Санкт-Петербургский политехнический университет Петра Великого

Кафедра компьютерных систем и программных технологий

Отчёт по лабораторной работе

Курс: Операционные системы»

Тема: «Структура контекста процесса»

Выполнил студент:

Бояркин Никита Сергеевич
Группа: 43501/3

Проверил:

Душутина Елена Владимировна

Санкт-Петербург
2016 г.

Содержание

Дополнительное задание №1

1.1 Характеристики системы

Некоторая информация об операционной системе и текущем пользователе:

```
nikita@nikita-VirtualBox:~$ who
nikita   tty7         2016-10-30 12:49 (:0)
nikita@nikita-VirtualBox:~$ cat /proc/version
Linux version 4.4.0-45-generic (buildd@lgw01-34) (gcc version 5.4.0 20160609 (Ubuntu
    5.4.0-6ubuntu1~16.04.2) ) #66-Ubuntu SMP Wed Oct 19 14:12:37 UTC 2016
```

1.2 Ход работы

1.2.1 Структура контекста процесса

Контекст процесса включает в себя содержимое адресного пространства задачи, выделенного процессу, а также содержимое относящихся к процессу аппаратных регистров и структур данных ядра. С формальной точки зрения, контекст процесса объединяет в себе пользовательский контекст (user-level context), регистровый контекст (register context) и системный контекст (system-level context).

Пользовательский контекст

Пользовательский контекст состоит из команд и данных процесса, стека задачи и содержимого совместно используемого пространства памяти в виртуальных адресах процесса. Те части виртуального адресного пространства процесса, которые периодически отсутствуют в оперативной памяти вследствие выгрузки или замещения страниц, также включаются в пользовательский контекст.

Регистровый контекст

Регистровый контекст включает в себя содержимое аппаратных регистров процесса, таких как:

- *Счетчик команд (PC)* - указывает на адрес следующей исполняемой команды. Адрес является виртуальным внутри пространства ядра или пространства задачи.
- *Регистр состояния процессора (PS)* - указывает на аппаратный статус машины по отношению к процессу. Регистр состояния процессора обычно содержит поля, которые указывают, является ли результат последнего вычисления нулевым, положительным, отрицательным, переполнен ли регистр с установкой бита переноса и др. В других имеющих важное значение полях регистра состояния процессора указывается текущий уровень прерывания процессора, а также текущий и предыдущий режимы выполнения процесса (режим ядра/задачи). По значению поля текущего режима выполнения процесса устанавливается, может ли процесс выполнять привилегированные команды и обращаться к адресному пространству ядра.
- *Указатель вершины стека (SP)* - в зависимости от архитектуры машины указатель вершины стека показывает на следующий свободный элемент стека или на последний используемый элемент. От архитектуры машины также зависит направление увеличения стека (к старшим или младшим адресам).
- *Регистры общего назначения* - в них содержится информация, сгенерированная процессом во время его выполнения.

Системный контекст

Системный контекст процесса имеет статическую и динамическую части. На протяжении всего времени выполнения процесс постоянно располагает единственную статическую часть системного контекста, но может иметь переменное число динамических частей. Динамическую часть системного контекста можно представить в виде стека, элементами которого являются контекстные уровни, которые помещаются в стек ядром или выталкиваются из стека при наступлении различных событий.

Статическая часть системного контекста включает в себя следующие составляющие:

- Запись в таблице процессов, описывающая состояние процесса и содержащая различную управляющую информацию, к которой ядро всегда может обратиться. Например, информацию о том, в каком режиме выполняется процесс, приостановлен ли процесс, находится ли в переходном состоянии и др.

- Часть адресного пространства задачи, выделенная процессу, где хранится управляющая информация о процессе, доступная только в контексте процесса. Общие управляющие параметры, такие как приоритет процесса, хранятся в таблице процессов, поскольку обращение к ним должно производиться за пределами контекста процесса.

- Записи частной таблицы областей процесса, общие таблицы областей и таблицы страниц, необходимые для преобразования виртуальных адресов в физические, в связи с чем в них описываются области команд, данных, стека и другие области, принадлежащие процессу. Если несколько процессов совместно используют общие области, эти области входят составной частью в контекст каждого процесса, поскольку каждый процесс работает с этими областями независимо от других процессов. В задачи управления памятью входит идентификация участков виртуального адресного пространства процесса, не являющихся резидентными в памяти.

Динамическая часть системного контекста включает в себя следующие составляющие:

- Стек ядра, в котором хранятся записи процедур ядра, если процесс выполняется в режиме ядра. Несмотря на то, что все процессы пользуются одними и теми же программами ядра, каждый из них имеет свою собственную копию стека ядра для хранения индивидуальных обращений к функциям ядра. Пусть, например, один процесс вызывает функцию *creat* и приостанавливается в ожидании назначения нового индекса, а другой процесс вызывает функцию *read* и приостанавливается в ожидании завершения передачи данных с диска в память. Оба процесса обращаются к функциям ядра и у каждого из них имеется в наличии отдельный стек, в котором хранится последовательность выполненных обращений. Ядро должно иметь возможность восстанавливать содержимое стека ядра и положение указателя вершины стека для того, чтобы возобновлять выполнение процесса в режиме ядра. В различных системах стек ядра часто располагается в пространстве процесса, однако этот стек является логически-независимым и, таким образом, может помещаться в самостоятельной области памяти. Когда процесс выполняется в режиме задачи, соответствующий ему стек ядра пуст.

- Динамическая часть системного контекста процесса, состоящая из нескольких уровней и имеющая вид стека, который освобождается от элементов в порядке, обратном порядку их поступления. На каждом уровне системного контекста содержится информация, необходимая для восстановления предыдущего уровня и включающая в себя регистровый контекст предыдущего уровня.

Ядро помещает контекстный уровень в стек при возникновении прерывания, при обращении к системной функции или при переключении контекста процесса. Контекстный уровень выталкивается из стека после завершения обработки прерывания, при возврате процесса в режим задачи после выполнения системной функции, или при переключении контекста. Таким образом, переключение контекста влечет за собой как помещение контекстного уровня в стек, так и извлечение уровня из стека: ядро помещает в стек контекстный уровень старого процесса, а извлекает из стека контекстный уровень нового процесса. Информация, необходимая для восстановления текущего контекстного уровня, хранится в записи таблицы процессов.

Компоненты контекста процесса

На Рисунке 1.1 изображены компоненты контекста процесса. Слева на рисунке изображена статическая часть контекста. В нее входят: пользовательский контекст, состоящий из программ процесса (машинных инструкций), данных, стека и разделяемой памяти (если она имеется), а также статическая часть системного контекста, состоящая из записи таблицы процессов, пространства процесса и записей частной таблицы областей (информации, необходимой для трансляции виртуальных адресов пользовательского контекста). Справа на рисунке изображена динамическая часть контекста. Она имеет вид стека и включает в себя несколько элементов, хранящих регистровый контекст предыдущего уровня и стек ядра для текущего уровня. Нулевой контекстный уровень представляет собой пустой уровень, относящийся к пользовательскому контексту.

Стрелка, соединяющая между собой статическую часть системного контекста и верхний уровень динамической части контекста, означает то, что в таблице процессов хранится информация, позволяющая ядру восстанавливать текущий контекстный уровень процесса.

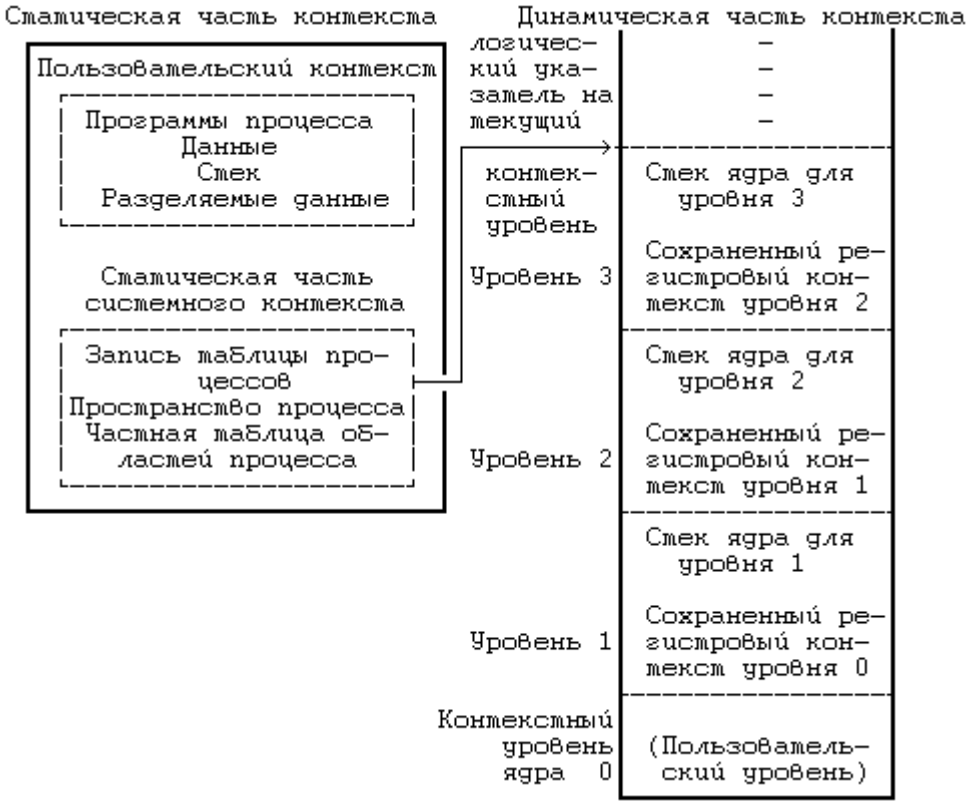

Рис. 1.1

Количество контекстных уровней ограничивается числом поддерживаемых в машине уровней прерывания.

Структура user

Контекст процесса формально описан структурой *struct user* в файле */usr/include/sys/user.h*. Эта структура содержит следующую информацию:

- Адрес дескриптора процесса в таблице процессов.
- Адрес таблицы открытых файлов процесса.
- Адрес таблицы сигналов.
- Блок управления процессом.
- Текущий каталог процесса.
- Корневой каталог процесса.
- Виртуальный адрес процедурного сегмента.
- Виртуальный адрес сегмента инициализированных данных.
- Виртуальный адрес сегмента неинициализированных данных.

1.2.2 Дескриптор процесса

Дескриптор процесса содержит такую информацию о процессе, которая необходима ядру в течение всего жизненного цикла процесса, независимо от того, находится ли он в активном или пассивном состоянии, находится ли образ процесса в оперативной памяти или выгружен на диск. Дескрипторы отдельных процессов объединены в список, образующий таблицу процессов. Память для таблицы процессов отводится динамически в области ядра. На основании информации, содержащейся в таблице процессов, операционная система осуществляет планирование и синхронизацию процессов. В дескрипторе прямо или косвенно (через указатели на связанные с ним структуры) может содержаться информация о состоянии процесса, расположении образа процесса в оперативной памяти и на диске, о значении отдельных составляющих приоритета, а также его итоговое значение – глобальный приоритет, идентификатор пользователя, создавшего процесс, информация о родственных процессах, о событиях, осуществления которых ожидает данный процесс и некоторая другая информация.

Структура proc

Дескриптор процесса формально описан структурой *struct proc* в заголовочном файле */usr/include/sys/proc.h*. Основные поля структуры *struct proc* могут быть классифицированы по характеру данных следующим образом:

- Поля идентификации процесса, такие как: личный идентификатор процесса, идентификатор процесса-предка, идентификатор группы процесса, реальный идентификатор владельца процесса, реальный идентификатор группы владельца процесса, эффективный идентификатор владельца процесса, эффективный идентификатор группы владельца процесса.
- Поля диспетчеризации процессов, такие как: приоритет процесса, системная составляющая приоритета процесса, пользовательская составляющая приоритета процесса, время нахождения процесса в RAM или в области своппинга.
- Поля внутренней синхронизации процессов, такие как: статус процесса, идентификатор события, которое ожидает процесс.
- Поля сигнальной коммуникации процессов, такие как: поле регистрации номеров полученных сигналов, поле регистрации сигналов с отложенной обработкой, номер текущего сигнала, маска номеров игнорируемых сигналов, маска номеров перехваченных сигналов.
- Поля адресации процесса, такие как: адрес сегментной таблицы процесса, адрес контекста процесса.
- Поля размеров сегментов процесса, такие как: размер процедурного сегмента образа процесса, размер сегмента данных образа процесса, размер стека образа процесса.
- Поля ссылок на дескрипторы других процессов, такие как: ссылка на следующий элемент очереди процессов,ссылка на первый элемент очереди процессов, ссылка на последний элемент очереди процессов, ссылка на следующий элемент таблицы процессов, ссылка на дескриптор процесса-предка, ссылка на дескриптор младшего процесса-потомка.

1.2.3 Передача опций в программу, функция getopt

Функция *getopt* разбирает аргументы командной строки. Аргументы *argc* и *argv* являются счетчиком и массивом аргументов, которые передаются функции *main* при запуске программы. Элемент *argv*, начинающийся с −, считается опцией. Символы этого элемента (не считая начального −) являются символами опций. При каждом повторном вызове *getopt* возвращаются символы следующей опции.

Если *getopt* находит символ опции, она возвращает этот символ, обновляя внешнюю переменную *optind* и статическую переменную *nextchar*, так что следующий вызов *getopt* может продолжить проверку со следующего символа опции или элемента *argv*. Если символов опций больше нет, то *getopt* возвращает -1.

Третий аргумент функции Если *getopt* является строкой, содержащей допустимые символы опций. Если за таким символом стоит двоеточие, то опция требует указания аргумента. Два двоеточия означают, что опция имеет необязательный аргумент.

Рассмотрим функцию *getopt* на примере тестовой программы:

```
#include <stdio.h>
#include <unistd.h>

int main(int argc, char *argv[]) {
  int result = 0;
```

```
  // Считываем аргументы, пока они не кончатся
  /*
    Настраиваем функцию getopt следующим образом:
    a – опция без аргументов
    b: – опция обязательно требует аргумент
    C:: – опция необязательно требует аргумент
    d – опция без аргументов
  */
  while ((result = getopt(argc, argv, "ab:C::d")) != -1) {
    switch (result) {
      case 'a':
        printf("found argument \"a\".\n");
        break;
      case 'b':
        printf("found argument \"b = %s\".\n",optarg);
        break;
      case 'C':
        printf("found argument \"C = %s\".\n",optarg);
        break;
      case 'd':
        printf("found argument \"d\"\n");
        break;
      case '?':
        printf("Error found !\n");
        break;
    };
  };

  return 0x0;
}
```

Рассмотрим вывод программы с различными опциями:

```
nikita@nikita-VirtualBox:~/files/work$ gcc getopt.c -o getopt
nikita@nikita-VirtualBox:~/files/work$ ./getopt -a -b -d -C
found argument "a".
found argument "b = -d".
found argument "C = (null)".
nikita@nikita-VirtualBox:~/files/work$ ./getopt -a -b -C -d
found argument "a".
found argument "b = -C".
found argument "d"
nikita@nikita-VirtualBox:~/files/work$ ./getopt -a -b3 -C -d
found argument "a".
found argument "b = 3".
found argument "C = (null)".
found argument "d"
nikita@nikita-VirtualBox:~/files/work$ ./getopt -a -b3 -b2 -b 2332
found argument "a".
found argument "b = 3".
found argument "b = 2".
found argument "b = 2332".
nikita@nikita-VirtualBox:~/files/work$ ./getopt -v
./getopt: invalid option -- 'v'
Error found !
```

Вывод программы значительным образом зависит от того требует ли опция аргумента или нет.

1.2.4 Работа с переменными окружения, функции getenv, setenv

Все переменные окружения хранятся в массиве строк __ *environ*, определенном в файле *unistd.h*. Выведем все переменные окружения для тестовой программы:

```
#include <stdio.h>
#include <unistd.h>

int main(int argc, char **argv) {
  int index = 0;

  // Выводим все параметры среды
  // Последний параметр имеет значение NULL, для обозначения конца списка
  while (__environ[index] != NULL) {
    printf("%s\n", __environ[index]);
   ++index;
  };

  return 0x0;
};
```

Результат работы программы:

```
nikita@nikita-VirtualBox:~/files/work$ gcc __environ.c -o __environ
nikita@nikita-VirtualBox:~/files/work$ ./__environ
XDG_VTNR=7
LC_PAPER=ru_RU.UTF-8
LC_ADDRESS=ru_RU.UTF-8
XDG_SESSION_ID=c1
XDG_GREETER_DATA_DIR=/var/lib/lightdm-data/nikita
LC_MONETARY=ru_RU.UTF-8
CLUTTER_IM_MODULE=xim
SESSION=ubuntu
GPG_AGENT_INFO=/home/nikita/.gnupg/S.gpg-agent:0:1
TERM=xterm-256color
VTE_VERSION=4205
XDG_MENU_PREFIX=gnome-
SHELL=/bin/bash
QT_LINUX_ACCESSIBILITY_ALWAYS_ON=1
(...)
PWD=/home/nikita/files/work
JOB=dbus
XMODIFIERS=@im=ibus
GNOME_KEYRING_PID=
LANG=en_US.UTF-8
GDM_LANG=en_US
MANDATORY_PATH=/usr/share/gconf/ubuntu.mandatory.path
LC_MEASUREMENT=ru_RU.UTF-8
COMPIZ_CONFIG_PROFILE=ubuntu
IM_CONFIG_PHASE=1
GDMSESSION=ubuntu
SESSIONTYPE=gnome-session
GTK2_MODULES=overlay-scrollbar
SHLVL=1
HOME=/home/nikita
XDG_SEAT=seat0
LANGUAGE=en_US
(...)
LC_NAME=ru_RU.UTF-8
XAUTHORITY=/home/nikita/.Xauthority
_=./__environ
```

Функция доступа к переменным окружения *getenv* возвращает указатель на строку если в качестве аргумента указано имя существующей переменной окружения, если переменная отсутствует, возвращается NULL. Попробуем вывести несколько переменных окружения:

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

```
int main(int argc,char **argv) {
  // Получение параметра стреды, содержащего домашний каталог пользователя
  printf("Home directory: \"%s\"\n", getenv("HOME"));

  // Получение параметра стреды, содержащего имя пользователя
  printf("User name: \"%s\"\n", getenv("USER"));

  // Получение параметра стреды, содержащего каталог из которого запустили программу
  printf("Program directory: \"%s\"\n", getenv("PWD"));

  return 0x0;
};
```

Результат работы программы:

```
nikita@nikita-VirtualBox:~/files/work$ gcc getenv.c -o getenv
nikita@nikita-VirtualBox:~/files/work$ ./getenv
Home directory: "/home/nikita"
User name: "nikita"
nikita@nikita-VirtualBox:~/files/work$ gcc getenv.c -o getenv
nikita@nikita-VirtualBox:~/files/work$ ./getenv
Home directory: "/home/nikita"
User name: "nikita"
Program directory: "/home/nikita/files/work"
```

Функция *setenv* добавляет новую или изменяет существующую переменную окружения. Если третий аргумент функции не равен нулю, то переменная перезаписывается, в противном случае переменная остается нетронутой, *setenv* возвращает 0 при успешном завершении и -1, если произошла ошибка. Создадим новую переменную окружения и изменим несколько уже существующих:

```
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>

int main(int argc, char **argv) {
  // Перезаписываем параметр стреды, содержащий домашний каталог пользователя
  // Третий аргумент отличен от нуля, параметр успешно перезапишется
  setenv("HOME", "/something", 1);
  printf("Home directory: \"%s\"\n", getenv("HOME"));

  // Перезаписываем параметр стреды, содержащий имя пользователя
  // Третий аргумент равен нулю, параметр не перезапишется
  setenv("HOME", "username", 0);
  printf("User name: \"%s\"\n", getenv("USER"));

  // Создаем новый параметр стреды, содержащий имя пользователя
  setenv("MY_SUPER_PARAMETR", "SUPER", 0);

  // Выведем последний параметр среды, он должен быть равен только что созданному новому параметру
  int index = 0;
  while (__environ[index] != NULL)
    ++index;

  --index;

  printf("New parametr from __environ: \"%s\"\n", __environ[index]);
  printf("New parametr from getenv: \"%s\"\n", getenv("MY_SUPER_PARAMETR"));

  return 0x0;
};
```

Результат работы программы:

```
nikita@nikita-VirtualBox:~/files/work$ gcc setenv.c -o setenv
nikita@nikita-VirtualBox:~/files/work$ ./setenv
Home directory: "/something"
User name: "nikita"
New parametr from __environ: "MY_SUPER_PARAMETR=SUPER"
New parametr from getenv: "SUPER"
```

1.3 Список литературы

- Модификация окружения [Электронный ресурс]. — URL: http://www.clinuxworld.com/programming/453-setenv (дата обращения 30.10.2016).
- Работа с переменными окружения [Электронный ресурс]. — URL: http://www.firststeps.ru/linux/r.php?17 (дата обращения 30.10.2016).
- Структура процессов [Электронный ресурс]. — URL: http://khpi-iip.mipk.kharkiv.edu/library/extent/os/bach/bach06.html (дата обращения 30.10.2016).
- Состояние процесса [Электронный ресурс]. — URL: http://citforum.ru/operating_systems/bach/glava_57.shtml (дата обращения 30.10.2016).
- Организация процессов OS UNIX [Электронный ресурс]. — URL: http://cad.narod.ru/methods/os_unix/unibas/process.html (дата обращения 30.10.2016).
- Контекст процесса [Электронный ресурс]. — URL: http://citforum.ru/operating_systems/bach/glava_59.shtml (дата обращения 30.10.2016).
- Мануал getopt [Электронный ресурс]. — URL: https://www.opennet.ru/man.shtml?topic=getopt (дата обращения 30.10.2016).