Секция № 2

Процессы

Содержание

1. Термины . . . 1
2. Разогрев . . . 2
   2.1. Привет, мир! . . . 2
3. Задачи . . . 2
   3.1. Форки . . . 2
   3.2. Размещение в стеке . . . 2
   3.3. Размещение в куче . . . 3
   3.4. Чуть более сложное размещение в стеке . . . 3
   3.5. Простой `wait` . . . 4
   3.6. Fork и дескрипторы файлов . . . 5
   3.7. Exec . . . 6
   3.8. Exec + Fork . . . 6
   3.9. Эффективная реализация `fork` (подход) . . . 7

1. Термины

Процесс — это экземпляр выполняемой на компьютере программы, он состоит из адресного пространства и одной или нескольких нитей.

Адресное пространство — это множество адресов памяти, доступные процессу. Адресные пространства разных процессов являются *приватными*, то есть один процесс не может доступиться до адресного пространства другого процесса, если пересечение адресных пространств специально, особым образом не настроено.

Стек — это область памяти используемая для исполнения нитей. При вызове функции, на вершине стека создается запись для хранения локальных переменных и некоторой дополнительной информации. При выходе из функции запись с вершины стека удаляется, на её месте может быть записана информация о другом вызове. Стек всегда работает по правилу LIFO (last-in-first-out — первым пришёл, последним ушёл), по-другому это правило называется «чулок с апельсинами».

Куча — это область памяти предназначенная для динамического размещения данных. В противоположность стеку для кучи не существует правила работы с блоками данных. Можно занимать и освобождать пространство в куче в любое время.

fork — C-функция, вызывающая `syscall` (системный вызов) для создания нового процесса в виде копии существующего. Новый процесс, называемый «дочерним», является абсолютной копией вызывающего процесса (за исключением некоторых деталей, подробнее смотри в `man`). Оба процесса и «родительский», и «дочерний» выходят из функции `fork`. При успешном выполнении, в родительском процессе функция вернёт `PID` дочернего процесса, а в «дочернем» процессе возвращаемое значение будет равно нулю. При возникновении ошибки в «родительском» процессе вернётся `-1`, а «дочерний» процесс не будет порождён.

wait — семейство C-функций, являющихся обертками над системными вызовами, использующимися для получения информации об изменении состояния «дочернего» процесса. Под изменением состояния процесса понимается: завершение процесса, останов процесса по сигналу или продолжение выполнения процесса.

Код выхода — однобайтовое целое значение возвращаемое при завершении из «дочернего» (вызываемого) процесса в «родительский» (вызывающий).

exec — семейство функций, заменяющих исполняемую программу в текущем процессе на новую. В качестве аргумента в `exec` передается имя исполняемого файла.

2. Разогрев

2.1. Привет, мир!

Что напечатает эта программа на C? Считай, что `PID` дочернего процесса `90210`.

Тут может быть более одного правильного ответа.

```
int main() {
  pid_t pid = fork();
  printf("Hello World: %d\n", pid);
}
```

Ответ

3. Задачи

3.1. Форки

Сколько новых процессов будет создано следующей программой. Считай, что все вызовы `fork` успешны.

```
int main(void)
{
  for (int i = 0; i < 3; i++) {
      pid_t pid = fork();
  }
}
```

Ответ

3.2. Размещение в стеке

Что будет напечатано?

```c
int main(void)
{
    int stuff = 5;
    pid_t pid = fork();
    printf("The last digit of pi is %d\n", stuff);
    if (pid == 0)
      stuff = 6;
}
```

Ответ

3.3. Размещение в куче

Что будет напечатано?

```c
int main(void)
{
    int* stuff = malloc(sizeof(int)*1);
    *stuff = 5;
    pid_t pid = fork();
    printf("The last digit of pi is %d\n", *stuff);
    if (pid == 0)
      *stuff = 6
}
```

Ответ

3.4. Чуть более сложное размещение в стеке

Что будет напечатано в этом случае?

```
void printTenNumbers(int *arr)
{
  int i;
  printf("\n");
  for(i=0; i<10; i++) {
    printf("%d",arr[i]);
  }
  exit(0);
}

int main() {
  int *arr, i;
  arr = (int *) malloc (sizeof(int));

  arr[0] = 0;
  for(i=1; i<10; i++) {
    arr = (int *) realloc( arr, (i+1) * sizeof(int));
    arr[i] = i;
    if (i == 7) {
      pid_t pid = fork();
      if (pid == 0) {
        printTenNumbers(arr);
      }
    }
  }
  printTenNumbers(arr);
}
```

Ответ

3.5. Простой `wait`

Что будет напечатано? Считай, что дочерний `PID` равен `90210`.

```
int main(void)
{
    pid_t pid = fork();
    int exit;
    if (pid != 0) {
        wait(&exit);
    }
    printf("Hello World\n: %d\n", pid);
}
```

Ответ

Как переписать эту программу с использованием `waitpid` вместо `wait`?

Ответ

3.6. Fork и дескрипторы файлов

Каково будет содержимое файла `output.txt` после выполнения программы?

```
int main(void)
{
  int fd;
  fd = open("output.txt", O_CREAT|O_TRUNC|O_WRONLY, 0666);
  if(!fork()) {
    write(fd, "hello ", 6);
    exit(0);
  } else {
    int status;
    wait(&status);
    write(fd, "world\n", 6);
  }
}
```

Ответ

3.7. Exec

Что будет напечатано?

```
int main(void)
{
  char** argv = (char**) malloc(3*sizeof(char*));
  argv[0] = "/bin/ls";
  argv[1] = ".";
  argv[2] = NULL;
  for (int i = 0; i < 10; i++) {
    printf("%d\n", i);
    if (i == 3)
      execv("/bin/ls", argv);
  }
}
```

Ответ

3.8. Exec + Fork

Как модифицировать предыдущую программу, используя `fork`, так, чтобы она и распечатала все числа от `0` до `9`, и вывела результат выполнения `ls`? Порядок вывода роли не играет. Нельзя удалять или переставлять строки кода, можно только добавлять (и использовать `fork`!)

Ответ

3.9. Эффективная реализация `fork` (подход)

Припомни, что `fork` создает полную копию родительского процесса, включая полную копию адресного пространства. Представь, что ты разработчик ОС, и перечисли шаги, которые были бы тобой предприняты для обеспечения более эффективного копирования адресного пространства.

Ответ