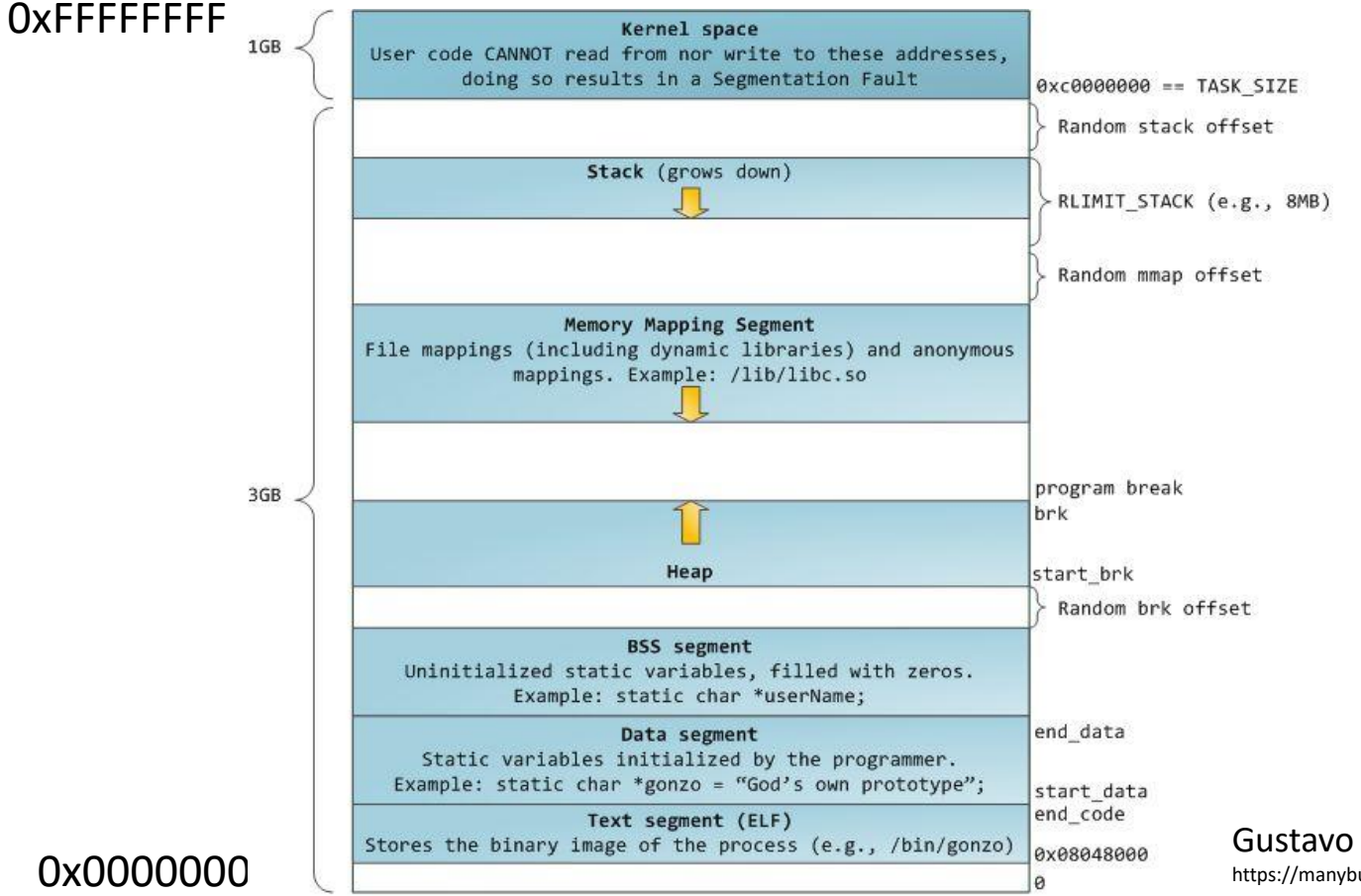
Процесс

- Исполняемый файл
- Карта памяти (`cat /proc/<pid>/maps`)
- Файловые дескрипторы
- Состояние процесса
- Атрибуты безопасности
- Атрибуты процесса (pid, ppid, gid, sid)
- Дескрипторы ресурсов ОС

https://elixir.bootlin.com/linux/latest/source/include/linux/sched.h#L739
struct **task_struct** {…};

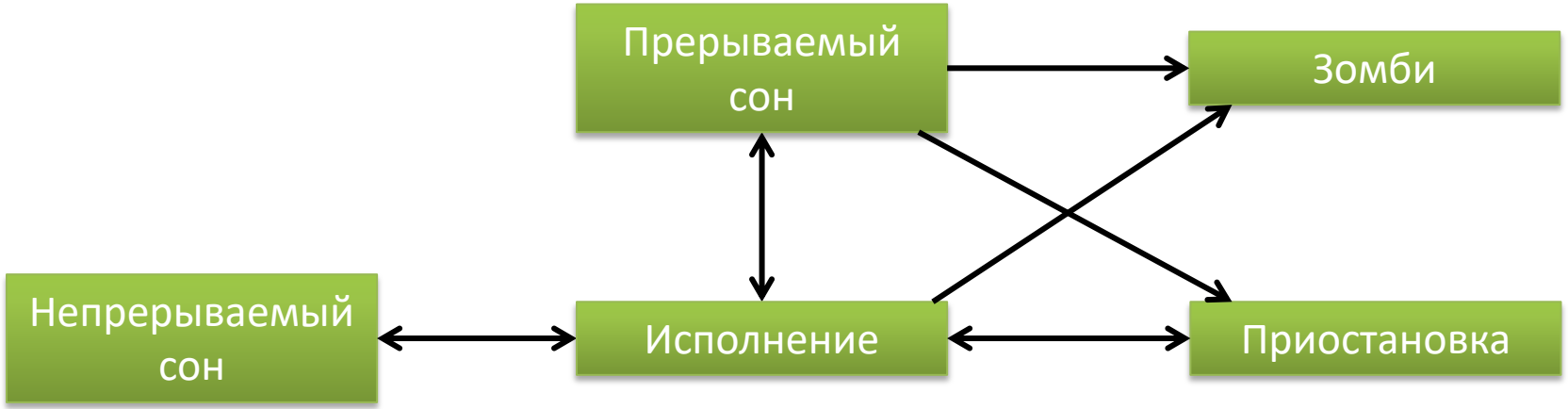
Карта памяти процесса



Gustavo Duarte

https://manybutfinite.com/post/anatomy-of-a-program-in-memory/

Состояния процесса



- **Исполнение (R)** – процесс готов выполняться и либо выполняется, либо находится в очереди на выполнение.
- **Прерываемый сон (S)** – ожидание доступности ресурса, при этом оно может прерваться при получении сигнала.
- **Непрерываемый сон (D)** – ожидание ресурса, которое невозможно прервать. Обычно используется в пространстве ядра.
- **Приостановка (T)** – временное прекращение исполнения (Ctrl + Z).
- **Зомби (Z)** – завершившийся процесс, но его ресурсы еще не высвобождены. Другие процессы могут ожидать завершения процесса, поэтому какое-то время он находится в таком состоянии.

Идентификаторы процесса

- **PID** (Process ID) – уникальный идентификатор процесса.
- **PPID** (Parent PID) – идентификатор родительского процесса.
- **UID** (User ID) – идентификатор пользователя, создавшего процесс.
- **EUID** (Effective UID) – это «эффективный» UID. Используется для определения ресурсов, к которым у процесса есть право доступа.
- **GID** (Group ID) – идентификатор группы, к которой принадлежит владелец процесса. **EGID** – «эффективный» GID.
- **SID** (Session ID) – идентификатор сессии.

ps, top, htop, pstree

Ключи команды ps

-A Все процессы.
-a Запущенные в текущем терминале, кроме главных системных.
-d Все, кроме главных системных процессов сеанса.
-e Все процессы.
f Показать дерево процессов с родителями.
T Все на конкретном терминале.
a Все, связанные с текущим терминалом и терминалами других пользователей.
r Список только работающих процессов.
x Отсоединённые от терминала.
u Показать пользователей, запустивших процесс.

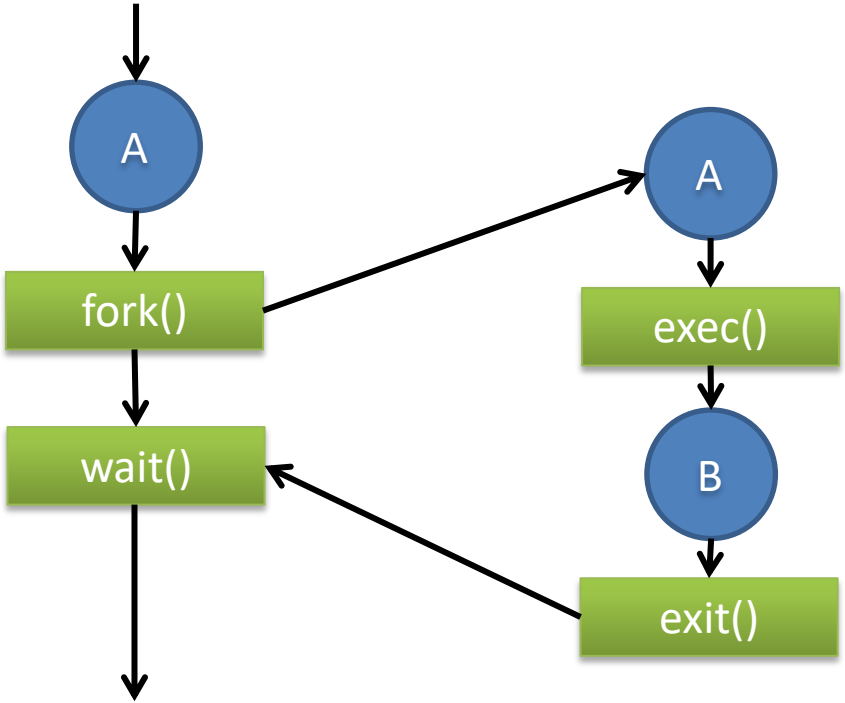
Системные вызовы

<sys/types.h>, <unistd.h>

- pid_t **getpid** (void);
- pid_t **getppid** (void);
- uid_t **getuid**(void); int **setuid**(uid_t uid);
- uid_t **geteuid**(void); int **seteuid**(uid_t uid);
- gid_t **getgid** (void); int **setgid**(gid_t gid);
- pid_t **getsid**(pid_t pid); pid_t **setsid**(void);

Задание 1.1 (1 балл). Проверить работу системных вызовов get*.

Управление процессами



Управление процессами

```
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>
int main()
{
  pid_t pid;
  int rv;
  switch(pid = fork()) {
    case -1:
      perror("fork");
      exit(EXIT_FAILURE);
    case 0:
      printf("CHILD: PID - %d\n", getpid());
      printf("CHILD: PPID - %d\n", getppid());
      exit(EXIT_SUCCESS);
    default:
      printf("PARENT: PID - %d\n", getpid());
      printf("PARENT: CHILD PID - %d\n", pid);
      wait(&rv);
      printf("PARENT: RETURN STATUS FOR CHILD
                      - %d\n", WEXITSTATUS(rv));
  }
  exit(EXIT_SUCCESS);
}
```

Завершение программы

- Обычное завершение программы
  `void exit(int status);`
  status: EXIT_SUCCESS и EXIT_FAILURE

- Регистрация функции, вызываемой при нормальном завершении работы программы

```
int atexit(void (*function)(void));
int on_exit(void (*function)(int, void *),
            void *arg);
```

Задания

1.2 (1 балл). Написать программу, порождающую дочерний процесс. Затем и родительский, и дочерний процессы выводят значения аргументов запуска
(`int main (int argc, char *argv[])`).
Добавить функцию, вызываемую при нормальном завершении работы программы и проверить ее работу.

1.3 (2 балла). Написать программу, вычисляющую площади квадратов с заданной длиной стороны. Длины сторон передаются как аргументы запуска. Расчеты делают родительский и дочерний процессы, разделяя задачи примерно поровну.

Замена образа процесса

```
#include <unistd.h>
extern char **environ; // окружение нового образа процесса
```

- `int execl(const char *path, const char *arg, ...);`
- `int execlp(const char *file, const char *arg, ...);`
- `int execle(const char *path, const char *arg , ..., char * const envp[]);`

  const char *arg: первый параметр – имя файла, который надо исполнить; последний – NULL.

- `int execv(const char *path, char *const argv[]);`
- `int execvp(const char *file, char *const argv[]);`

Замена образа процесса

```
// Программа hello

#include <stdio.h>
int main(int argc, char *argv[])
{
  int i=0;
  printf("%s\n",argv[0]);
  printf("Args: ");
  while(argv[++i] != NULL)
   printf("%s ",argv[i]);
  return 0;
}
```

```
#include <stdio.h>
#include <unistd.h>
int main(int argc, int *argv[])
{
  printf("Run: %s", argv[0]);
  execl("hello"," ","Hello", "World!", NULL);
  // execv("hello", argv);
  return 0;
}
```

Задания

1.4 (2 балла). Написать программу, похожую на командный интерпретатор. При запуске выводится приглашение, и пользователь вводит имя и аргументы программы, которую желает запустить. Программы для запуска находятся в том же каталоге (например, программа для вычисления суммы аргументов, «склеивания» строк, поиска наибольшего значения или наибольшей длины строки и т.д.).

1.5 (3 балла). Усовершенствовать программу 1.4: добавить возможность запуска программ, установленных в системе.

Взаимодействие процессов

- Работа с файлами
- Каналы
- Сигналы
- Разделяемая память
- System V и POSIX
- Удаленный вызов процедур
- Сетевое взаимодействие