Процессы

- *Программа — это статическая последовательность команд (инструкций),* описывающих решение определенной задачи.

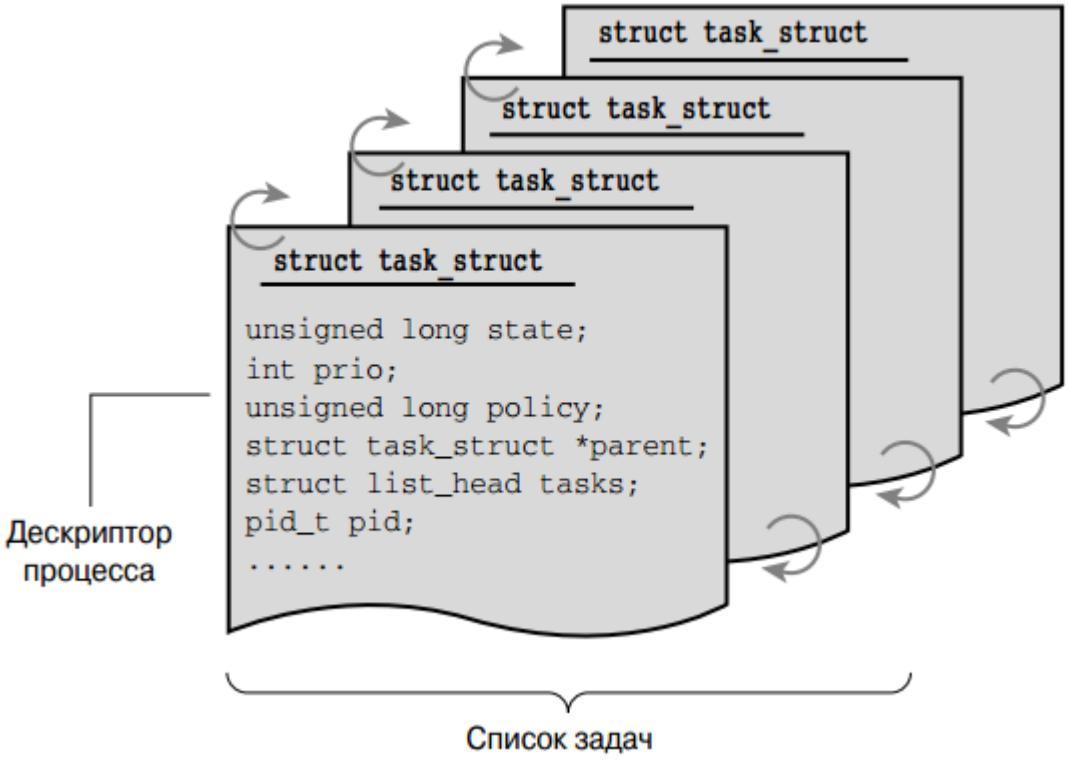
- **Процесс** – это система действий, реализующая выполнение программы в компьютерной системе.
- **Процесс** — это контейнер для набора ресурсов, используемых при выполнении экземпляра программы.
- **Проце́сс** — это идентифицируемая абстракция совокупности взаимосвязанных системных ресурсов на основе отдельного и независимого виртуального адресного пространства в контексте которой организуется выполнение потоков.

Как хранится информация о процессах ?

- При управлении процессами операционная система создает три информационные структуры :
  - 1) дескриптор процесса (системный контекст) ;
  - 2) регистровый контекст;
  - 3) пользовательский контекст.

Системный контекст + регистровый контекст = Блок управления процессом (process control block) PCB.

Системный контекст и структура ядра



Где хранится системный контекст?

- **Системный контекст :**
  - В Linux – структура ***task_struc***t
  - В Windows **Объект - процесс EPROCESS**

Системный контекст хранится в области ядра и доступен только ядру, приложение не может самостоятельно напрямую модифицировать его.

Что входит в системный контекст

- Информация, которая необходима ОС в течение всего жизненного цикла процесса:
  - время запуска;
  - идентификатор пользователя, создавшего процесс;
  - состояние процесса;
  - расположение процесса в оперативной памяти и на диске
  - приоритет процесса;
  - используемое процессорное время;
  - информация о родственных процессах;
  - параметры планирования;
  - системный стек процесса
  - указатели на открытые процессом файлы;
  - информация об операциях ввода-вывода, используемая процессом.

Регистровый контекст

- ***Регистровый контекст :***
  - Сохраняется текущее состояние регистров процессора каждый раз, когда ОС прерывает выполнение процесса
  - Извлекается из регистрового контекста обратно в регистры процессора, когда ОС возобновляет выполнение процесса.

Управляющий блок процесса

- PCB – (process control block)=Системный контекст

  +

  Регистровый контекст

Пользовательский контекст

- ***В пользовательском контексте хранится :***
- код программы процесса
- данные программы процесса
- пользовательский стек процесса
- общая совместная память используемая процессами

Пользовательский контекст процесса хранится в пользовательской области памяти процесса и перемещается при необходимости вместе с ним.

Образ процесса в памяти (все вместе)

В образ процесса в памяти входит следующая информация.

1. Команды программы

2. Данные программы

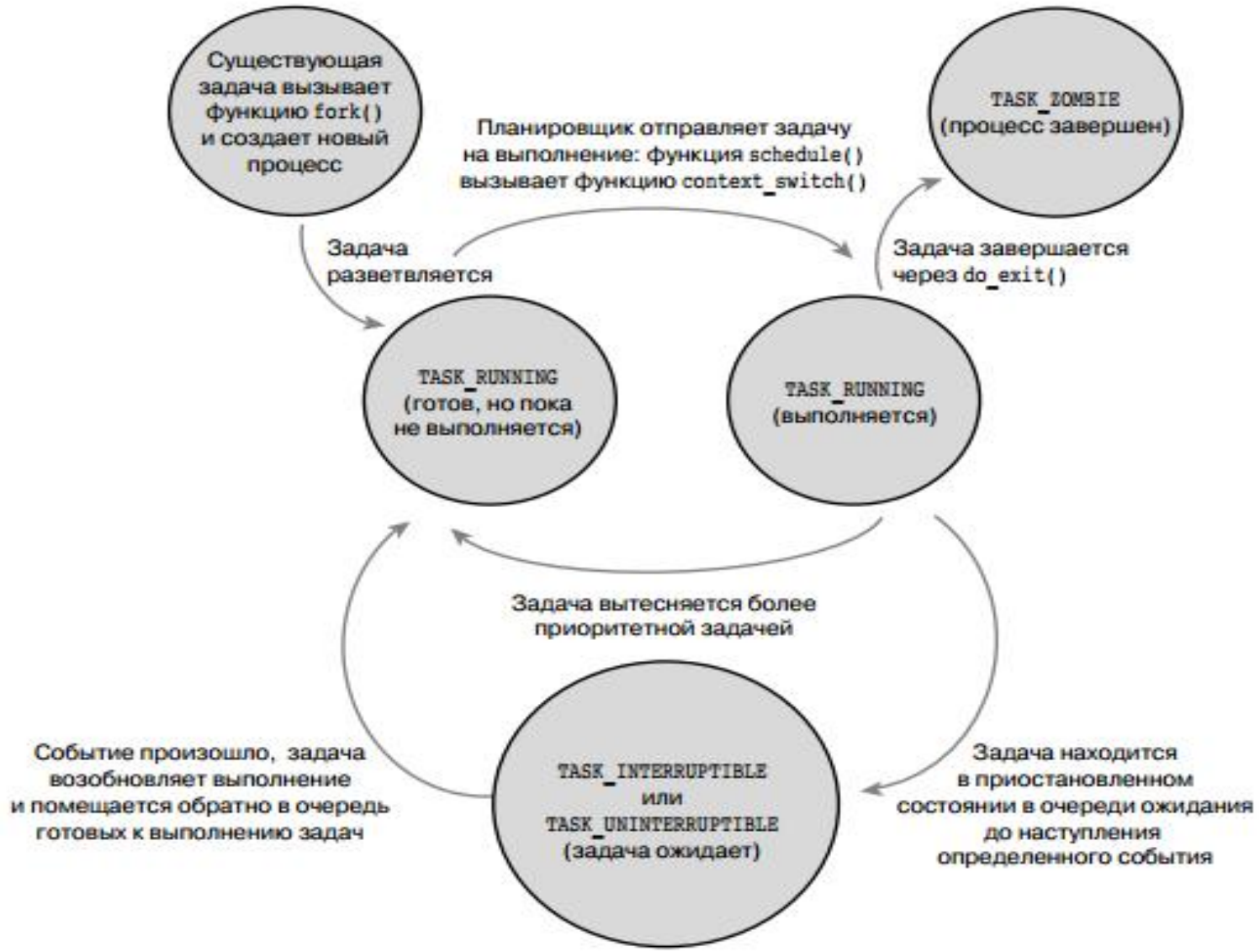
3. Пользовательский стек.

4. Heap – Куча
5. Общая разделяемая память
6. Блок управления процессом
   - Системный контекст
   - Регистровый контекст

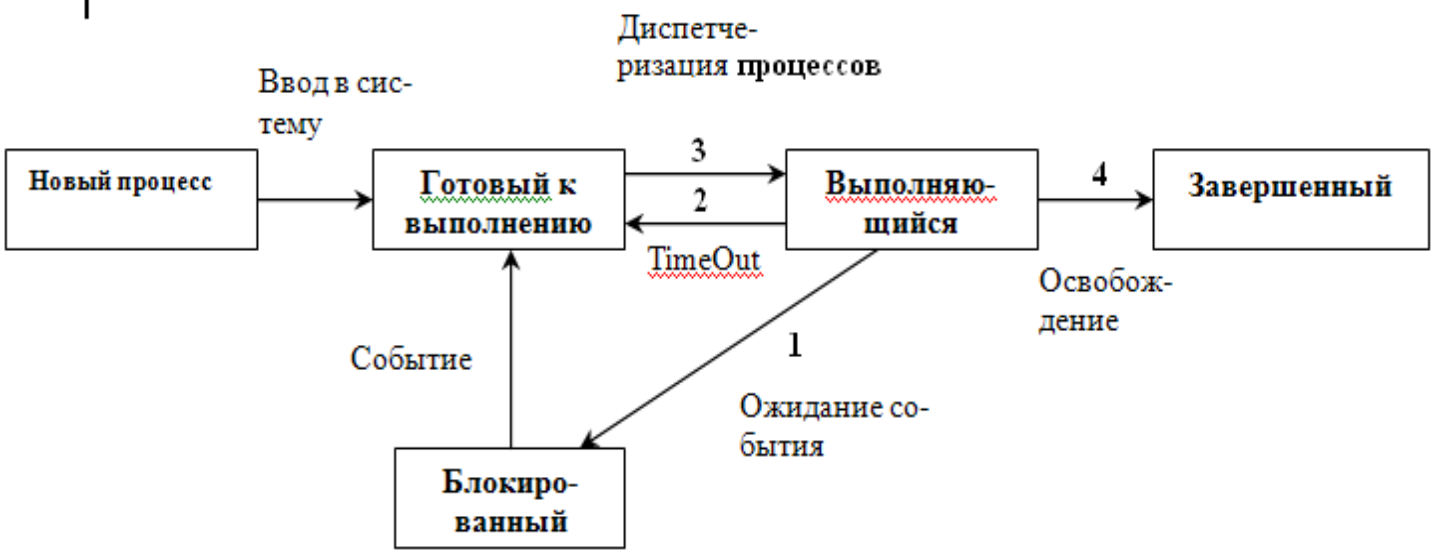
7. Системный стек процесса.

Размещение образа процесса в памяти

Диаграмма процесса



Модель состояний процесса



- *Новый процесс*. Только что созданный процесс, информация о процессе помещена ОС во множество (таблицу) процессов, но процесс не загружен в оперативную память .
- *Готовый к выполнению.* Процесс загружен в память и будет запущен, как только представится возможность.
- *Выполняющийся.* Процесс, выполняющийся процессором в данный момент.
- *Блокированный.* Процесс, который ожидает некоторого события.
- *Завершающийся.* Процесс, удаленный из множества запущенных процессов.

Функции ОС по управлению процессами

а) Создание и завершение;

б) Планирование и диспетчеризация;

в) Переключение;

г) Синхронизация(взаимодействие);

Планировщик(диспетчер) процессов - та часть операционной системы, которая занимается планированием процессов.

Создание и завершение процессов

Последовательность создания процесса

1. Создать и проиниацилизировать блок управления процессом PCB
2. Присвоить новому процессу уникальный идентификатор (занести новую запись в таблицу процессов);
3. Выделить адресное пространство для образа процесса;
4. Загрузить часть команд и данных процесса в оперативную память.
5. Поместить процесс в очередь “готовых” процессов;

Завершение процесса

- 1. Обычное завершение (добровольно).
- 2. Превышение процессом времени, отведенной для его выполнение (принудительно).
- 3. Недостаточный объем памяти (принудительно).
- 4. Ошибки в самом процессе (коде программы, - деление на ноль, переполнение разрядной сетки; неверная команда и.т.д.) (принудительно).
- 5. Ошибки ввода-вывода (принудительно).
- 6. Завершение другим процессом (принудительно).
- 7. Вмешательство оператора (принудительно).

Создание новых процессов:

1. Загрузка системы
   - При загрузке ОС создаются несколько начальных процессов.В UNIX/linux процессы Sched(pid0) и init(pid1) в Windows - System(Pid4).

2. Начальные процессы создают другие процессы :
   - одна часть из них взаимодействуют с пользователем;
   - вторая часть является фоновыми процессами (в Linux - демоны) не связана с пользователем и выполняет системные функции.

3. Запрос пользователя на создание нового процесса.

Пример :
- Пользователь вызывает команду из текстовой оболочки или графической оболочки. ОС создает новый процесс, родитель которого – оболочка ОС.

Для создания процесса необходимо обратиться к ядру ОС с помощью специального системного вызова и попросить её создать процесс.

Один процесс создает другие процессы.

Системные вызовы для создания процесса в UNIX

- В UNIX/Linux в два этапа:

  - Системный вызов ***fork ()*** создает точную копию родительского процесса.
    - Точна копия позволяет дочернему процессу наследовать уже созданное окружение родительского процесса (открытые файлы, устройства ввода вывода и.т.д.)
  - Системный вызов **exec()** запускает на выполнение программу в созданном системным вызовом fork() процессе
    - Пример вызова ***execl("/home/user/test",NULL,NULL);***

Системный вызов для создания процесса в Linux

```
#include <stdio.h>
#include <unistd.h>
#include<sys/types.h>
int  main ()
{
pid_t  pid;      /* Pid_t тип данных для ID процесса*/
printf ("Пока всего один процесс\n");
pid = fork ();     /*Создание нового процесса */
printf ("Уже два  процесса\n");
if (pid = = 0){
                printf ("Это Дочерний процесс его pid = %d\n", getpid());
                printf ("А pid его Родительского процесса=%d\n", getppid());
    // Сюда можно загружать исполняемый файл
    execl("/home/user/test",NULL,NULL);
}
elseif (pid> 0)
                printf ("Это Родительский процессpid=%d\n", getpid());
else
printf ("Ошибка вызова fork, потомок не создан\n");
}
```

Системные вызовы wait() и waitpid().

- **wait()**
  - Приостанавливает родительский процесс до тех пор, пока его дочерние процессы не будут остановлены или не завершены, после чего возвращает информацию о том, какой процесс завершился и что стало причиной его завершения.
- **waitpid()**
  - Дожидается выполнения конкретного процесса
- **Функция exit()**
- void exit(int status) прекращает процесс, из которого эта функция была вызвана.

Планирование процессов

Планирование процессов

Управление процессами

- Для организации многозадачности ОС осуществляет планирование процессов:
- Долгосрочное планирование – определяет будет ли создан новый процесс.
- Среднесрочное планирование определяет какой из процессов будет загружен в память для выполнения вместо выгруженного на диск процесса (переход новый – готовый).
- Краткосрочное планирование - какой процесс будет выполняться на процессоре.(переход готовый – выполняющийся)
-

Краткосрочное планирование процессов

- При краткосрочном планировании решаются задачи:
  - 1. Когда выбирать процесс на исполнение его процессором ;
  - 2. Какой процесс выбирать;
  - 3. Переключение контекстов "старого" и "нового" процессов.

Алгоритмы краткосрочного планирования

- Делятся на две группы
  - **вытесняющие алгоритмы** (выполнение процесса может прерываться);
  - **не вытесняющие алгоритмы** (выполнение процесса не может прерываться).

Вытесняющие алгоритмы планирования

- Выполняющийся в настоящий момент процесс может быть прерван и переведен операционной системой в состояние готовности к выполнению.

- Планирование возлагается на операционную систему.

Не вытесняющие алгоритмы.

- Процесс выполняется до тех пор, пока он сам, не отдаст управление диспетчеру процессов ОС для запуска на выполнение другого процесса.

- Планирование возлагается на программу (программиста, коотрый создает программу) и операционную систему.

Вытесняющие алгоритмы планирования

Алгоритмы планирования
Квантование и приоритеты

- Наиболее часто встречающиеся алгоритмы :
  - алгоритмы, основанные на *квантовании (когда выбирать)*
  - алгоритмы, основанные на *приоритетах (кого выбирать)*.

Алгоритм квантования

- **Циклический алгоритм. Круговое планирование**
- Таймер генерирует прерывания через определенные кванты времени.
- Каждый процесс выполняется в течении кванта времени.
- При каждом прерывании от таймера исполняющийся в настоящий момент процесс прерывается и помещается в очередь готовых к выполнению процессов, а вместо него начинает выполняться следующий процесс.

Приоритеты

- *Приоритет* - это число, характеризующее степень привилегированности процесса при использовании ресурсов вычислительной системы (чем выше приоритет, тем выше привилегии).

- Приоритет может выражаться целыми или дробными, положительным или отрицательным значениями.

- Приоритет назначается самой ОС по определенным правилам.

- Приоритет может оставаться фиксированным на протяжении всей жизни процесса либо изменяться во времени.

Алгоритмы на основе приоритетов

- 1**Первым поступил – первым обслужен**
- «Первым поступил — Первым обслужен" (first-come-first-served – FCFS)
  - Для выполнения выбирается процесс, который находился в очереди дольше других.

Алгоритмы на основе приоритетов

- **Выбор самого короткого процесса**
- Для выполнения выбирается процесс с наименьшим ожидаемым временем исполнения.

- Основная трудность – предварительно оценить время выполнения каждого процесса.

Переключение процессов

Переключение процессов

- Пример механизма переключений:
- 1. По истечению кванта времени выполнения процесса процессором, возникает прерывание от таймера по которому выполнение процесса приостанавливается, содержимое регистров процессора (счетчик команд, слово состояния и др.) сохраняется в регистровом контексте процесса в ядре.
- Планировщик процессов выбирает нужный, ранее прерванный процесс (*согласно алгоритма планирования*). В регистры процессора из регистрового контекста этого процесса загружается сохраненное ранее содержимое регистров процессора.
- Выполнение процесса возобновляется с команды программы, следующей после последней команды, выполненной процессором на предыдущем цикле выполнения процесса.

Взаимодействие процессов

Взаимодействие процессов

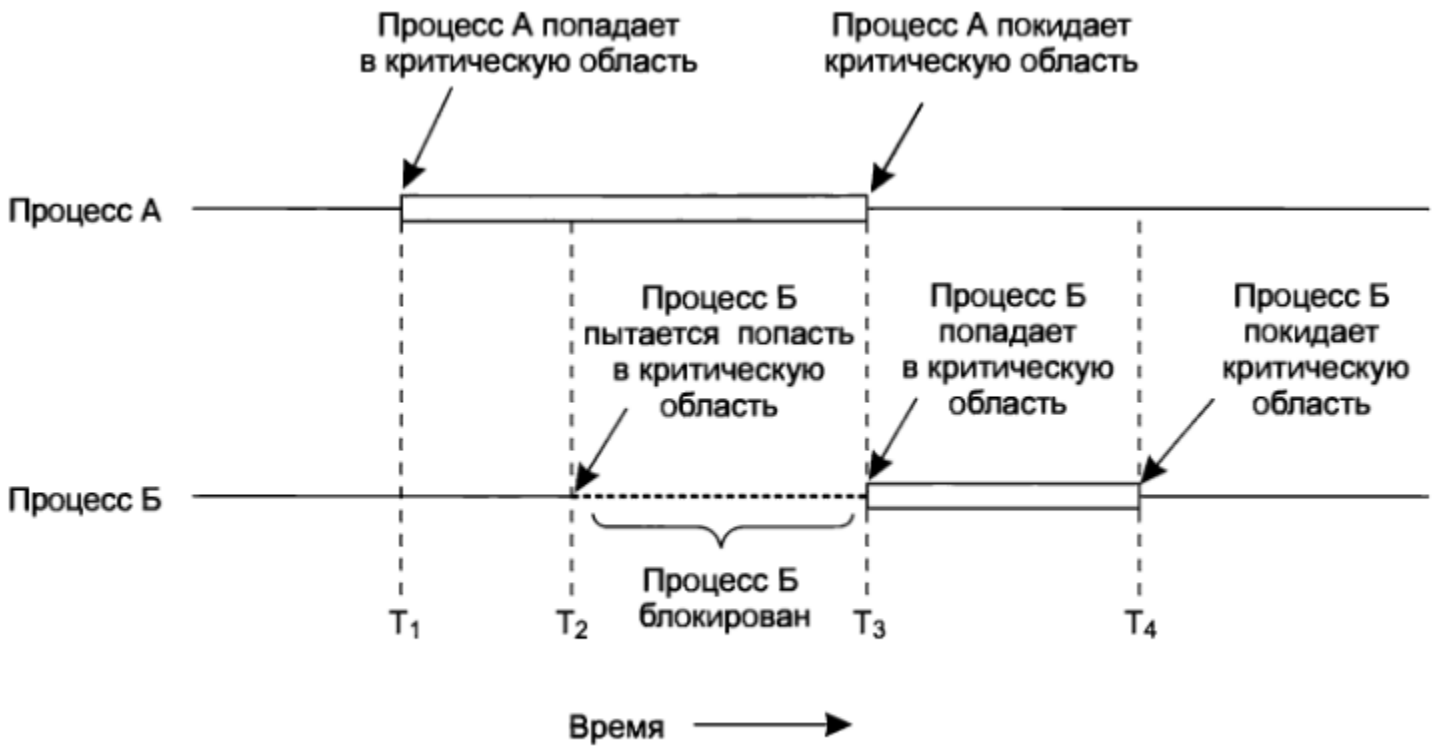
- При взаимодействии процессов необходимо решать несколько задач:
  - 1) Синхронизация процессов. Совместная работа без создания помех (**взаимоисключения)**;
  - 2) Передача данных от одного процесса другому (межпроцессное взаимодействие);

Критические секции

- **Критическая область или критическая секция** - **часть программы**, в которой используется доступ к **общим ресурсам** (общей памяти, общим переменным, общим файлам и.т.д.).

- Для корректной работы необходимо, что бы ни какие два процесса не находились одновременно в своих критических секциях, т.е. взаимоисключали друг друга.

Взаимное исключение с использованием критических областей



Взаимное исключение использования критических областей

Взаимное исключение

- Для задания режима взаимного исключения используются различные механизмы:
- 1) запрещение прерываний. В этом случае запрещаются прерывания по таймеру. И процессор не может переключится по таймеру на другой процесс.(недостаток – запрещение прерываний может привести к краху всей системы)
-

Взаимное исключение

- ***Блокирующие переменные (БП)***
- Используется одна общая (видимая всем процессам) переменная, которая назначается разделяемому ресурсу и ей присваивается значение 0 (ресурс свободен)
- Когда процессу требуется войти в свою критическую область, он проверяет значение блокирующей переменной.
- Если значение БП равно **0**, процесс устанавливает его в **1** и после этого входит в критическую секцию(**осуществляет операции обращения к ресурсу**).
- Другие процессы опрашивают значение БП и ждут пока оно не станет равным **0** (т.е. пока ресурс не освободиться).

Недостаток блокирующих переменных

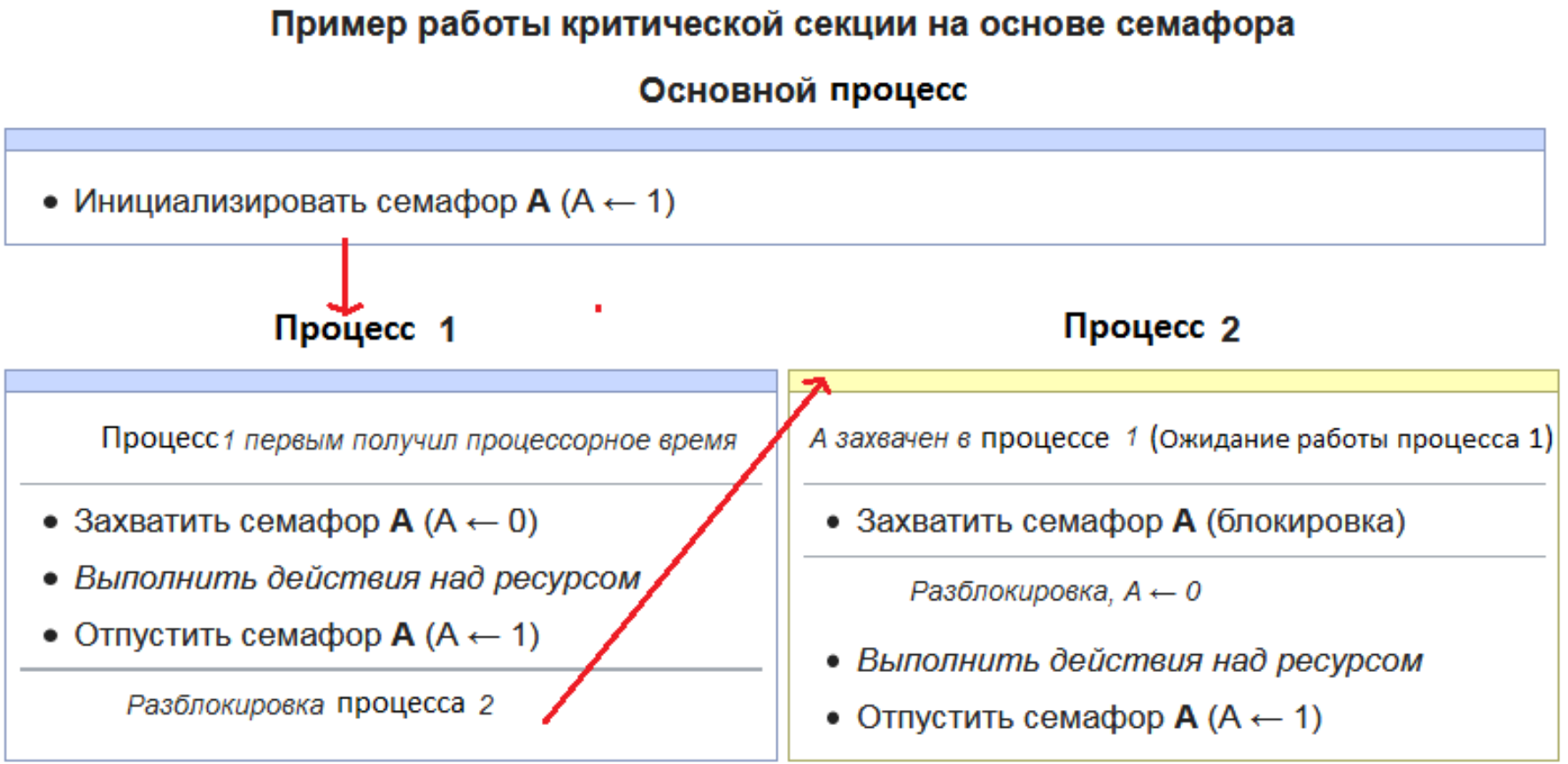
- Отсутствие атомарности (неделимости)

- Операция изменения значения блокирующей переменной раскладывается компилятором на несколько машинных инструкций.
- Может возникнуть такая ситуация, когда процесс P1 при захвате блокирующей переменной (привязана к общему файлу) не успевает записать её обновленное значение в память и прерывается ОС, думая, что он владеет файлом.
- Новый процесс P2 тоже захватывает эту же переменную устанавливает её в единицу и пишет в общий файл.
- P1 при возобновлении выполнения записывает значение блокирующей переменной ,равное 1 в память (хотя значение переменной уже установлено в 1 процессом P2). P1 пишет в тот же файл, затирая то, что записал P2

Семафор

- Семафор – неотрицательная целая переменная, над которой возможны операции:
- Инициализация семафора (*задать начальное значение семафора = 1 – ресурс свободен*):
- Захват семафора (*установить семафор в 0 войти в критическую секцию и занять ресурс*)
- Освобождение семафора (*выйти из критической секции, освободить ресурс, установить семафор в 1*).
- Операции захвата и освобождения – атомарные (не делимые) не могут прерываться со стороны ОС
  - **Бывает:**
  - **бинарный семафор – может принимать два значения (занят/свободен)**

Использование бинарного семафора



Семафор счетчик

- **Семафор счетчик –** назначается ресурсу, который может принимать несколько значений из множества значений (например, количество записанных блоков в файл.)
- Пример:
  - Один процесс захватывает семафор и записывает блоки в файл увеличивая значение семафора на 1).
  - Другой процесс так же может захватить семафор для чтения и удаления блока из файла, при этом уменьшая значение семафора)

Взаимодействие процессов

Межпроцессные взаимодействия

Необходимость взаимодействия процессов

- Совместное использование общих данных .

- Удобства работы пользователя, желающего, например, редактировать и отлаживать программу одновременно. В этой ситуации процессы редактора и отладчика должны уметь взаимодействовать друг с другом.

Межпроцессное взаимодействие

- Процессы *выполняются в раздельных адресных пространствах. Для организации межпроцессного взаимодействия существуют специальные методы:*

- 1) общие файлы;
- *2) разделяемую память*
  3) сигналы (signal);
- *4) каналы (pipe);*
- 5) семафоры.
- 6)Сокеты (Sockets)

- 1. **Общие файлы**. Оба процесса открывают один и тот же файл, с помощью которого обмениваются информацией.

Межпроцессное взаимодействие

- 2. ***Разделяемая память*** специальная область памяти, позволяющей иметь к ней доступ нескольким процессам.

Межпроцессное взаимодействие

- 3. ***Сигналы*** **-** уведомление процесса о каком-либо событии. Когда сигнал послан процессу, операционная система прерывает выполнение процесса и управление передается функции-обработчику сигнала.

- По окончании обработки сигнала процесс вновь запускается на исполнение.

- В Unix для передачи сигналов используется системный вызов *kill* .

Межпроцессное взаимодействие

- 4 Каналы. Программный канал — это файл особого типа (FIFO: «первым вошел - первым вышел»). Процессы могут записывать и считывать данные из канала как из обычного файла.
-
- Если канал заполнен, процесс записи в канал останавливается до тех пор, пока не появится свободное место, чтобы снова заполнить его данными.

- Если канал пуст, то читающий процесс останавливается до тех пор, пока пишущий процесс не запишет данные в этот канал.

Межпроцессное взаимодействие

- 5 Семафор - переменная определенного типа, которая доступна параллельным процессам.

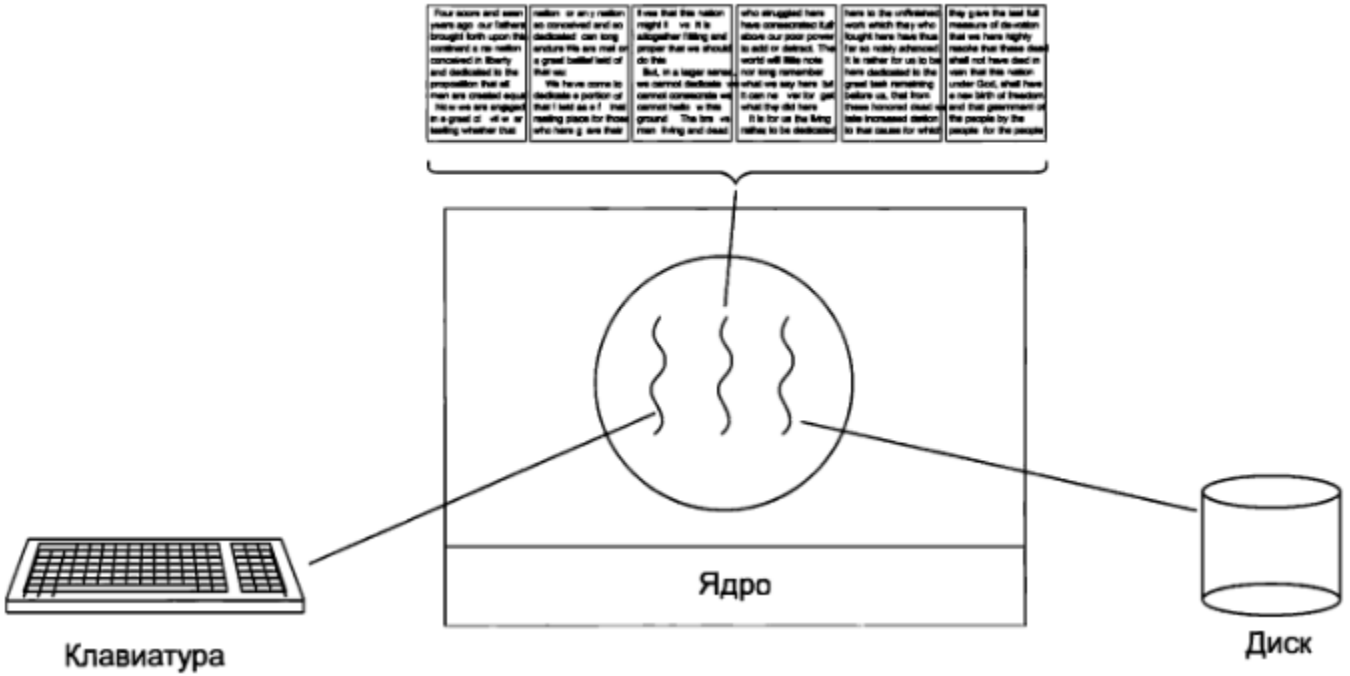
ПОТОКИ

Потоки

- Основная задача процессов – обеспечение режима многозадачности.

- Для повышения эффективности выполнения процесса и загрузки процессора, процесс разбивается на более мелкие смысловые части называемые **ПОТОКАМИ**.
- Потоки по очереди выполняются процессором под управлением ОС, в рамках одного процесса.
- Потоки повышают производительность использования процессора.
- Потоки – единица планирования процессора.

Пример многопоточного приложения



Текстовый процессор использующий три потока

– поток чтения данных с клавиатуры;

– поток вывода на экран ;

– поток сохранения на диск

Потоки

Приложения | Процессы | Службы | Быстродействие | Сеть | Пользователи

Имя образа	Пользо...	ЦП	Память (...	Выделенная память	Базовый при...	Счетчик потоков	Описание
svchost.exe	NETWO...	00	1 680 КБ	2 100 КБ	Средний	5	Хост-процесс для служб Wind...
svchost.exe	LOCAL ...	00	7 152 КБ	9 324 КБ	Средний	23	Хост-процесс для служб Wind...
System	SYSTEM	00	512 КБ	584 КБ	Средний	146	NT Kernel & System
SystemWebS...	SYSTEM	00	4 604 КБ	5 720 КБ	Средний	9	System Web Server Daemon
TabTip.exe	Adminis...	00	4 656 КБ	6 568 КБ	Высокий	17	Tablet PC Input Panel Accessory
TabTip32.exe...	Adminis...	00	668 КБ	824 КБ	Средний	1	Tablet PC Input Panel Helper
taskeng.exe	Adminis...	00	2 248 КБ	2 676 КБ	Средний	7	Обработчик планировщика за...
taskhost.exe	Adminis...	00	3 000 КБ	8 500 КБ	Средний	9	Хост-процесс для задач Wind...
taskhost.exe	Adminis...	00	4 536 КБ	7 808 КБ	Средний	6	Хост-процесс для задач Wind...
taskmgr.exe	Adminis...	01	3 352 КБ	3 724 КБ	Высокий	6	Диспетчер задач Windows
TOTALCMD64...	Adminis...	00	12 376 КБ	18 676 КБ	Средний	13	Total Commander
USBChargerPl...	Adminis...	00	696 КБ	2 272 КБ	Ниже средн...	3	ASUS USB Charger Plus
USBGuard.ex...	Adminis...	00	7 648 КБ	8 688 КБ	Средний	7	USB Disk Security
vmnat.exe *32	SYSTEM	00	1 360 КБ	1 676 КБ	Средний	6	VMware NAT Service
vmnetdhcp.e...	SYSTEM	00	7 372 КБ	7 600 КБ	Средний	3	VMware VMnet DHCP service
vmware-auth...	SYSTEM	00	3 000 КБ	5 076 КБ	Средний	6	VMware Authorization Service
vmware-host...	SYSTEM	00	28 068 КБ	37 108 КБ	Средний	22	vmware-hostd
vmware-usba...	SYSTEM	00	2 476 КБ	3 756 КБ	Средний	5	VMware USB Arbitration Service
wcourier.exe ...	Adminis...	00	5 996 КБ	6 788 КБ	Средний	2	A program that manage wireles...
WDC.exe *32	SYSTEM	00	1 168 КБ	1 432 КБ	Средний	1	WDC
wininit.exe	SYSTEM	00	1 348 КБ	1 672 КБ	Высокий	3	Автозагрузка приложений Wi...
winlogon.exe	SYSTEM	00	2 696 КБ	3 184 КБ	Высокий	3	Программа входа в систему W...
WINWORD.EXE	Adminis...	00	39 468 КБ	49 564 КБ	Средний	12	Microsoft Word
wisptis.exe	SYSTEM	00	3 152 КБ	3 456 КБ	Высокий	5	Компонент пера и сенсорного ...
wisptis.exe	Adminis...	00	4 168 КБ	4 680 КБ	Высокий	10	Компонент пера и сенсорного ...
WmiPrvSE.exe	SYSTEM	00	2 676 КБ	3 368 КБ	Средний	7	WMI Provider Host

Потоки

- Каждый поток может иметь доступ к любому адресу памяти в пределах адресного пространства процесса
- Защита памяти между потоками одного процесса отсутствует:
  - один поток может считывать и записывать данные из другого потока процесса
- Каждый поток может использовать одни и те же открытые файлы процесса и устройства ввода-вывода

Необходимость создания потоков

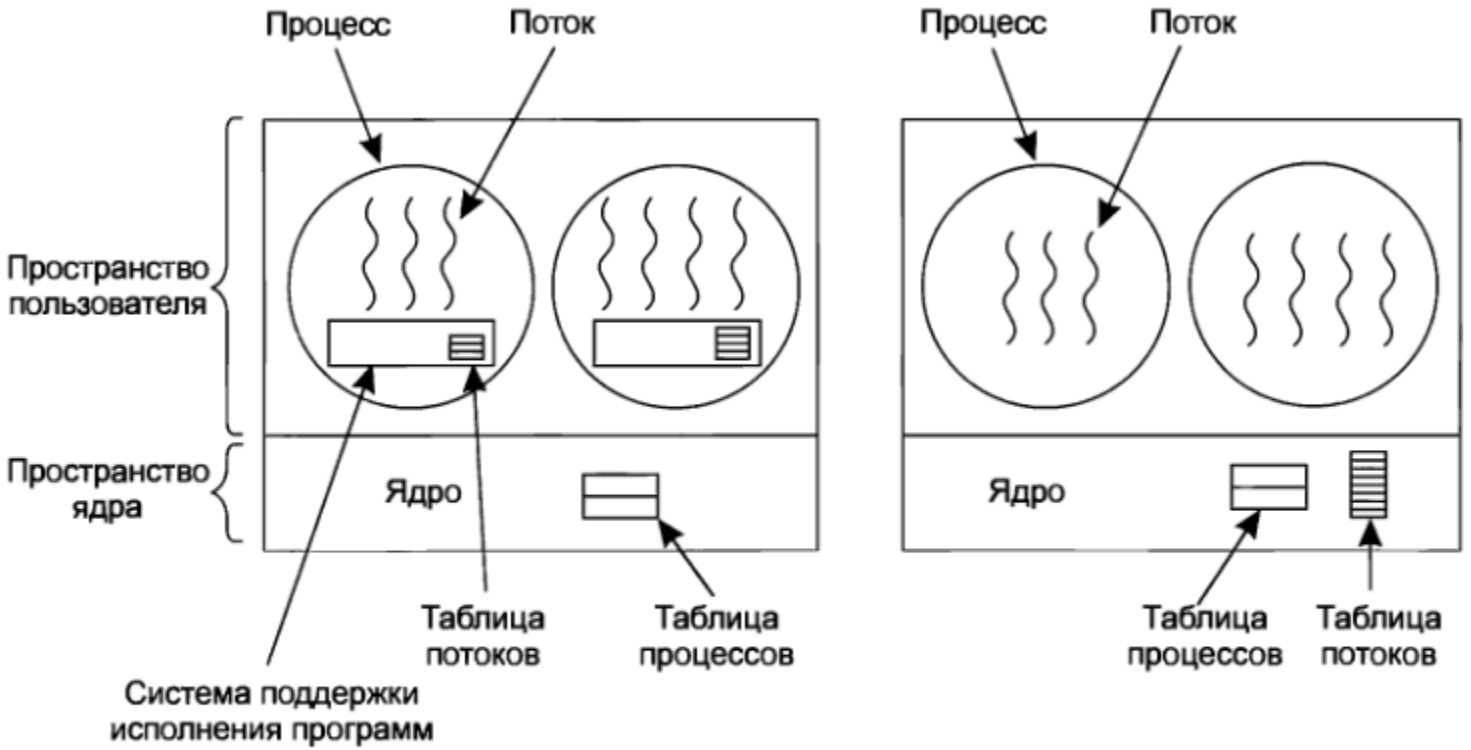
- Распараллеливание вычислительного процесса (*например при использовании блокирующих функций*).
- Повышение скорости выполнения процессов, в которых большая часть времени тратится на ожидание ввода-вывода.
- Не все задачи удобно распараллеливать с помощью процессов (должны быть общие адресные пространства);
- Потоки полезны для систем, имеющих несколько процессоров/ядер, где есть реальная возможность параллельных вычислений (отдельные потоки могут выполняться на отдельныхпроцессорах/ ядрах).

Многопоточная модель процесса

Для каждого потока создаётся:

- Управляющий блок потока;
- Стек потока и стек ядра.

Варианты расположения таблицы потоков



Набор потоков на пользовательском уровне (*а*); набор потоков, управляемый ядром (*б*)

Создание потоков в Linux

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

// определение функции потока
void * thread_func(void *arg)
{ int i;
int loc_id = * (int *) arg;
for (i = 0; i < 4; i++) {
printf("Thread %i is running\n", loc_id);
sleep(1);
            }
}

//главная программа
int main(int argc, char * argv[])
{
 int id1, id2, result;
pthread_t     thread1, thread2;  //обьявление индентификаторов  потока
id1 = 1;
result = pthread_create(&thread1, NULL, thread_func, &id1);// создание потока1
if (result != 0) {
perror("Creating the first thread");
return EXIT_FAILURE;
```

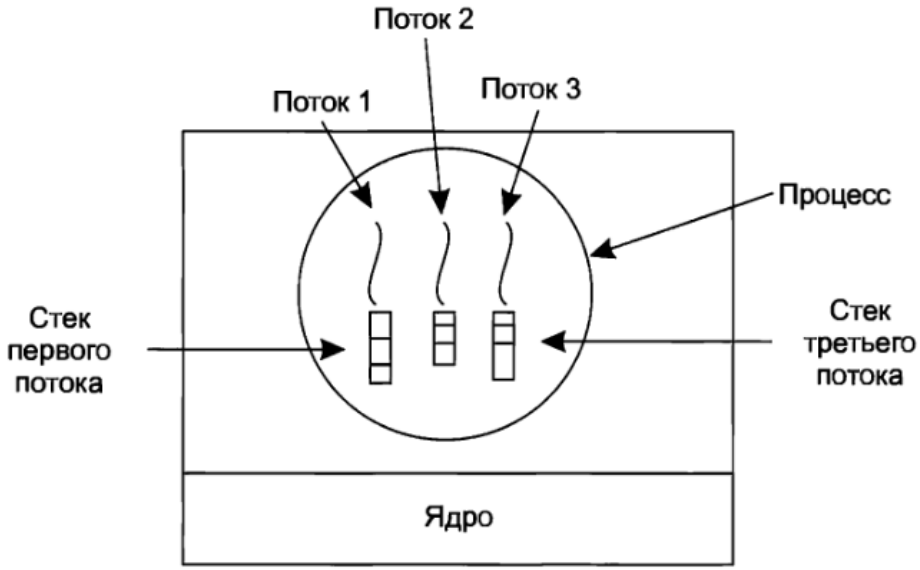
Потоки

- thread1 – переменные куда будет сохранен ID-потока
- NULL – атрибуты потока
- thread_func – функция, которая будет выполняться в потоке
- Id1 – это аргумент, или аргументы, которые будут переданы потоку
- При удачном выполнении функция возвращает 0

Процессы и Потоки

Использование объектов потоками

Элементы, присущие каждому процессу	Элементы, присущие каждому потоку
Адресное пространство	Счетчик команд
Глобальные переменные	Регистры
Открытые файлы	Стек
Дочерние процессы	Состояние
Необработанные аварийные сигналы	
Сигналы и обработчики сигналов	
Учетная информация	



У каждого потока имеется свой собственный стек