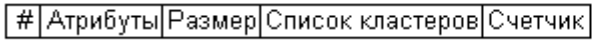
```
#include <sys/types.h> // gcc example.c -o example -lrt
#include <aio.h>
#include <fcntl.h>
#include <errno.h>
  struct aiocb cb;
int number, f;

int main() {
  f = open("/dev/random", O_RDONLY, 0);
  memset(&cb, 0, sizeof(struct aiocb));
  cb.aio_nbytes = sizeof(int);
  cb.aio_fildes = f;
  cb.aio_offset = 0;
  cb.aio_buf = &number;
  aio_read(&cb);
  while(aio_error(&cb) == EINPROGRESS); // Ждем
  printf(“0x%X”, number);
}
```

9. Файловая система

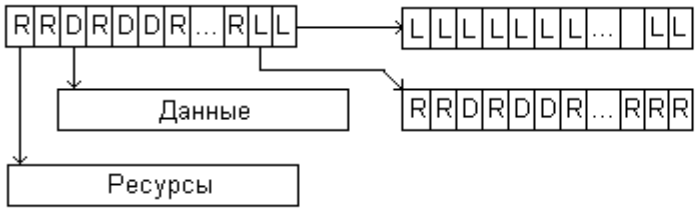
Файловые системы UNIX: ext*fs (Linux), HFS и HFS+ (Mac OS X), UFS (BSD) и пр.

9.1. Диск

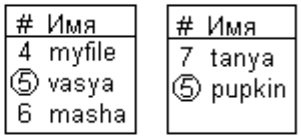
9.2. i-node



9.3. Список кластеров



9.4. Каталоги



9.5. Файлы ссылок

masha - символическая ссылка
6 masha - псевдоним (alias).

ГОУВПО
«САМАРСКИЙ ГОСУДАРСТВЕННЫЙ АЭРОКОСМИЧЕСКИЙ УНИВЕРСИТЕТ
имени академика С.П. Королева»

Краткий справочник по UNIX в вопросах и ответах

Методические указания к лабораторному практикуму

Сост.: к.т.н., доц. К.Е. Климентьев

САМАРА 2012

1. Задачи и вопросы общего назначения
2. Работа с дисками, каталогами и файлами
3. Работа со внешними устройствами (флэшкой)
4. Работа с процессами в памяти
5. Работа с Midnight Commander
6. Работа с текстами и архивами
7. Компиляция и отладка программ
Примеры заданий для закрепления материала

1. Задачи и вопросы общего назначения

Q: Что в этой методичке понимается под UNIX?
A: Семейство частично и полностью POSIX-совместимых операционных систем, образовавшихся под влияним UNIX, например: операционные системы на ядре Linux (Debian, Ubuntu, RedHat, Fedora, Slackware, Gentoo), на ядре Darwin (FreeBSD, OpenBSD, Mac OS X), некоторые операционные системы реального времени (QNX, VxWorks), некоторые оригинальные системы (Solaris, Xenix, HP-UX) и пр.

Q: Как узнать логин и пароль для root?
A: Для операционных систем Debian он принципиально недоступен пользователю. Для некоторых LiveCD (например, Slax или Porteus) логин «root», пароль «toor». Для некоторых сборок Ubuntu логин «root», пароль «thoughtpolice».

Q: Как выключить компьютер
A: Например, так: «shutdown –h 0» (где «0» - задержка в минутах) или: «shutdown now». Можно еще так: «halt». Или так: «reboot».

Q: Каково назначение различных каталогов UNIX
A: Вот наиболее важные:

- /root – корневой каталог, домашний каталог суперпользователя
- /home – содержит домашние подкаталоги пользователей, работать рекомендуется здесь
- /bin – программы и утилиты операционной системы
- /dev – файлы устройств
- /lib - библиотеки
- /mnt – сюда монтировать съемные устройства
- /usr/include - .h-файлы для компилятора

Q: Как выполнить команду от имени другого пользователя

A: Например, так: «sudo команда», будет запрошен пароль текущего пользователя

Q: Как переключиться на другого пользователя
A: Например так: «su root», будет запрошен пароль

Q: Как узнать распределение оперативной памяти
A: Например так: «free».

Q: Как узнать распределение дисковой памяти
A: Например так: «df».

Q: Как узнать загруженность процессора
A: Например так: «vmstat».

Q: Как узнать версию ядра операционной системы
A: Например так: «uname».

2. Работа с дисками, каталогами и файлами

Q: Как посмотреть содержимое текущего каталога
A: Если только файлы, то просто: «ls». Если с каталогами, то: «ls –l».

Q: Как перейти в конкретный каталог
A: Например, так: «cd /mnt/sda1_removable».

Q: Как перейти в дереве каталогов на один уровень вверх
A: «cd ..»

Q: Как создать каталог
A: Например, так: «mkdir /home/masha»

Q: Как убить каталог
A: Например, так: «rmdir /home/vasya». Каталог должен быть пуст!

Q: Как скопировать один файл в другой
A: Например, так: «cp oldfile.txt newfile.txt»

Q: Как переименовать или переместить файл
A: Например, так: «mv /home/vasya/myname.txt /home/masha/yourname.txt»

Q: Как убить файл
A: Например, так: «rm myfile.txt»

Q: Что означают биты свойств файлов

A: Первый бит – тип файла: d – каталог; b – блочное устройство; c – символьное устройство; s – сокет; p – канал; l – ссылка на файл. Далее 9 битов, разделенных на группы по 3. Первая группа касается владельца, вторая группы владельца, третья – всех остальных: r – чтение, w – запись, x – исполнение. Например: «---x--x--x» – «невидимый» файл только для выполнения.

Q: Как изменить биты свойств файла или каталога
A: Например, так: «chmode 666 myfile.txt».

Q: Как изменить хозяина файла или каталога
A: Например, так: «chown user /home/myfile.txt».

Q: Как протестировать файловую систему
A: Например, так: «fsck».

3. Работа со внешними устройствами (флэшкой)

Q: Как смонтировать флэшку?
A: Командой типа «mount –rw /dev/sda1 /mnt/sda1_removabe», где «sd*» должны соответствовать файлам устройств, реально присутствующим в «/dev» и «/mnt». Варианты: «mount /dev/sdb1 /mnt/sdb1 –t vfat –o umask=000,dmask=000» - с указанием файловой системы и масок для установки битов доступности компонентам смонтированной файловой системы. Если каталога «/mnt/sda1_removabe» нет, его можно создать командой «mkdir».

Q: Флэшка смонтировалась, как посмотреть содержимое?
A: Командой типа «ls –l /mnt/sda1_removable».

Q: А как избавиться от кракозябров в именах файлов и каталогов вместо русских букв?
A: Простого способа нет. Но можно попытаться смонтировать флэшку так: «mount /dev/sdb1 /mnt/sdb1 –codepage=866,iocharset=koi8-r».

Q: Как размонтировать флэшку, чтобы вынуть?
A: Командой типа «umount /mnt/sda1_removabe».

4. Работа с процессами в памяти

Q: Как посмотреть процессы, запущенные текущим пользователем?
A: Командой «ps».

Q: Как посмотреть все процессы?
A1: Командой «ps –e» или просто «ps» (для работающих в текущей сессии).
A2: Утилитой «top». Команды: «Q» – выход; «K» - убить процесс; «N» - отсортировать по PID; «R» - переопределить приоритет;

A3: Командой «pstree» в виде иерархии материнских и дочерних процессов.

Q: Как прекратить выполнение зациклившейся задачи?
A1: Если задача текущая, то послать ей сигнал завершения командой «CTRL-C».
A2: Если задача текущая, то перевести ее в фоновый режим при помощи «CTRL-E» и послать ей сигнал завершения командой «kill имя».
A3: Если задача фоновая, то послать ей сигнал завершения командой «kill имя».
A4: Утилитой «top».

5. Работа с MidNight Commander

Q: Как запустить Midnight Commander?
A: Командой «mc»

Q: Как работать с Midnight Commander?
A: Вот основные клавиатурные комбинации:

Клавиши	Назначение
TAB	переключение между панелями
F3	просмотр файла
F4	редактирование файла
F5	копирование файла
F6	переименование (перемещение) файла
F7	создание каталога
F8	удаление файла
F9	активизация меню
F10	выход и командной оболочки
CTRL+O	убрать обе панели
CTRL+U	поменять панели местами
Insert	пометка файлов
плюс	выбор группы файлов

Q: Как раабботать со встроенным в Midnight Commander текстовым редактором?
A: Вот основные клавиатурные комбинации:

Клавиши	Назначение
F3	Начать/закончить выделение текста
Shift+F3	Начать/закончить выделение блока текста
F5	Скопировать выделенный текст
F6	Переместить выделенный текст
F8	Удалить выделенный текст
ESC+i	Вкл/выкл автовыравнивания возвратом каретки

ESC+l	Переход к строке по её номеру
ESC+q	Вставка непечатного символа.
ESC+t	Сортировка строк выделенного текста
ESC+u	Выполнить внешнюю команду и вставить ее результат
Ctrl+f	Выделенный фрагмент записать во внешний файл
Ctrl+k	Удалить часть строки до конца строки
Ctrl+n	Создать новый файл
Ctrl+s	Включить или выключить подсветку синтаксиса
Ctrl+t	Выбрать кодировку текста
Ctrl+u	Отменить действия
Ctrl+x	Перейти в конец следующего
Ctrl+y	Удалить строку
Ctrl+z	Перейти на начало предыдущего слова
Shift+F5	Прочитать внешний файл
ESC+Enter	Диалог перехода к определению функции
ESC+-	Возврат после перехода к определению функции
ESC++	Переход вперед к определению функции
ESC+n	Включение/отключение отображения номеров строк
tab	Отодвигает вправо выделенный текст
ESC-tab	Отодвигает влево выделенный текст
Shift+Стрелки	Выделение текста
ESC+Стрелки	Выделение вертикального блока
ESC+Shift+-	Вкл/выкл отображения невидимых символов
ESC+Shift++	Вкл/выкл автовыравнивания возвратом каретки

6. Работа с текстами и архивами

Q: Как выдать на экран текст файла?
A: Например, так: «cat myfile.txt»

Q: Как выдать на экран большой текст по частям
A: Например, так: «cat myfile.txt | more»

Q: Как сравнить два двоичных файла
A: Например, так: «cmp file1 file2».

Q: Как сравнить два текстовых файла
A: Например, так: «diff file1 file2».

Q: Как получить статистику о текстовом файле
A: Например, так: «wc file».

Q: Как отсортировать строки в файле по алфавиту
A: Например, так: «sort file».

Q: Как перенаправить вывод команды в файл, создав новый
A: Воспользоваться перенаправителем «>», например, так: «ls >file».

Q: Как перенаправить вывод команды, добавив его к существующему файлу
A: Воспользоваться перенаправителем «>>», например, так: «ls >>file».

Q: Как распаковать архив?
A: Если он имеет расширения .tgz или tar.gz, то командой типа: «tar xfvz *.tgz». Если расширение .bz или .tbz, то: «tar xfvj *.tbz».

7. Компиляция программ

Q: Как сразу скомпилировать и скомпоновать программу
A: Например, так: «cc myprog.c». Результат попадет в файл «a.out».

Q: Как подключить к программе при компоновке дополнительную библиотеку
A: Например, так: «cc –lpthread myprog.c»

Q: Как скомпилировать и скомпоновать в определенный файл
A: Например, так: «cc myprog.c –o myprog»

Q: Как получить ассемблерный листинг с ошибками?
A: Например, так: «cc –Wa, –ahls=myprog.lst myprog.c»

Примеры заданий для закрепления материала

1. Создать текстовый файл, содержащий следующую информацию:
   - текущую дату и время;
   - информацию о ядре операционной системе;
   - количество свободной оперативной и дисковой памяти;
   - иерархию запущенных процессов;
   - список подкаталогов каталога /root, отсортированный по алфавиту.

ГОУВПО
«САМАРСКИЙ ГОСУДАРСТВЕННЫЙ АЭРОКОСМИЧЕСКИЙ УНИВЕРСИТЕТ
имени академика С.П. Королева»

Многозадачное программирование в UNIX

Методические указания к лабораторному практикуму

Сост.: к.т.н., доц. К.Е. Климентьев

САМАРА 2012

Введение

Многозадачность – это режим работы операционной системы, при котором параллельно могут выполняться несколько программ (задач). Для создания эффекта одновременности программы (задачи) выполняются поочередно короткими фрагментами. Существуют две разновидности многозадачности:

- *кооперативная* – при которой задача сама принимает решение о передаче управления другой задаче;
- *вытесняющая* – при которой операционная система через небольшие *кванты времени* принудительно прерывает выполнение одной задачи, чтобы передать управление другой.

В современных версиях операционных систем UNIX реализована вытесняющая многозадачность.

Процесс – задача, монопольно занимающая выделенное ей адресное пространство и владеющая некоторым набором ресурсов. Все процессы конкурируют друг с другом за процессорное время. Они могут порождать и уничтожать друг друга, таким образом существуют *процессы-предки* и *процессы-потомки*. Самым первым процессом в UNIX, являющимся прямым или косвенным предком всех остальных, является «init». Процессы, не имеющие прямого предка (например, он уже завершен), называются «*зомби-процессами*».

Потоки – задачи, разделяющие общее адресное пространство. Наличие их является требованием стандарта POSIX. Однако, потоки отсутствуют во многих версиях UNIX и UNIX-подобных систем.

Если задачи (процессы или потоки) выполняются согласованно друг с другом, то говорят, что их выполнение *синхронизировано*. В стандартные библиотеки UNIX включены многочисленные функции, обеспечивающие синхронизацию задач.

1. Создание и уничтожение процессов

Для программирования с использованием функций, описанных в этом разделе, необходимо подключить файлы «stdlib.h» или «unistd.h».

1.1. Функция system()

Простейший способ запуска процесса – использование функции system(), использующей в качестве «посредника» интерпретатор команд пользователя. Таким образом, работа функции system() идентична «ручному» запуску процесса из командной строки консоли.

int system (char *s) – выполняет команду пользователя, заданную в строке s. Возвращает –1 в случае ошибки или 127 в случае невозможности запуска командного интерпретатора.

1.2. Функции fork(), exec*(), wait() и *exit()

Эти функции обеспечивают запуск процесса и работу с ним при помощи прямого запроса к ядру операционной системы, минуя интерпретатор команд.

pid_t fork(void) – создает точную копию адресного пространства процесса, вызвавшего функцию, и запускает в нем дочернюю копию текущего (родительского) процесса с команды, следующей за fork(). Родительская копия также продолжает выполнение с той же точки. Возвращает:

- для родительского процесса - системный идентификатор процесса PID, который будет загружен в это адресное пространство;
- для дочернего процесса - 0.

```
int execlp( const char *file, const char *arg0,
                ... const char *argN,(char *) NULL );
int execvp( const char *file, char *argv[] );
int execl( const char *path, const char *arg0,
                ... const char *argN,(char *) NULL );
int execv( const char *path, char *argv[] );
int execle( const char *path, const char *arg0,
                 ... const char *argN,(char *)NULL, char * envp[] );
int execve( const char *path, char *argv[], char *envp[] );
```

- загружают во вновь созданное (при помощи fork()) адресное пространство программу и запускают ее на выполнение. Параметры:

- file – строка имени файла, находящегося в текущем каталоге или в одном из умалчиваемых каталогов;
- path – полное имя файла, включая путь к нему;

- env, envp – массив (строка) переменных среды;
- argn – указатель на количество параметров, передаваемых процессу;
- argv, arg0, arg1… - строки параметров, передаваемых процессу.

Все эти функции в случае ошибки возвращают –1. При этом переменная errno может принять значения: ENOEXEC – неверный формат программы; ENOMEM – не хватает памяти.

pid_t wait (int *status_ptr) – переводит родительский процесс в состояние ожидания завершения любого дочернего процесса. Для использования этой функции необходимо подключить «wait.h».

pid_t getpid(void) – возвращает для обратившегося процесса его системный идентификатор PID.

void exit (int status) – завершает работу текущего процесса, возвращая процессу-родителю статус. Автоматически закрываются все открытые файлы, освобождаются захваченные области памяти и т.п.

void _exit (int status) – завершает работу текущего процесса, возвращая процессу-родителю статус. «Сборка мусора» не выполняется.

2. Потоки в стиле POSIX

Потоки – задачи, использующие общее адресное пространство. Для работы с потоками необходимо подключение заголовочного файла «pthread.h» и библиотеки «pthread» (например, «cc proba.c -pthread»). Используются функции:

int pthread_create(pthread_t * thread, pthread_attr_t *attr, void* (*start_routine) (void*), void * arg) - создаёт новый поток. Возвращает 0 в случае успеха или –1 в случае ошибки. Параметры:

- thread – возвращаемый идентификатор потока (если NULL, то не возвращается);
- attr – атрибуты запуска (0 – по умолчанию);
- start_routine – адрес функции потока;
- arg – аргумент, передаваемый функции.

Возвращает 0 в случае успеха.

pthread_t pthread_self(void) - возвращает потоку собственный идентификатор.

void pthread_exit(void *retval) - завершает текущий поток. Параметр:

- retval – адрес для кода возврата.

int pthread_join(pthread_t th, void **thr_return) - ожидает завершения потока. Параметры:

- ht – идентификатор потока;
- thr_return – указатель на код возврата.

int pthread_kill (pthread_t thread, int signo) - посылает потоку сигнал. Параметры:

- thread – идентификатор потока;
- signo – код сигнала.

int sched_yield(void) – передать управление другим потокам, не ожидая завершения кванта. При успешном выполнении возвращает 0, а в случае неудачи –1.

int pthread_cancel(pthread_t thread_id) – принудительно завершает работу потока. Параметр:

- thread – идентификатор потока;

void pthread_cleanup_push(void (*handler) (void *), void *arg) – устанавливает обработчик, вызываемый перед принудительным завершением текущего потока. Параметры:

- handler – функция обработчика;
- arg – аргумент, передаваемый обработчику.

void pthread_cleanup_pop(int execute) – сбрасывает ранее установленный обработчик, вызываемый перед принудительным завершением текущего потока. Параметр:

- execute – вызвать ли обработчик перед сбросом.

3. Синхронизация задач

3.1. Сигналы

Сигналы – это программный аналог прерывания. Сигнал может быть послан от одного процесса другому или всем сразу. Сигналы посылаются со стороны операционной системы процессам, при выполнении которых обнаружены исключительная ситуация. Имена сигналов определены в файле signal.h:

- SIGABRT - сигнал аварийного завершения процесса; подразумеваемая реакция предусматривает, помимо аварийного завершения, создание файла с образом памяти процесса.
- SIGALRM - срабатывание будильника; подразумеваемая реакция - аварийное завершение процесса.

- SIGCHLD - завершение, остановка или продолжение порожденного процесса; подразумеваемая реакция - игнорирование.
- SIGCONT - продолжение процесса, если он был остановлен; подразумеваемая реакция - продолжение выполнения или игнорирование (если процесс не был остановлен).
- SIGFPE - некорректная арифметическая операция; подразумеваемая реакция - аварийное завершение и создание файла с образом памяти процесса.
- SIGILL - некорректная команда; подразумеваемая реакция - аварийное завершение и создание файла с образом памяти процесса.
- SIGINT - сигнал прерывания, поступивший с терминала; подразумеваемая реакция - аварийное завершение процесса.
- SIGKILL - уничтожение процесса (этот сигнал нельзя перехватить для обработки или проигнорировать); подразумеваемая реакция - аварийное завершение процесса.
- SIGPIPE - попытка записи в канал, из которого никто не читает; подразумеваемая реакция - аварийное завершение процесса.
- SIGQUIT - сигнал выхода, поступивший с терминала; подразумеваемая реакция - аварийное завершение и создание файла с образом памяти процесса.
- SIGSEGV - некорректное обращение к памяти; подразумеваемая реакция - аварийное завершение и создание файла с образом памяти процесса.
- SIGSTOP - остановка выполнения (этот сигнал нельзя перехватить для обработки или проигнорировать); подразумеваемая реакция - остановка процесса.
- SIGTERM - сигнал терминирования; подразумеваемая реакция - аварийное завершение процесса.
- SIGTSTP - сигнал остановки, поступивший с терминала; подразумеваемая реакция - остановка процесса.
- SIGTTIN - попытка чтения из фонового процесса; подразумеваемая реакция - остановка процесса.
- SIGTTOU - попытка записи из фонового процесса; подразумеваемая реакция - остановка процесса.
- SIGUSR1, SIGUSR2 - определяемые пользователем сигналы; подразумеваемая реакция - аварийное завершение процесса.
- SIGPOLL - опрашиваемое событие; подразумеваемая реакция - аварийное завершение процесса.
- SIGPROF - срабатывание таймера профилирования; подразумеваемая реакция - аварийное завершение процесса.
- SIGSYS - некорректный системный вызов; подразумеваемая реакция - аварийное завершение и создание файла с образом памяти процесса.
- SIGTRAP - попадание в точку трассировки/прерывания; подразумеваемая реакция - аварийное завершение и создание файла с образом памяти процесса.

- SIGVTALRM - срабатывание виртуального таймера; подразумеваемая реакция - аварийное завершение процесса.
- SIGXCPU - исчерпан лимит процессорного времени; подразумеваемая реакция - аварийное завершение и создание файла с образом памяти процесса.
- SIGXFSZ - превышено ограничение на размер файлов; подразумеваемая реакция - аварийное завершение и создание файла с образом памяти процесса.

Для работы с сигналами используются следующие функции:

int kill (pid_t pid, int sig) – посылает указанный сигнал процессу с указанным идентификатором. Параметры:

- pid – идентификатор процесса (частные случаи: 0 – всем процессам из той же группы, -1 – всем процессам, кому данный процесс имеет право посылать сигналы);
- sig – код сигнала.

Возвращает 0 в случае успеха или –1 в случае ошибки.

void abort(void) – послать сигнал SIGABRT**.**

int raise (int sig) – послать указанный сигнал самому себе.

int sigaction (int sig, const struct sigaction *restrict act, struct sigaction *restrict oact) – позволяет устанавливать и изменять способ обработки сигнала в стиле стандарта POSIX. Параметры:

- sig – обрабатываемый сигнал;
- restrict_act – новый способ обработки сигнала;
- restrict_oact – старый способ обработки сигнала.

Тип sigaction:

```
struct sigaction {
  void (*sa_handler) (int);// Указатель на простой обработчик
  sigset_t sa_mask;        // Блокируемые во время обработки сигналы
  int sa_flags;            // Флаги, влияющие на поведение сигнала
  void (*sa_sigaction) (int, siginfo_t *, void *); // Указатель на обработчик
}
```

int pause (void) – вызывает задержку текущего процесса до прихода любого сигнала. В случае ошибки возвращает –1.

int sigwait(const sigset_t *set, int *signum) – вызывает ожидание конкретного сигнала. Параметры:

- set – набор ожидаемых сигналов;

- signum – номер полученного сигнала.

В случае успеха возвращает 0, иначе возвращает код ошибки.

3.2. Семафоры

Семафор – универсальный механизм, обеспечивающий синхронизацию задач. Существуют, по крайней мере, два набора функций для работы с семафорами.

3.2.1. Семафоры в стиле System V

Эти функции сохранились в UNIX-системах с 1970-годов и обеспечивают «традиционную» работу с семафорами. Их описания находятся в заголовочных файлах «sys/ipc.h» и «sys/sem.h».

key_t ftok (const char *path, int id) – создает ключ для объекта ядра (семафора, очереди сообщений и т.п.) операционной системы. Параметры:

- path – имя объекта (существующий и доступный файл);
- id – желаемый номер объекта (>0).

Возвращает –1 в случае ошибки или заполненный key_t.

int semget (key_t key, int nsems, int flags) – создает набор (неициализированных!) семафоров. Параметры:

- key – ключ для набора семафоров;
- nsems – количество семафоров в наборе;
- flags – битовые флаги (IPC_CREAT – создать новый семафор, IPC_EXCL – не создавать копию существующего семафора).

Возвращает –1 в случае ошибки или идентификатор семафора.

int semctl (int semid, int semnum, int cmd, union semun arg) - позволяет выполнять операции, определенные командой cmd над набором семафоров, указанным в semid. Первый семафор в наборе обозначен величиной 0 для semnum. Параметры:

- semid – идентификатор семафора;
- semnum – номер семафора;
- cmd – действие над семафорами (IPC_STAT - получить информацию о наборе семафоров; IPC_SET - переустановить характеристики набора семафоров; IPC_RMID - удалить из системы набор семафоров; SETVAL - устанавливает значение семафорной переменной; GETVAL – получает значение семафорной переменной; GETALL - получает значение для всех семафоров в наборе; SETALL – устанавливает значение для всех семафоров в наборе; GETPID – получает PID последнего процесса, обратившегося к семафору);
- arg – аргумент команды типа «semun»:

```
union semun {
        int val;                    // Целое число
        struct semid_ds *buf;       // semid
        unsigned short array        // массив байтов
        }
```

и

```
struct semid_ds {
        struct ipc_perm;            // Права доступа
        unsigned short sem_nsems;   // Размер набора
        time_t sem_otime;           // Последнее обращение к semop
        time_t sem_ctime            // Последнее обращение к semctl
        }.
```

int semop (int semid, struct sembuf *sops, size_t nsops) – управляет семафором или группой семафоров. Параметры:

- semid – идентификатор семафора;
- sembuf – буфер операций типа «sembuf»:

```
struct sembuf {
        unsigned short sem_num; // Номер семафора
        short sem_op;           // Операция
        short sem_flg           // Флаги (рекомендуется SEM_UNDO)
        },
```

возможны операции: >0 – увеличить значение семафора на sem_op; 0 – процесс блокируется, пока семафор не обнулится; <0 – процесс блокируется, пока текущее значение семафора не сравняется или не превысит |sem_op|, если это уже достигнуто, то |sem_op| вычитается из значения семафора.

- nsops - количество операций в буфере.

3.2.2 Семафоры в стиле POSIX

Эти функции появились в UNIX-системах в связи с требованиями стандарта POSIX. Их описания находятся в заголовочном файле «semaphore.h». Они присутствуют не во всех UNIX-системах.

sem_t sem_open (char*name, int flags, mode_t mode, unsigned int value) - создает новый или открывает существующий семафор. Возвращает указатель на sem_t, или -1 в случае неудачи. Параметры:

- name – имя семафора (должно начинаться с символа «/»);

- flags - 0, если семафор уже существует; O_CREAT – создать новый; O_EXCL – возвратить ошибку, если семафор существует.
- mode - это права доступа;
- init_value – начальное значение семафора.

int sem_getvalue (sem_t *idp, int*valuep) - определяет текущее значение семафора. Всегда возвращает 0. Параметры:

- idp – семафор;
- valuep – значение семафора.

int sem_wait (sem_t *idp) - уменьшает значение семафора на единицу. Если его значение меньше или равно 0, то вызывающий процесс блокируется. Функция всегда возвращает 0. Параметр:

- idp – семафор.

int sem_trywait (sem_t *idp) – если значение >0, то уменьшает значение семафора на единицу и возвращает 0; если значение уже меньше или равно 0, то возвращает EAGAIN. Параметр:

- idp – семафор.

int sem_post (sem_t *idp) - увеличивает значение семафора на единицу. В случае успеха возвращает 0, в случае слишком большого значения возвращает *ERANGE*. Параметр:

- idp – семафор.

int sem_close (sem_t *idp) – отключает семафор. В случае успеха возвращает 0, а в случае неудачи -1. Параметр:

- idp – семафор.

int sem_unlink (char *name) - удаляет семафор из системы. В случае успешного выполнения функция возвращает 0, а в случае неудачи -1. Параметр:

- name – имя семафора (должно начинаться с символа «/»);

int sem_init (sem_t *addr, int pshared, unsigned int value) – создает неименованный семафор. Параметры:

- addr – область памяти под семафор;
- pshared – доступность для других процессов;
- value – начальное значение.

В случае успешного выполнения данная функция возвращает 0, а в случае неудачи: EINVAL превышено максимально возможное количество семафоров в системе; ENOSYS аргумент pshared не равен нулю.

int sem_destroy (sem_t* idp) – удаляет неименованный семафор, созданный при помощи sem_init(). В случае успеха возвращает 0, а если есть заблокированные семафором задачи, то EBUSY.

3.2.3. Синхронизация потоков при помощи мьютексов

Мьютекс – двоичный семафор, способный принимать значения «свободно» и «занятно». Для работы с потоками необходимо подключение заголовочного файла «pthread.h». Для синхронизации используются функции:

int pthread_mutex_init (pthread_mutex_t *mutex, const pthread_mutexattr_t *mutexattr) - инициализирует мьютекс. Параметры:

- mutex - возвращаемый идентификатор мьютекса;
- mutexattr – это указатель на атрибуты или 0, если по умолчанию.

int pthread_mutex_lock (pthread_mutex_t *mutex) – попытка захвата мьютекса. Если мьютекс уже захвачен, то поток блокируется до его освобождения. Параметр:

- mutex – идентификатор мьютекса.

int pthread_mutex_trylock (pthread_mutex_t *mutex) – не блокирующая попытка захвата мьютекса. Если мьютекс уже захвачен, то возвращает код ошибки EBUSY, иначе 0. Параметр:

- mutex – идентификатор мьютекса.

int pthread_mutex_unlock (pthread_mutex_t *mutex) - освобождает мьютекс. Параметр:

- mutex – идентификатор мьютекса.

int pthread_mutex_destroy (pthread_mutex_t *mutex) – удаляет мьютекс. Параметр:

- mutex – идентификатор мьютекса.

3.2.4. Синхронизация потоков при помощи переменных состояния

int pthread_cond_init (pthread_cond_t *cond, pthread_condattr_t *cond_attr) - инициализирует переменную состояния. Параметры:

- cond – возвращаемый идентификатор переменной состояния;
- cond_attr – атрибуты переменной состояния.

int pthread_cond_wait (pthread_cond_t *cond, pthread_mutex_t *mutex) - блокирует поток на переменной состояния, ожидая его наступления. Параметры:

- cond – идентификатор переменной состояния;

- mutex – идентификатор мьютекса.

int pthread_cond_timedwait (pthread_cond_t *cond, pthread_mutex_t *mutex, const struct timespec *abstime) - блокирует поток на переменной состояния на определенное время. Параметры:

- cond – идентификатор переменной состояния;
- mutex – идентификатор мьютекса;
- abstime – время.

int pthread_cond_signal (pthread_cond_t *cond) – сигнализирует о наступлении состояния и снимает блокировку с потока, ожидающего переменную состояния. Параметр:

- cond – идентификатор переменной состояния.

int pthread_cond_broadcast (pthread_cond_t *cond) - снимает блокировку со всех потоков, ожидающих переменную состояния. Параметр:

- cond – идентификатор переменной состояния.

int pthread_cond_destroy (pthread_cond_t *cond) – удаляет переменную состояния. Параметр:

- cond – идентификатор переменной состояния.

4. Информационный обмен между задачами

По умолчанию, каждый процесс выполняется в своем виртуальном адресном пространстве и, для обмена информацией с другими процессами, ему предоставляются операционной системой специальные средства.

4.1. Файловый ввод-вывод

Система организации файлов – базовый механизм межзадачного обмена. Он обеспечивает доступ не только к наборам данных, размещенных на носителях информации, но и к устройствам, объектам ядра операционной системы и т.п.

int open(char *path, int flags, mode_t perms) – открывает существующий или создает новый файл. Указатель чтения-записи устанавливается в 0. Возвращает дескриптор открытого файла или –1 в случае ошибки. Параметры:

- path – имя файла;
- flags – битовые флаги доступа (O_CREAT – создать новый, O_EXCL – при создании существующего файла вернуть ошибку, O_APPEND – только для добавления, O_RDONLY – только для чтения, O_WRONLY – только для записи, O_RDWR – для чтения-записи, O_BINARY – в

двоичном режиме, O_TEXT – в текстовом режиме, O_TRUNC – обрезать до 0 длины);
- perms – (S_IWRITE – разрешение записи, S_IREAD – разрешение чтения).

int creat(char *path, mode_t perms) – создает новый файл. Возвращает дескриптор созданого файла или –1 в случае ошибки. Параметры:
- path – имя файла;
- perms – (S_IWRITE – разрешение записи, S_IREAD – разрешение чтения).

ssize_t write(int fd, void *buf, size_t nbytes) – записывает данные в файл или устройство. Указатель чтения-записи перемещается автоматически. Возвращает количество успешно записанных байтов или –1. Параметры:
- fd – дескриптор файла или устройства;
- buf – буфер данных;
- nbytes – количество байтов для записи.

ssize_t read(int fd, void *buf, size_t nbytes) – читает данные из файла или устройства. Указатель чтения-записи перемещается автоматически. Возвращает количество успешно прочитанных байтов или –1. Параметры:
- fd – дескриптор файла или устройства;
- buf – буфер данных;
- nbytes – количество байтов для чтения.

int close(int fd) – закрывает файл или устройство. Параметр:
- fd – дескриптор файла или устройства.

off_t lseek(int fd, off_t pos, int whence) – изменяет положение указателя чтения-записи в файле. Параметры:
- fd – дескриптор файла;
- pos – относительное смещение;
- whence - точка отсчета (SEEK_SET – от начала файла, SEEK_CUR – от текущей позиции, SEEK_END – от конца файла).

4.2. Каналы

Каналы – механизм межзадачного обмена с последовательной структурой доступа (очередью типа FIFO – «первый пришел, первый вышел»). Чтение и запись в них осуществляются при помощи файловых функций read() и write(). Открытые дескрипторы закрываются при помощи close(). При их использовании в программе достаточно подключить файл «unistd.h». Типы данных описаны в «sys/types.h» и «sys/stat.h».

int pipe(int pfd[2]) – создает неименованный канал с двумя открытыми дескрипторами. Возможна передача их значений родительскому процессу в виде строковых параметров вызова. Возвращает 0 в случае успеха или –1 в случае ошибки. Параметр:

- pfd – массив дескрипторов (pfd[0] – для записи, pfd[1] – для чтения).

int mkfifo(char *path, mode_t perm) – создает именованный канал, доступ к которому затем выполняется при помощи файловых операций open() с разными правами доступа, read(), write() и close(). Возвращает 0 в случае успеха или –1 в случае ошибки.

- path – имя файла;
- perm – (S_IWRITE – разрешение записи, S_IREAD – разрешение чтения).

4.3. Очереди сообщений

4.3.1. Очереди сообщений в стиле SYSTEM V

Для программирования очередей сообщений могут понадобиться включаемые файлы «sys/types.h», «sys/ipc.h» и «sys/msg.h». Очередь сообщений является объектом, имеющим уникальный идентификатор (токен). Операционная система поддерживает несколько стандартных очередей, кроме того, существует возможность создавать и использовать новые. Буфер сообщения должен иметь следующую структуру:

```
struct msgbuf {
   long mtype;        /* тип сообщения */
   char mtext[1];     /* текст сообщения */
};
```

Если mtype=0, то работа идет с первым в очереди сообщением любого типа; если mtype>0, то с первым по счету сообщением указанного типа; если mtype<0, то – первое сообщение с номером, который равен или меньше указанного.

- **key_t ftok(const char *path, int id)** – см. раздел «Семафоры в стиле System V»

int msgget(key_t key, int flags) – возвращает идентификатор очереди сообщений, соответствующей указанному токену. Возвращает идентификатор или –1 в случае ошибки. Параметры:

- key_t key – токен;
- int flags – битовые флаги (IPC_CREAT – создать новую очередь, 9 младших битов – биты прав доступа).

int msgsnd(int msqid, const void *msgp, size_t msgsize, int flags) – записывает собщение в очередь. Возвращает 0 в случае успеха или –1 в случае ошибки. Параметры:

- msqid – идентификатор очереди;
- msgp – сообщение;
- msgsize – размер сообщения;
- flags – битовые флаги.

ssize_t msgrcv(int msqid, void *msgp, size_t mtextsize, long msgtype, int flags) – считывает сообщение из очереди. Возвращает число считанных байтов или –1 в случае ошибки. Параметры:

- msqid – идентификатор очереди;
- msgp – сообщение;
- msgsize – размер сообщения;
- msgtype – тип запрашиваемого сообщения;
- flags – битовые флаги.

int msgctl(int msgid, int cmd, struct msgid_ds *data) – управляет очередью сообщений. Возвращает 0 в случае успеха или –1 в случае ошибки. Параметры:

- msgid – идентификатор очереди;
- cmd – команда (IPC_CMD – удалить очередь, IPC_STAT – заполнить data сведениями об очереди, IPC_SET – установить для очереди параметры из data);
- data – параметры очереди.

Структура блока параметров:

```
struct msgid_ds {
 struct ipc_perm msg_perm; /* Права доступа */
 msgqnum_t msg_qnum;       /* Текущее количество сообщений */
 msglen_t msg_qbytes;      /* Максимально допустимый размер */
 pid_t msg_lspid;          /* ID последнего процесса-отправителя*/
 pid_t msg_lrpid;          /* ID  последнего процесса-читателя */
 time_t msg_stime;         /* Время последней отправки */
 time_t msg_rtime;         /* Время последнего приема */
 time_t msg_ctime;         /* Время последнего изменения параметров */
}
```

4.3.2. Очереди сообщений в стиле POSIX

Эти функции появились в UNIX-системах в связи с требованиями стандарта POSIX. Они присутствуют не во всех UNIX-системах. Для их использования в программе может понадобиться включаемый файл «mqueue.h».

mqd_t mq_open(char *name, int flags [, mode_t perms, struct mq_attr]) – открывает или создает очередь сообщений. Возвращает дескриптор очереди или –1 в случае ошибки. Параметры:

- name – имя очереди;
- flags – битовые флаги доступа (O_CREAT – создать новую, O_EXCL – при создании существующей очереди вернуть ошибку, O_RDONLY – только для чтения, O_WRONLY – только для записи, O_RDWR – для чтения-записи, O_BINARY – в двоичном режиме, O_TEXT – в текстовом режиме, O_NONBLOCK – без блокирования операций чтения при опустошении или записи при переполнении очереди);
- perms – разрешения (S_IWRITE – разрешение записи, S_IREAD – разрешение чтения);
- mq_attr – возможно NULL или в соответствии со структурой:

```
struct mq_attr {
        long mq_flags;      /* Флаги */
        long mq_maxmsg;     /* Максимальное число сообщений */
        long mq_msgsize;    /* Максимальный размер сообщений */
        long mq_msgs;       /* Количество сообщений в очереди */
}
```

int mq_close(mqd_t mqd) – закрывает очередь сообщений. Возвращает 0 в случае успеха или –1 в случае ошибки. Параметр:

- mqd – дескриптор очереди.

int mq_unlink(const char *name) – удаляет очередь сообщений. Возвращает 0 в случае успеха или –1 в случае ошибки. Параметр:

- name – имя удаляемой очереди.

int mq_send(mqd_t mqd, char *msg, size_t msgsize, unsigned priority) – помещает сообщение в очередь в соответствии с приоритетом. Возвращает 0 в случае успеха или –1 в случае ошибки. Параметры:

- mqd - дескриптор очереди;
- msg - сообщение;
- msgsize - размер сообщения;
- priority - приоритет (от 0 до 31).

int mq_receive(mqd_t mqd, char *msg, size_t msgsize, unsigned *priorityp) – извлекает сообщение из очереди. Возвращает размер сообщения в случае успеха или –1 в случае ошибки. Параметры:

- mqd - дескриптор очереди;
- msg - буфер для сообщения;
- msgsize - размер буфера;

- prioritytyp - приоритет принятого сообщения или NULL.

4.4. Разделяемая память

Разделяемая память – фрагмент адресного пространства, к которому имеют доступ несколько процессов одновременно. Доступ к этой области осуществляется при помощи указателей. При программировании используются включаемые файлы «sys/ipc.h» и «sys/shm.h».

4.4.1. Разделяемая память в стиле SYSTEM 5

int shmget(key_t key, size_t size, int flags) – создает общую область. Возвращает идентификатор области в случае успеха или –1 в случае ошибки. Параметры:

- key - токен объекта (см. описание функции ftok() в разделе «Очереди сообщений в стиле SYSTEM V»);
- size - размер вновь создаваемой области;
- flags – флаги (IPC_CREAT – создать новую область, IPC_PRIVATE – защищенная область).

void *shmat(int shmid, const void *shmaddr, int flags) – подключение программы к разделяемой области. Возвращает указатель на начало общей области в случае успеха или –1 в случае ошибки. Параметры:

- shmid – идентификатор общей области;
- shmaddr – желаемый адрес области или NULL;
- flags – флаги.

int shmdt (const void *shmaddr) – отключение программы от разделяемой области. Возвращает 0 в случае успеха или –1 в случае ошибки. Параметр:

- shmaddr – указатель на область.

4.4.2. Разделяемая память в стиле POSIX

Эти функции появились в UNIX-системах в связи с требованиями стандарта POSIX. Они присутствуют не во всех UNIX-системах.

int shm_open(const char *name, int flags, mode_t perms) – открывает объект разделяемой памяти. Возвращает дескриптор области в случае успеха или –1 в случае ошибки. Параметры:

- name – имя области памяти;
- flags – битовые флаги(O_CREAT – создать новый, O_EXCL – при создании существующего объекта вернуть ошибку, O_APPEND – только для добавления, O_RDONLY – только для чтения, O_WRONLY

– только для записи, O_RDWR – для чтения-записи, O_TRUNC – обрезать до 0 длины);
- perms – права доступа (S_IWRITE – разрешение записи, S_IREAD – разрешение чтения).

int shm_unlink(const char name) – удаляет ранее созданный объект разделяемой памяти. Возвращает 0 в случае успеха или –1 в случае ошибки. Параметр:
- name – имя объекта.

int ftruncate(int fd, off_t length) - устанавливает размер объекта, заданного дескрптором. Возвращает 0 в случае успеха или –1 в случае ошибки. Параметры:
- fd – дескритор объекта;
- length – новый размер объекта.

void *nmap(vpid *addr, size_t len, int prot, int flags, int fd, off_t off) – отображает страницы памяти, принадлежащие объекту, в адресное пространство процесса. Возвращает указатель на сегмент или MAP_FAILED в случае ошибки. Параметры:
- addr – желаемый адрес отображения (или NULL);
- len – размер сегмента;
- prot – защита сегмента (PROT_NONE – доступ запрещен, PROT_READ – для чтения, PROT_WRITE – для записи, PROT_EXEC – для исполнения);
- flags – флаги (MAP_SHARED – изменения доступны другим процессам, MAP_PRIVATE – видны только самому процессу);
- fd – дескриптор объекта;
- off – смещение в объекте разделяемой памяти.

int nunmap(vpid *addr, size_t len) – отключает отображение страниц памяти объекта в память процесса. Возвращает 0 в случае успеха или –1 в случае ошибки. Параметры:
- addr – указатель на сегмент;
- len – длина сегмента.

Приложение А. Индивидуальные задания

Написать, отладить и продемонстрировать преподавателю систему из нескольких независимых процессов (**A**) или потоков (**B**), совместно решающих квадратное уравнение (**L**), вычисляющих гипотенузу по двум катетам (**M**) или дисперсию выборки из трех чисел (**N**). Процессы или потоки должны отображать на экране ход своего выполнения в виде отладочных сообщений. Система должны состоять из:

- главного процесса или потока, принимающего с клавиатуры исходные данные и выводящего на экран результат;
- нескольких служебных процессов или потоков, способных по отдельности выполнять элементарные арифметические действия - сложение, вычитание, умножение, деление, вычисление квадратного корня и т.п.

При этом использовать способ синхронизации **C**, **D**, **E** или **F** и способ передачи данных между процессами или потоками - **G**, **H**, **I, J** или **K**.

Условные обозначения:

- **A** – независимые процессы;
- **B** – потоки одного процесса;
- **C** – мьютексы;
- **D** – семафоры;
- **E** – условные переменные;
- **F** – сигналы;
- **G** – обычные файлы;
- **H** – разделяемая память;
- **I** – каналы FIFO;
- **J** – сокеты;
- **K** – очереди сообщений;
- **L** – квадратное уравнение;
- **M** – гипотенуза;
- **N –** дисперсия.

Язык программирования – любой.

Индивидуальные задания

Вариант	Задание	Вариант	Задание	Вариант	Задание
1	ACIL	10	AGJL	19	ACJL
2	ADJM	11	AHKM	20	ADIM
3	BCGN	12	BCKN	21	BCJN
4	AFKL	13	AFGL	22	ADHL
5	ACJM	14	ACHM	23	AFGM
6	BDJN	15	BDGN	24	BDKN
7	ADKL	16	ADIL	25	ACGL
8	AFIM	17	AFJM	26	ADHM
9	BEKN	18	BEJN	27	BEGN

Тесты к зачету по курсу «АРХИТЕКТУРА СОВРЕМЕННЫХ ОПЕРАЦИОННЫХ СИСТЕМ»

1.1Каково основное назначение операционных систем?
А) Обеспечивает интерфейс пользователя с аппаратурой.
Б) Обеспечивает интерфейс прикладных программ с аппаратурой и распределение ресурсов.
В) Обеспечивает интерфейс пользователя с прикладными программами.
Г) Обеспечивает взаимодействие компьютеров друг с другом.

1.2 Может ли ЭВМ, в общем случае, работать без операционной системы
А) Нет
Б) Да, но выполнять прикладные программы не сможет
В) Да, и сможет выполнять прикладные программы

1.3Что такое вычислительные ресурсы?
А) Любые ресурсы ЭВМ, которые могут быть выделены прикладной программе.
Б) Только ресурсы ЭВМ, позволяющие прикладной программе выполнять вычисления.
В) Программные ресурсы, которые могут быть выделены прикладной программе и/или ОС.
Г) Аппаратные ресурсы, которые могут быть выделены прикладной программе и/или ОС.
Д) Любые ресурсы ЭВМ, которые могут быть выделены прикладной программе и/или ОС

1.4Что такое «монолитное ядро» операционной системы?
А) Все менеджеры ресурсов плюс интерфейс доступа к прикладным программам
Б) Все менеджеры ресурсов плюс прикладные программы
В) Всевозможные менеджеры ресурсов, работающие в kernelmode
Г) Всевозможные менеджеры ресурсов вне зависимости от режима работы, плюс драйвера внешних устройств
Д) Всевозможные менеджеры ресурсов, работающие в user mode

1.5 Что такое «микроядро» операционной системы?
А) Совокупность минимально необходимых менеджеров ресурсов, работающих в защищенном режиме
Б) Всевозможные менеджеры ресурсов, оптимизированные по размеру
В) Всевозможные менеджеры ресурсов, оптимизированные по быстродействию
В) Совокупность минимально необходимых менеджеров ресурсов, работающих в user mode

1.6 Каковы достоинства ОС с монолитной архитектурой?
А) Высокая реактивность и масштабируемость

Б) Высокая производительность
В) Эффективная загрузка процессора

1.7 Каковы достоинства ОС с микроядерной архитектурой?
А) Высокая реактивность и масштабируемость
Б) Высокая производительность
В) Эффективная загрузка процессора

1.8. Что такое Linux
А) Операционная система
Б) Ядро, на основе которого строятся конкретные операционные системы
В) Семейство библиотек, обеспечивающих единообразный интерфейс прикладных программ к ядру операционных систем
Г) Семейство стандартов на единообразный интерфейс прикладных программ и пользователей к ядру операционных систем

1.9. Что такое POSIX
А) Операционная система
Б) Ядро, на основе которого строятся конкретные операционные системы
В) Семейство библиотек, обеспечивающих единообразный интерфейс прикладных программ к ядру операционных систем
Г) Семейство стандартов на единообразный интерфейс прикладных программ и пользователей к ядру операционных систем

1.10. Что такое Win API
А) Операционная система
Б) Ядро, на основе которого строятся конкретные операционные системы
В) Семейство библиотек, обеспечивающих единообразный интерфейс прикладных программ к ядру операционных систем
Г) Семейство стандартов на единообразный интерфейс прикладных программ и пользователей к ядру операционных систем

2.1. Каковы режимы работы современных 32-битовых процессоров Intel/Amd ?
а) Реальный и защищенный
б) Реальный, защищенный и упрощенный
в) Реальный, защищенный и режим виртуального 80x86
г) Реальный и режим виртуального 80x86
д) Реальный, защищенный, упрощенный и режим виртуального 80x86

2.2. Каковы разрядности регистров EAX, BX и CL в процессорах семейства Pentium?
а) 32, 24 и 16
б) 24,16 и 8
в) 32, 16 и 8

г) 32, 24 и 4

2.3. Какая машинная команда перешлет содержимое ячейки с адресом 123 в регистр EAX
а) MOV 123, EAX
б) MOV EAX, 123
в) MOV [123], EAX
г) MOV [EAX], 123
д) MOV EAX, [123]
е) MOV [123], EAX

2.4. Что заносится в стек при попытке обработки исключения?
А) Адрес текущей команды
Б) Адрес следующей команды
В) Адрес текущей команды и содержимое только регистра флагов
Г) Адрес следующей команды и содержимое только регистра флагов
Д) Адрес текущей команды и содержимое всех регистров процессора
Е) Адрес следующей команды и содержимое всех регистров процессора

2.5. Когда производится попытка обработки исключения?
А) Немедленно при возникновении, а потом попытка выполнить команду повторяется
Б) После выполнения текущей команды
В) До выполнения текущей команды

2.6. Когда производится попытка обработки прерывания?
А) Немедленно при возникновении, а потом попытка выполнить команду повторяется
Б) После выполнения текущей команды
В) До выполнения текущей команды

2.7. Что заносится в стек при попытке обработки прерывания?
А) Адрес текущей команды
Б) Адрес следующей команды
В) Адрес текущей команды и содержимое только регистра флагов
Г) Адрес следующей команды и содержимое только регистра флагов
Д) Адрес текущей команды и содержимое всех регистров процессора
Е) Адрес следующей команды и содержимое всех регистров процессора

2.8. Какая машинная команда перешлет число 123 в регистр EAX
а) MOV 123, EAX
б) MOV EAX, 123
в) MOV [123], EAX
г) MOV [EAX], 123
д) MOV EAX, [123]

е) MOV [123], EAX

2.9. Какая машинная команда перешлет число 123 в ячейку, адрес которой находится в регистре EAX
а) MOV 123, EAX
б) MOV EAX, 123
в) MOV [123], EAX
г) MOV [EAX], 123
д) MOV EAX, [123]
е) MOV [123], EAX

2.10. Что помещается в стек при выполнении машиной команды INT (генерация прерывания)
а) Адрес возврата (команды, следующей за INT)
б) Адрес возврата (команды INT)
в) Адрес возврата и содержимое регистра флагов
г) Ничего не помещается

3.1. Какой физический адрес памяти соответствует сегментному адресу 1234h:4321h ?
а) 05555
б) 16661
в) 56661
г) 15555

3.2. Где, по умолчанию, расположены вектора прерываний в реальном режиме процессора?
А) В самых младших адресах ОЗУ
Б) В самых старших адресах ОЗУ
В) В этом режиме она отсутствует
Г) В любых адресах памяти

3.3. Из каких частей состоит адрес в защищенном режиме с сегментами?
А) Из селектора и дескриптора
Б) Из селектора и смещения в сегменте
В) Из дескриптора и смещения
Г) Из индекса в таблице страниц, индекса внутри страницы и смещения в странице

3.4. Из каких частей состоит адрес памяти в страничном защищенном режиме?
А) Из селектора и дескриптора
Б) Из индекса в таблице страниц, индекса внутри страницы и смещения в странице
В) Из смещения в сегменте и селектора
Г) Из таблицы страниц и таблицы дескрипторов

3.5. Как осуществляется защита доступа к сегментам в защищенном режиме?
А) При помощи текущего уровня привилегий и битовых полей в адресе и дескрипторе сегмента
Б) При помощи битовых полей доступа в строках таблицы страниц
В) При помощи текущего уровня привилегий и битового поля в дескрипторе сегмента
Г) Не осуществляется совсем

3.6. Как осуществляется защита доступа к страницам в защищенном режиме?
А) При помощи текущего уровня привилегий и битовых полей в адресе и дескрипторе сегмента
Б) При помощи текущего уровня привилегий и битового поля в дескрипторе сегмента
В) При помощи битовых полей доступа в строках таблицы страниц
Г) Не осуществляется совсем

3.7. Как осуществляется защита доступа к памяти в реальном режиме работы процессора?
А) При помощи текущего уровня привилегий и битовых полей в адресе и дескрипторе сегмента
Б) При помощи текущего уровня привилегий и битового поля в дескрипторе сегмента
В) При помощи битовых полей доступа в строках таблицы страниц
Г) Не осуществляется совсем

3.8. Что происходит при попытке процессора, работающего в защищенном режиме, обратиться к несуществующей ячейке памяти
А) Возбуждается исключение
Б) Возбуждается прерывание
В) Просто ничего не происходит

3.9. Что происходит при выполнении процессором машинной команды «INT»
А) Возбуждается исключение
Б) Возбуждается прерывание
В) Просто ничего не происходит

3.10. Что такое «карта ввода-вывода» (iomap)
А) Таблица портов ввода-вывода, при обращении к которым возбуждается исключение
Б) Список устройств ввода-вывода подключенных к компьютеру
Г) Список дескрипторов устройств, доступных прикладной программе

4.1. Каков механизм «виртуальной памяти»?
а) Свопинг страниц в соответствии с состоянием, хранящимся в таблицах ОС
б) Свопинг страниц при исключении после обращения к отсутствующей в ОЗУ странице
в) Свопинг страниц при прерывании от таймера

г) Подкачка страниц с другой ЭВМ по сети

4.2. Что такое «монолитный драйвер»?
а) Драйвер, использованный для построения «монолитного ядра»
б) Драйвер, выполняющий все функции по взаимодействию приложения и устройства
в) Драйвер, исходный текст которого размещается в одном файле
г) Драйвер, предназначенный для работы с реально существующими устройствами

4.3. Что такое «драйвер-фильтр»?
а) Драйвер, который может быть одного из двух типов: «блочный» или «побайтовый»
б) Драйвер, используемый для цифровой фильтрации данных
в) Драйвер, предназначенный для переадресации информационных потоков внутри ОС
г) Драйвер, управляющий устройством аналоговой фильтрации данных

4.4. Что такое «блочный» и «побайтовый» («символьный») драйвера?
А) Драйвера, организованные в виде единого блока или в виде множества байтовых полей
Б) Драйвера для блочного и потокового шифрования данных
В) Драйвера, реализующие типовые стратегии обмена данными
Г) Драйвера для работы с устройствами блочного и потокового шифрования данных

4.5. Что такое «контекст» задачи?
А) Адресное пространство задачи
Б) Список устройств ввода-вывода, которыми пользуется текущая задача
В) Набор значений регистров, указателей стека и прочей служебной информации, связанных с определенной задачей и позволяющей отличить ее от других задач, выполняющихся в том же адресном пространстве

4.6. Что такое синхронный ввод/вывод?
А) Режим в/в, при котором блокируется работа программы до завершения ввода-вывода
Б) Режим в/в, при котором немедленно возвращается признак завершения ввода-вывода
В) В/в через синхронный адаптер

4.7. Что такое «рабочая задача» («рабочий поток», «рабочий процесс») операционной системы?
А) Задача, запускаемая ОС для отложенной обработки прерываний, обнуления памяти, асинхронного в/в и т.п.
Б) Любая задача, в виде которой реализованы компоненты операционной системы
В) Прикладная программа, предназначенная для работы со внешними устройствами
Г) Компонент операционной системы, предназначенный для работы со внешними устройствами

4.8. Что такое асинхронный ввод/вывод?
А) Режим в/в, при котором блокируется работа программы до завершения ввода-вывода
Б) Режим в/в, при котором немедленно возвращается признак завершения ввода-вывода, а сама работа выполняется позже
В) В/в через синхронный адаптер

4.9. Каков механизм виртуализации внешнего устройства в защищенном режиме
А) Работу со внешним устройством обслуживает принадлежащая операционной системе процедура обработки исключения, возбуждаемого при обращении прикладной программы к связанному с устройством порту ввода-вывода
Б) Работу со внешним устройством обслуживает принадлежащая прикладной программе процедура обработки исключения, возбуждаемого при обращении прикладной программы к связанному с устройством порту ввода-вывода
В) Работу со внешним устройством обслуживает прикладная программа, напрямую обращающаяся к портам ввода-вывода

4.10. В каком режиме работы процессора в адресном пространстве задачи может использоваться дисковая память
А) В защищенном с сегментной адресацией
Б) В защищенном режиме со страничной адресацией
В) В реальном режиме
Г) Ни в каком

5.1 Какие из механизмов используются для синхронизации задач?
А) Очереди сообщений
Б) Мьютексы
В) Почтовые ячейки
Г) Каналы

5.2 Какие из механизмов используются для межзадачного обмена?
А) Сигналы
Б) Мьютексы
В) Семафоры
Г) Каналы

5.3 Какова роль функции fork() в ядре UNIX?
А) Открытие файла или устройства
Б) Установка нового значения семафора или мьютекса
В) Проверка значения семафора или мьютекса
Г) Загрузка новой задачи в созданное адресное пространство и запуск ее
Д) Создание копии адресного пространства текущей задачи

5.4 Какова роль функции exec() в ядре UNIX?
А) Открытие файла или устройства
Б) Установка нового значения семафора или мьютекса
В) Проверка значения семафора или мьютекса
Г) Загрузка новой задачи в созданное адресное пространство и запуск ее
Д) Создание копии адресного пространства текущей задачи

5.5 Что такое «клинч» (deadlock) задач
А) Ситуация, когда одной задаче для продолжения работы не хватает ресурсов, принадлежащих другой задаче, а той не хватает ресурсов, принадлежащих первой
Б) Конкурирование нескольких задач за общий вычислительный ресурс
В) Ситуация, когда в вычислительной системе вообще отсутствуют ресурсы, необходимые для выполнения задачам

5.6 Каков метод борьбы операционной системы с «голоданием» задач
А) «Разгон» (boosting) приоритетов
Б) «Наследование» приоритетов
В) Решается только на уровне самих задач (прикладных программ)

5.7 Каков метод борьбы операционной системы с «инверсией» приоритетов
А) «Разгон» (boosting) приоритетов
Б) «Наследование» приоритетов
В) Решается только на уровне самих задач (прикладных программ)

6.1 Что такое кэширование?
А) Сверхбыстрая сортировка данных
Б) Неоднозначное отображение «большого» множества данных на «маленькое» - признаков
В) Упреждающее чтение обрабатываемых данных в сверхбыстрый буфер

6.2 Что такое хеширование?
А) Сверхбыстрая сортировка данных
Б) Неоднозначное отображение «большого» множества данных на «маленькое» - признаков
В) Упреждающее чтение обрабатываемых данных в сверхбыстрый буфер

6.3 Что такое Crypto API
а) Стандартные библиотеки Windows, реализующие основные криптографические алгоритмы
б) Стандартные библиотеки UNIX, реализующие основные криптографические алгоритмы
в) Стандартные библиотеки Windows и UNIX, реализующие только алгоритмы

шифрования

6.4 Где хранятся данные при использовании NTFS?
А) В неслужебной области тома
Б) В метафайле
В) В метафайле и неслужебной области тома
Г) В FAT-таблице

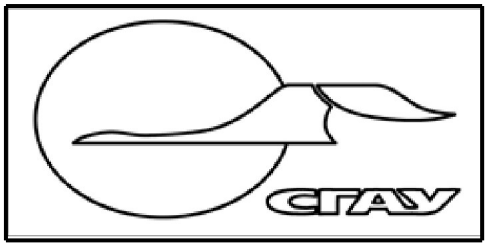
6.5 Где хранятся данные при использовании FAT?
А) В неслужебной области тома
Б) В метафайле
В) В метафайле и неслужебной области тома
Г) В FAT-таблице

6.6. Что такое «дисковый кластер»
А) Цепочка из одного или нескольких дисковых секторов, разбросанных по диску, но используемых как единица хранения данных
Б) Один или несколько соседствующих дисковых секторов, используемых как единица хранения данных
В) Весь набор дисковых секторов, занимаемых определенным файлом

6.7. Что такое FAT
А) Таблица, содержащая характеристики логической структуры диска
Б) Таблица, хранящая номера кластеров и описывающая их цепочки, принадлежащие тем или иным файлам
В) Таблица, хранящая описания имен файлов, их атрибутов и списки используемых ими кластеров

6.8. Что такое «метафайл»
А) Таблица, содержащая характеристики логической структуры диска
Б) Таблица, хранящая номера кластеров и описывающая их цепочки, принадлежащие тем или иным файлам
В) Таблица, хранящая описания имен файлов, их атрибутов и списки используемых ими кластеров и, возможно, данные

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
«САМАРСКИЙ ГОСУДАРСТВЕННЫЙ АЭРОКОСМИЧЕСКИЙ
УНИВЕРСИТЕТ ИМЕНИ АКАДЕМИКА С.П. КОРОЛЕВА
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)»
(СГАУ)



СОГЛАСОВАНО

Управление образовательных программ

_____/ А.В. Дорошин /

"_____"_____ 20___ г.

УТВЕРЖДАЮ

Проректор по учебной работе

_____/ В.Н. Матвеев /

"____" _____ 20___ г.

РАБОЧАЯ ПРОГРАММА ДИСЦИПЛИНЫ

Наименование модуля (дисциплины)

М2.В.ОД.1. - Архитектура современных операционных систем

Цикл, в рамках которого происходит освоение модуля (дисциплины)	Профессиональный цикл
Часть цикла	М2.В.ОД - обязательные дисциплины

Код учебного плана

230100.68.2-13-56-3022

Факультет	Информатики
Кафедра	Инф. систем и технологий

Курс	5		
Семестр	9		
Лекции (СЛ)	18		
Семинарские и практические занятия (СП)	0		
Лабораторные занятия (СЛР)	36	Экзамен	
Контроль самостоятельной работы / Индивидуальные занятия (КСР / ИЗ)	0	Зачет	9
Самостоятельная работа (СРС)	54		
Всего (Всего с экзаменами)	108		

Наименование стандарта, на основании которого составлена рабочая программа:

230100.68 "Информатика и выч. техника"

Соответствие содержания рабочей программы, условий ее реализации, материально-технической и учебно-методической обеспеченности учебного процесса по дисциплине всем требованиям государственных стандартов подтверждаем.

Составители:

К.Е. Климентьев _____
(подпись)

Заведующий кафедрой:

С.А. Прохоров _____
(подпись)

Рабочая программа обсуждена на заседании кафедры

Инф. систем и технологий

Протокол №___ от "____" _____ 20___ г.

Наличие основной литературы в фондах научно-технической библиотеки (НТБ) подтверждаем:

Директор НТБ _____ /_____/
(подпись) (расшифровка подписи)

Согласовано:

Декан _____ /_____/
(подпись) (расшифровка подписи)

1 Цели и задачи модуля (дисциплины), требования к уровню освоения содержания

1.1 Перечень развиваемых компетенций

ОК-12, ПК-9, ПК-11.

1.2 Цели и задачи изучения модуля (дисциплины)

Целью дисциплины является выработка у студентов навыков использования современных системных программных средств (операционных систем, операционных оболочек, обслуживающих сервисных программ), а также ознакомление с проблемами и направлениями развития системных программных средств; формирование представлений об основных принципах организации операционных систем - прежде всего, предназначенных для мобильных устройств.

1.3 Требования к уровню подготовки студента, завершившего изучение данного модуля (дисциплины)

После окончания изучения курса студент должен знать: аппаратные особенности вычислительных устройств, ориентированных на современные опекрационные системы; принципы построения современных операционных систем; интерфейсы программирования современых опера
ционных систем.
Уметь: использовать возможности современных операционных систем как на уровне пользователя, так и с точки зрения разработчика прикладного и системного программного обеспечения.

1.4 Связь с предшествующими модулями (дисциплинами)

Изложение материала базируется на знании студентами следующих дисциплин:
- программирование на ЭВМ;
- операционные системы;

1.5 Связь с последующими модулями (дисциплинами)

Дисциплина является предшествующей для выполнения квалификационной работы магистра.

2 Содержание рабочей программы (модуля)

Семестр 1		
СЛ 0,1667 18 часов 0,5001 ЗЕТ	Активные 0	
	Интерактивные 0	

	Традиционные 0	1. Основные понятия операционных систем – определение, назначение, решаемые задачи, классификация. История операционых систем от Microsoft – MS-DOS, ветви Windows 9X и Windows NT. История операционных ситем Unix-совместимых операционных систем. – 2 часа.
		2. Архитектура Windows. Программные интерфейсы Win API. Обзор системных возможностей. Работа с консолью. – 2 часа.
		3. Архитектура UNIX. Прграммные интерфейсы LibC и др. Обзор системных возможностей. Работа с консолью, скрипты – 2 часа.
		4. Стандарт POSIX. Назначение, история, особенности. Средства ввода-вывода в стандрте POSIX. – 2 часа.
		5.Многозадачность в Unix. Потоки и процессы. Содание, удаление, управление. – 2 часа.
		6.Средства синхронизации задач в Unix. – 2 часа.
		7.Средства межзадачного обмена в Unix. – 2 часа.
		8. Современный взгляд на архитектуру операционных систем. Виртуальные машины. Java-машина. – 2 часа.
		9.Аппаратура мобильных устойств. Мобильные операционные системы: Symbian, iOS, Android. – 2 часа.
СП 0 0 часов 0 ЗЕТ	Активные 0	
	Интерактивные 0	
	Традиционные 0	
СЛР 0,3333 36 часов 0,9999 ЗЕТ	Активные 0	
	Интерактивные 0	исследование работы с консолью в Unix. - 6 часов.
		исследование средств многозадачности в Unix. – 12 часов.
	Традиционные 0	

КСР 0 0 часов 0 ЗЕТ	Активные 0	
	Интерактивные 0	
	Традиционные 0	
СРС 0,5 54 часов 1,5 ЗЕТ	Активные 0	
	Интерактивные 0	
	Традиционные 0	

3 Инновационные методы обучения

Лабораторные работы выполняются с использованием вычислительной техники

4 Технические средства и материальное обеспечение учебного процесса

Лабораторные работы выполняются средствами вычислительной техники.
Требования к аппаратному обеспечению: наличие локальной сети.
Требования к программному обеспечению:
- операционная система Windows версий NT, 2000 или XP;
- UNIX-совместимая операционная система (например, FreeBSD или на основе ядра Linux) ;
- бесплатный компилятор GCC C/C++ любой версии.

5 Учебно-методическое обеспечение

5.1 Основная литература

1. Робачевский, Андрей М.. Операционная система UNIX [Текст] : Учеб.пособие для вузов / Андрей Робачевский. - СПб. : БХВ-Петербург, 2002. - 514 с. Экземпляров всего:16
2. Керниган, Брайан В.. UNIX. Программное окружение [Текст] : [пер. с англ.] / Брайан Керниган, Роб Пайк. - СПб. ; М. : Символ, 2003. - 414 с. Экземпляров всего:5

5.2 Дополнительная литература

1. Бэкон, Джин. Операционные системы [Текст] : парал. и распредел. системы : [пер. с англ.] / Джин Бэкон, Тим Харрис. - СПб. [и др.] : Питер [и др.], 2004. - 799 с. Экземпляров всего:8
2. Таненбаум, Эндрю. Современные операционные системы [Текст] / Э. Таненбаум. - 2-е изд. - СПб. [и др.] : Питер : Питер принт, 2005. - 1037 с. Экземпляров всего:5
3. Гласс, Грэм. UNIX для программистов и пользователей [Текст] : [пер. с англ.] / Г. Гласс, К. Эйблс. - СПб. : БХВ-Петербург, 2004. - XXVIII, 820 с. Экземпляров всего:5

5.3 Электронные источники и интернет ресурсы

Материалы сайта: http://developer.apple.com

5.4 Методические указания и рекомендации

Целью лабораторной работы №2 является разработка и отладка программы, использующей различные методы управления задачами, синхронизации задач и межзадачного обмена в среде UNIX-подобной операционной системы.
Рекомендуемый компилятор: GCC C/C++.