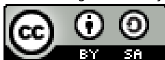
Процессы и межпроцессное взаимодействие (IPC)

Основные понятия и определения

Это произведение доступно по лицензии
Creative Commons “Attribution-ShareAlike” (“Атрибуция — На тех же условиях”) 3.0 Непортированная.
http://creativecommons.org/licenses/by-sa/3.0/deed.ru



Определение процесса

Определение процесса (*process*) составим из компонентов, необходимых для выполнения поставленной перед системой задачи (*task*).

- **программа**

Определение процесса

Определение процесса (*process*) составим из компонентов, необходимых для выполнения поставленной перед системой задачи (*task*).

- **программа**
- **загруженная** в память, непосредственно доступную процессору для выполнения;

Определение процесса

Определение процесса (*process*) составим из компонентов, необходимых для выполнения поставленной перед системой задачи (*task*).

- **программа**
- **загруженная** в память, непосредственно доступную процессору для выполнения;
- **и выполняющаяся** (или способная выполняться).

Программы бывают

Бинарные — обычно написанные на компилируемом языке (C, C++, Fortran, Pascal), и превращенные в форму, понятую процессору **компилятором** и **компоновщиком**. Некоторые языки требуют выполнения в *виртуальной машине* (Java).

Программы бывают

Бинарные — обычно написанные на компилируемом языке (C, C++, Fortran, Pascal), и превращенные в форму, понятую процессору **компилятором** и **компоновщиком**. Некоторые языки требуют выполнения в *виртуальной машине* (Java).

Интерпретируемые (скрипты) — написанные на таких языках программирования, которые анализируют и выполняют текст программы каждый раз, когда их запускают. На самом деле выполнятся бинарная программа — интерпретатор (bash, awk, sed). Часто происходит предварительная компиляция в промежуточную форму (perl, python).

Бинарные программы бывают

Статически скомпонованные — все необходимые части программы (функции и данные), как написанные программистом, так и взятые из библиотек (.a или .lib) помещены в её файл. Программу осталось только загрузить и передать управление на точку входа. В настоящее время используются редко.

Бинарные программы бывают

Статически скомпонованные — все необходимые части программы (функции и данные), как написанные программистом, так и взятые из библиотек (.a или .lib) помещены в её файл. Программу осталось только загрузить и передать управление на точку входа. В настоящее время используются редко.

Динамически скомпонованные в файл программы помещены только те части программы, которые в ней написаны. Всё остальное подключается в момент загрузки программы из библиотек динамической компоновки (sharable objects .so или dynamic link libraries .dll).

Компоненты загруженной программы - 1

Рассмотрим фрагмент программы на языке C:

```
int global_var = 7;
int f( int a, int b )
{
  static int n = 0;
  int c = a + b, d = n + 1, e = c + d;
  return c + d + n++;
}
int main()
{
  int g = f( 5, 2 );
  return 0;
}
```

Как будут представлены части программы в бинарном представлении (образе программы)?

Компоненты загруженной программы - 2

Код программы

Действия, выполняемые в программе (операторы, инструкции, операторы), представляют собой КОД программы. Превращаются компилятором в бинарный вид, понятный конкретному классу процессоров.

Код программы располагается в **сегменте кода** (CODE segment). Текущая выполняющаяся инструкция задаётся **указателем кода** (IP, instruction pointer).

Обычно не допускается модификация сегмента кода в процессе выполнения, за исключением ручной загрузки библиотек.

Сегментов кода может быть несколько.

Компоненты загруженной программы - 3

Сегмент данных

Глобальные переменные, статические переменные файлов, функций и классов располагаются в **сегменте данных** (DATA segment).

```
int global_var = 7; // <——— глобальная переменная
int f( int a, int b )
{
  static int n = 0; // <——— статическая переменная
                    //        функции
  int c = a + b, d = n + 1, e = c + d;
  return c + d + n++;
}
int main()
{
  int g = f( 5, 2 );
  return 0;
}
```

Часто разделяют инициализированные данные (DATA) и не инициализированные (BSS).

Компоненты загруженной программы - 4a

Что ещё надо хранить

На первый взгляд сегмента данных достаточно для хранения данных. Однако, в большинстве языков одну и ту же функцию можно вызвать из разных мест программы — надо хранить адрес возврата из функции.

Более того — функция может вызывать сама себя (рекурсия) прямо или косвенно. Правило требуют, что бы каждый раз у функции были **отдельные** локальные переменные. Неизвестно, какой будет уровень вложенности — нельзя заранее зарезервировать место в сегменте данных.

Функции получают аргументы и возвращают значения. Эти объекты тоже невозможно разместить в сегменте данных.

Компоненты загруженной программы - 4b

Сегмент стека. 1

Адреса возврата из функций, локальные переменные функций, аргументы (и, при необходимости, возвращаемые значения) размещаются динамически в **сегменте стека** (STACK segment). Текущая позиция в стеке задаётся **указателем стека** (SP, stack pointer). В архитектурах x86, ARM стек растёт вниз, т.е при помещении (PUSH) объекта в стек значение SP уменьшается, а при снятии со стека (POP или RET) — увеличивается.
При каждом вызове функции создаётся **кадр стека**.

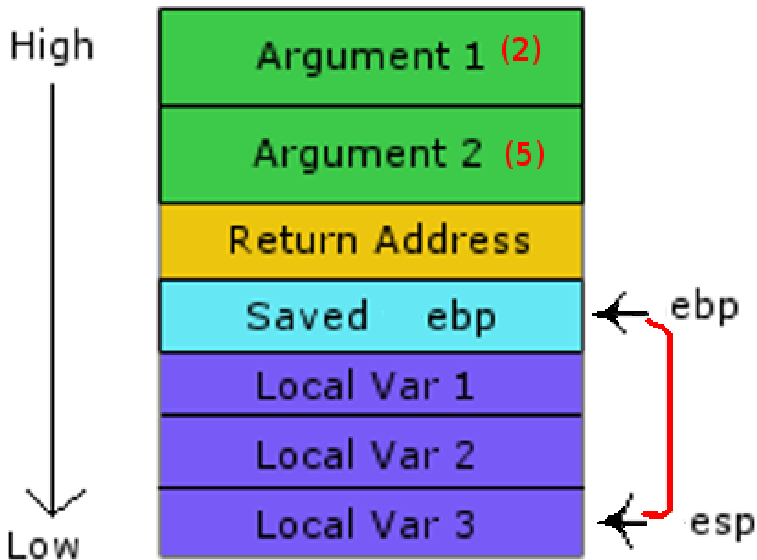
Компоненты загруженной программы - 4b

Сегмент стека. 2

Действия при вызове функции и возврате из неё:

- Вызывающая функция при необходимости резервирует место в стеке под возвращаемое значение, изменяя SP.
- Затем в стек помещаются значения аргументов (в языке C — начиная с последнего).
- В стек помещается адрес возврата (IP+n) и происходит переход на код функции (CALL).
- Функция изменяет SP так, что бы между адресом возврата и вершиной стека поместились все локальные переменный.
- Перед возвратом функция возвращает SP на место, и выполняет инструкцию RET, которая снимает из стека адрес возврата и переходит на него.
- Вызывающая функция убирает со стека всё, что положила туда при вызове (C).

Кадр стека



Классификация ОС по работе с процессами

Операционные системы бывают:

однозадачные — обеспечивают выполнение одного пользовательского процесса (DOS . . .).

многозадачные — обеспечивают параллельное или псевдо-параллельное (путём переключения) выполнение нескольких пользовательских процессов

Многозадачность бывает

кооперативная (или невытесняющая) — процессы пользователя сами отдают управление ядру (Windows 3.x, 9x?).

вытесняющая (или настоящая) — ядро переключает процессы независимо от их действий (Unix-ы, Linux, Windows NT+)

Планировщик

Задача переключения между процессами возложена на **планировщик** (scheduler) — неотъемлемую часть многозадачного ядра.
При вытесняющей многозадачности планировщик обычно получает управление от обработчика прерывания аппаратного таймера.
По методам работы планировщика операционные системы делят на

общего назначения — работают как могут.

реального времени — гарантирующие, что каждый процесс получит не процессорного времени не менее заданного в единицу времени, и не реже заданного интервала.

Единицы измерения времени в ОС

При измерении временных интервалов в планировании процессов, помимо привычных единиц измерения используются специальные:

такт — интервал между двумя последовательными срабатываниями задающего генератора процессора. Процессор может выполнить и несколько команд за такт, так и одну команду на несколько тактов.

тик — интервал времени между двумя последовательными срабатываниями аппаратного таймера. Планировщик обычно не может получить управление чаще, чем раз в 1 тик, и, соответственно, не может гарантированно выделить задаче меньший интервал времени.

квант — минимальный интервал времени, который планировщик может выделить процессу. Обычно несколько тиков.

Идентификация процессов

Операционная система должна однозначно идентифицировать все процессы. Для этого каждому процессу присваивается число — PID (Process IDentificator).

В ядре существует **таблица процессов**. Для каждого существующего процесса в этой таблице хранится указатель на структуру, которая определяет все параметры процесса — выделенные сегменты, состояние регистров процесса и параметры планирования, информацию об пользователе и группе, открытых файлах и . . . . . .

Также процесс может находится в различных состояниях:

- работает — выполняется в данный момент
- готов к работе — будет выполнятся, как только планировщик выделит время
- заблокирован — ожидает какого-либо события
- . . .

Иллюстрация — ядро и процессы