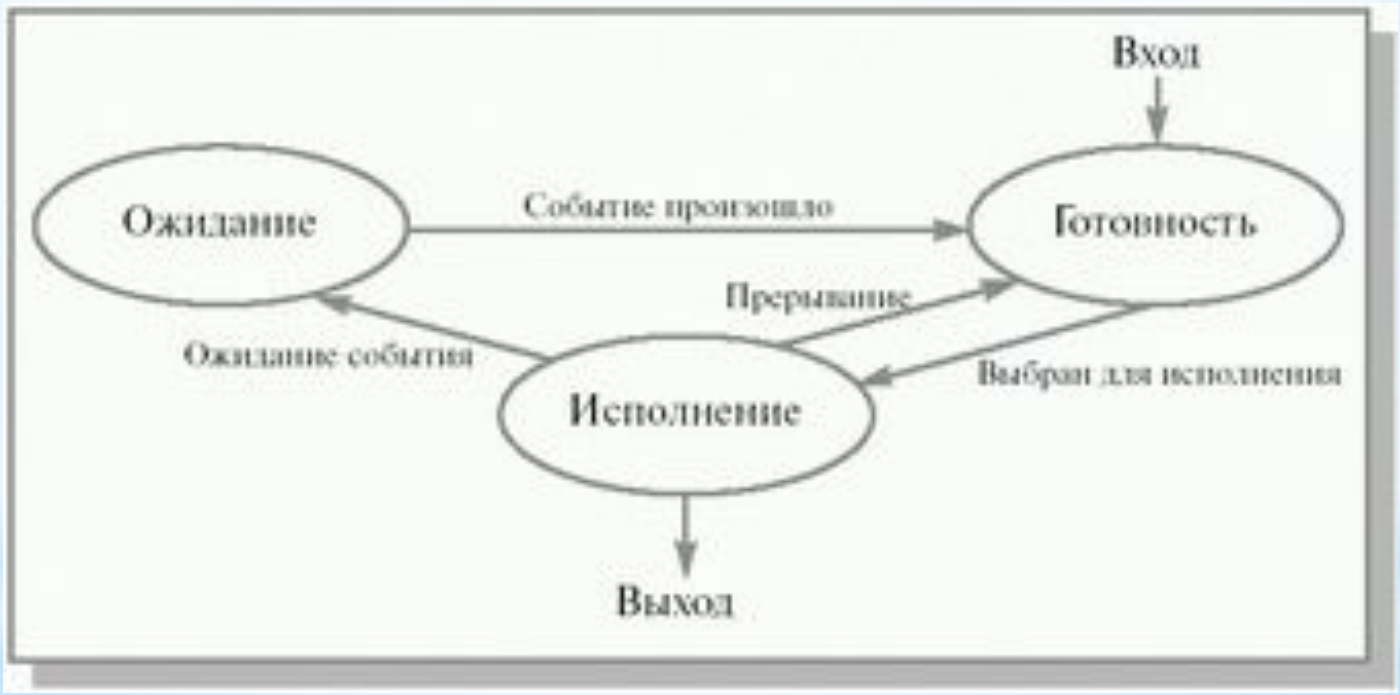
Процеси

Розглянемо приклад. Два студента запускають програму добування квадратного кореня. Один хоче обчислити квадратний корінь з 4, а другий - з 1. З погляду студентів, запущена одна і та ж програма; з погляду комп'ютерної системи, їй доводиться займатися двома різними обчислювальними процесами, тому що різні вихідні дані приводять до різних наборів обчислень. Отже, на рівні обчислювальної системи ми не можемо використовувати термін "програма" в користувацькому сенсі слова.

Нехай обоє студента намагаються витягти корінь квадратний з 1, тобто нехай вони сформували ідентичні завдання, але завантажили їх в обчислювальну систему із зсувом за часом. У той час як одне з виконуваних завдань приступило до друку отриманого значення і чекає закінчення операції введення-виведення, друге тільки починає виконуватися. Чи можна говорити про ідентичність завдань всередині обчислювальної системи в даний момент? Ні, так як стан процесу їх виконання різному. Отже, і слово "завдання" в користувацькому сенсі не може застосовуватися для опису того, що відбувається в обчислювальній системі.

На однопроцесорній комп'ютерній системі в кожен момент часу може виконуватися тільки один процес. Для мультипрограмних обчислювальних систем псевдопаралельна обробка декількох процесів досягається за допомогою перемикання процесора з одного процесу на інший. Поки один процес виконується, інші чекають своєї черги.

Розіб'ємо стан процес не виконується на два нових стани: готовність і очікування



Вийти з цього стану (виконується) процес може з трьох причин:

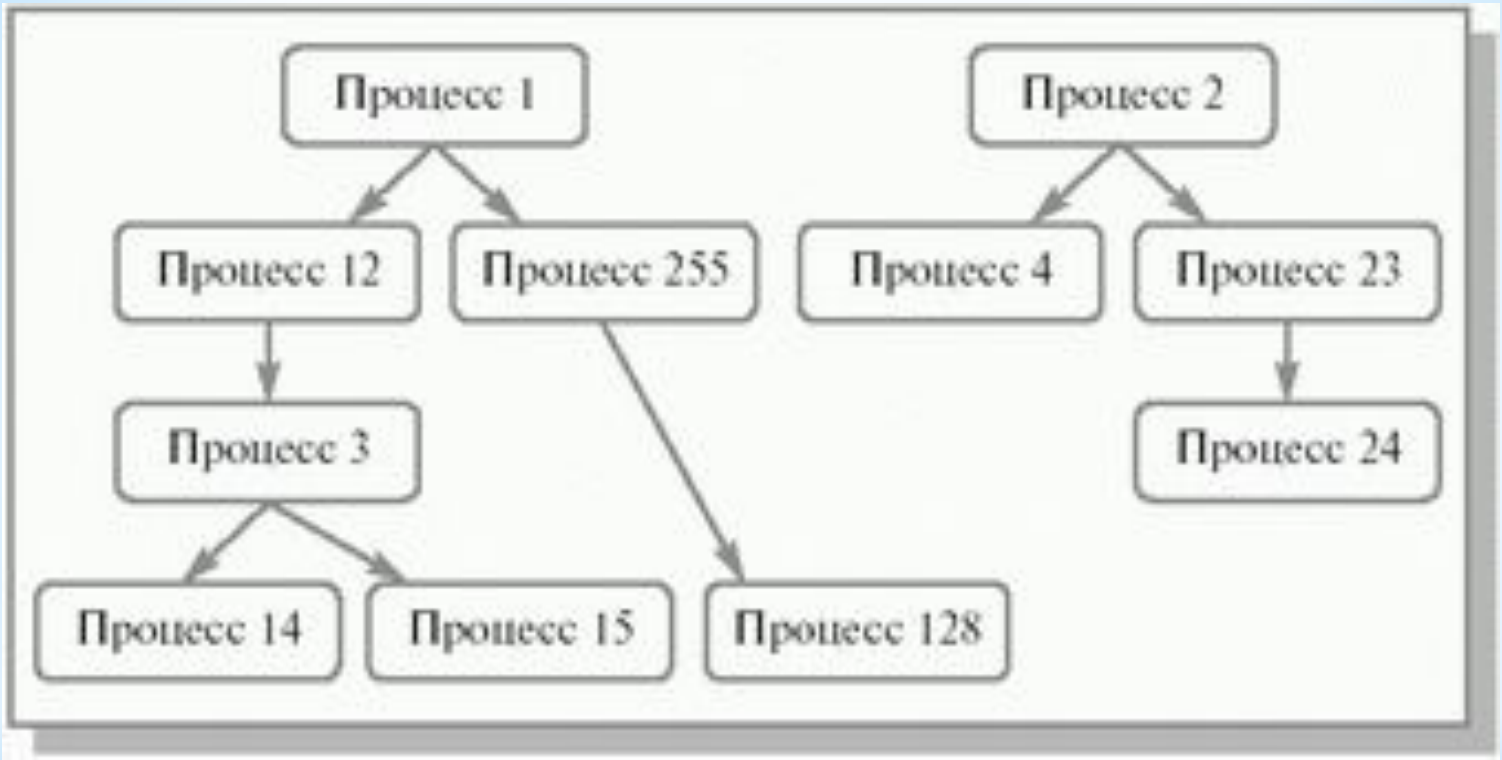
- операційна система припиняє його діяльність;
- він не може продовжувати свою роботу, поки не відбудеться деяка подія, і операційна система переводить його в стан очікування;
- в результаті виникнення переривання в обчислювальній системі (наприклад, переривання від таймера після закінчення передбаченого часу виконання) його повертають в стан готовність.


Рождение
Допуск к планированию
Ожидание
Событие произошло
Готовность
Прерывание
Ожидание события
Выбран для исполнения
Исполнение
Завершение работы
Закончил
исполнение

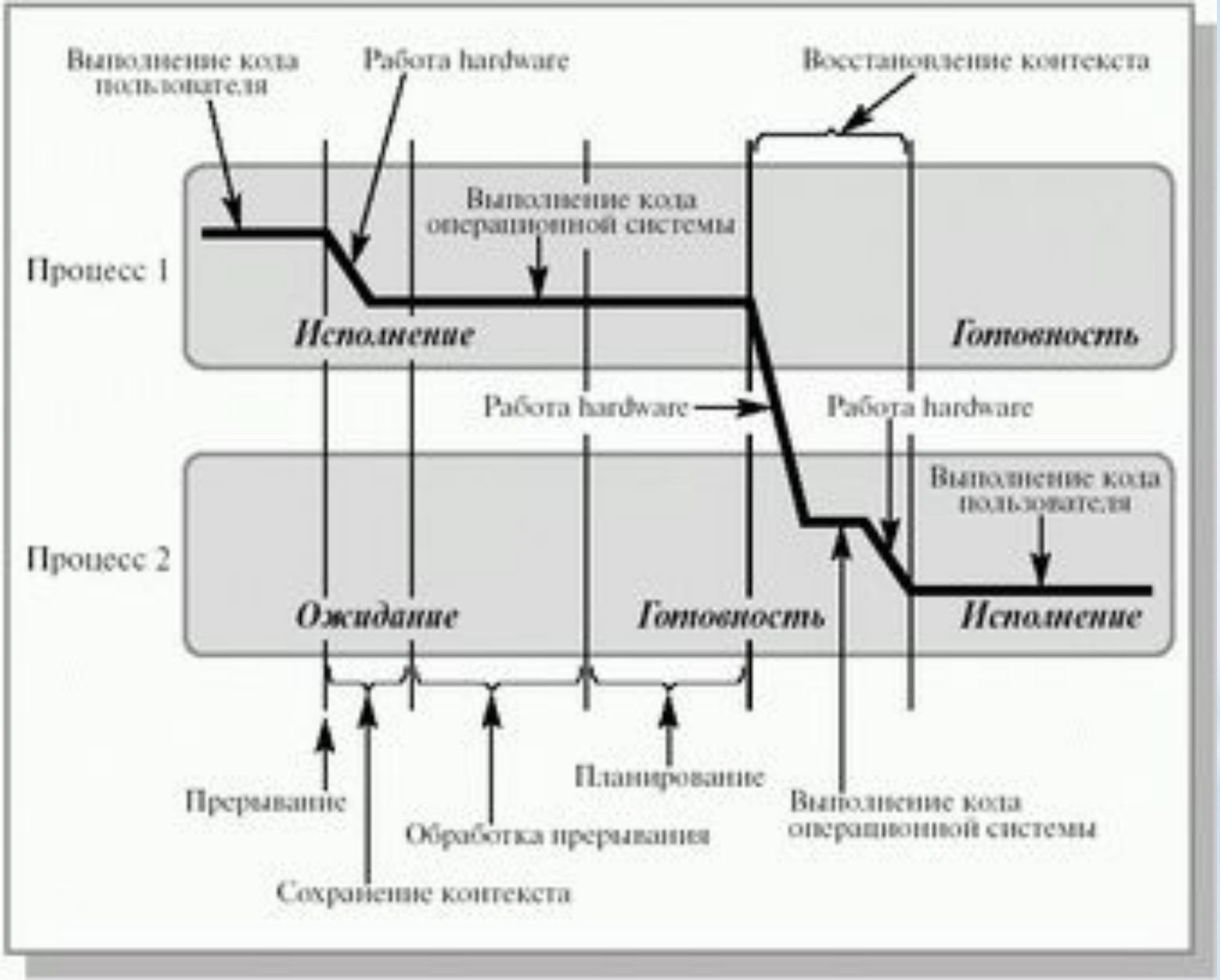
Process Control Block і контекст процесу

Для того щоб операційна система могла виконувати операції над процесами, кожен процес подається в ній деякою структурою даних. Ця структура містить інформацію, специфічну для даного процесу:

- **стан, в якому перебуває процес;**
- **програмний лічильник процесу або, іншими словами, адресу команди, яка повинна бути виконана для нього наступною;**
- **вміст регістрів процесора;**
- **дані, необхідні для планування використання процесора і управління пам'яттю (пріоритет процесу, розмір і розташування адресного простору і т.д.);**
- **облікові дані (ідентифікаційний номер процесу, який користувач ініціював його роботу, загальний час використання процесора даним процесом і т.д.);**
- **відомості про пристрої введення-виведення, пов'язаних з процесом (наприклад, які пристрої закріплені за процесом, таблиці відкритих файлів).**


Процесс 1
Процесс 2
Процесс 12
Процесс 255
Процесс 4
Процесс 23
Процесс 3
Процесс 24
Процесс 14
Процесс 15
Процесс 128

Розглянемо наступний приклад. Нехай процес з номером 2515 був породжений процесом з номером 2001 і після завершення його роботи залишається в обчислювальній системі необмежено довго. Тоді не виключено, що номер 2001 буде використаний операційною системою повторно для зовсім іншого процесу. Якщо не змінити інформацію про процес-батьку для процесу 2515, то генеалогічний ліс процесів виявиться некоректним - процес 2515 вважатиме своїм батьком новий процес 2001, а процес 2001 буде відхрещуватися від нежданого нащадка. Як правило, "осиротілі" процеси "усиновлюються" одним із системних процесів, який породжується при старті операційної системи і функціонує весь час, поки вона працює.


Выполнение кода пользователя
Работа hardware
Восстановление контекста
Выполнение кода операционной системы
Процесс 1
Исполнение
Готовность
Работа hardware
Работа hardware
Выполнение кода пользователя
Процесс 2
Ожидание
Готовность
Исполнение
Прерывание
Планирование
Выполнение кода операционной системы
Обработка прерывания
Сохранение контекста

Для ілюстрації сказаного давайте розглянемо наступну програму:

```
#include <sys/types.h>
#include <unistd.h>
#include <stdio.h>
int main()
{
    pid_t pid, ppid;
    int a = 0;
    (void)fork();

    a = a+1;

    pid = getpid();
    ppid = getppid();

printf("My pid = %d, my ppid = %d,
      result = %d\n", (int)pid, (int)ppid, a);
    return 0;
}
```

```
pid = fork();
if(pid == -1){
    ...
    /* ошибка */
    ...
} else if (pid == 0){
    ...
    /* ребенок */
    ...
} else {
    ...
    /* родитель */
    ...
}
```

```
#include <sys/types.h>
#include <unistd.h>
#include <stdio.h>
int main(int argc, char *argv[],
        char *envp[]){


(void) execle("/bin/cat", "/bin/cat",
            "03-2.c", 0, envp);

/* Сюда попадаем только при
   возникновении ошибки */
printf("Error on program start\n");
exit(-1);
return 0;      /* Никогда не выполняется, нужен
                для того, чтобы компилятор не
                выдавал warning */
}
```