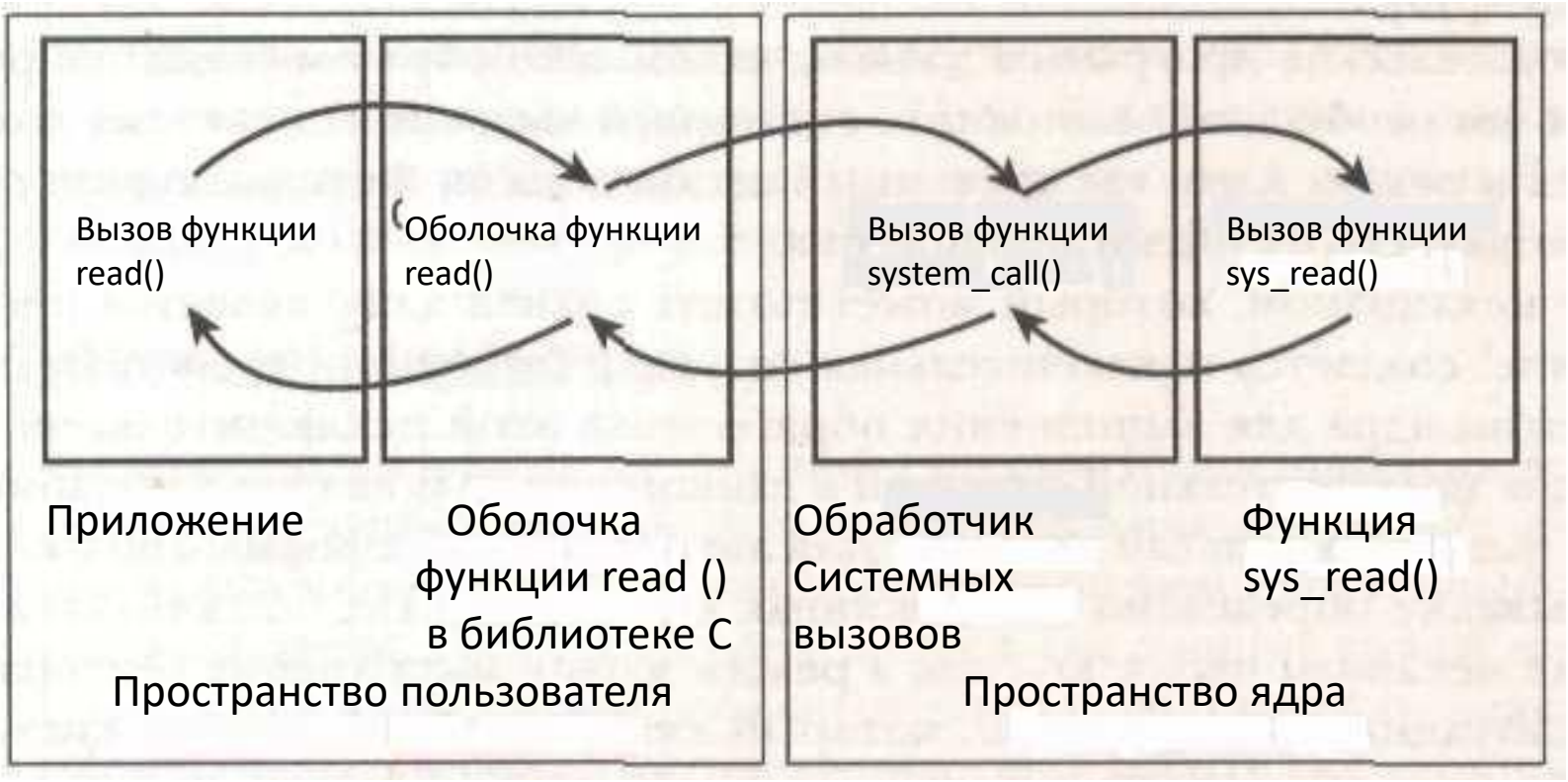
Последовательность выполнения системного вызова



Вход и выход из системного вызова

Приложения могут вызывать системные вызовы двумя различными способами:

- медленный вызов (с помощью инструкций int $0x80 и iret);
- быстрый вызов (с помощью инструкций sysenter и sysexit).

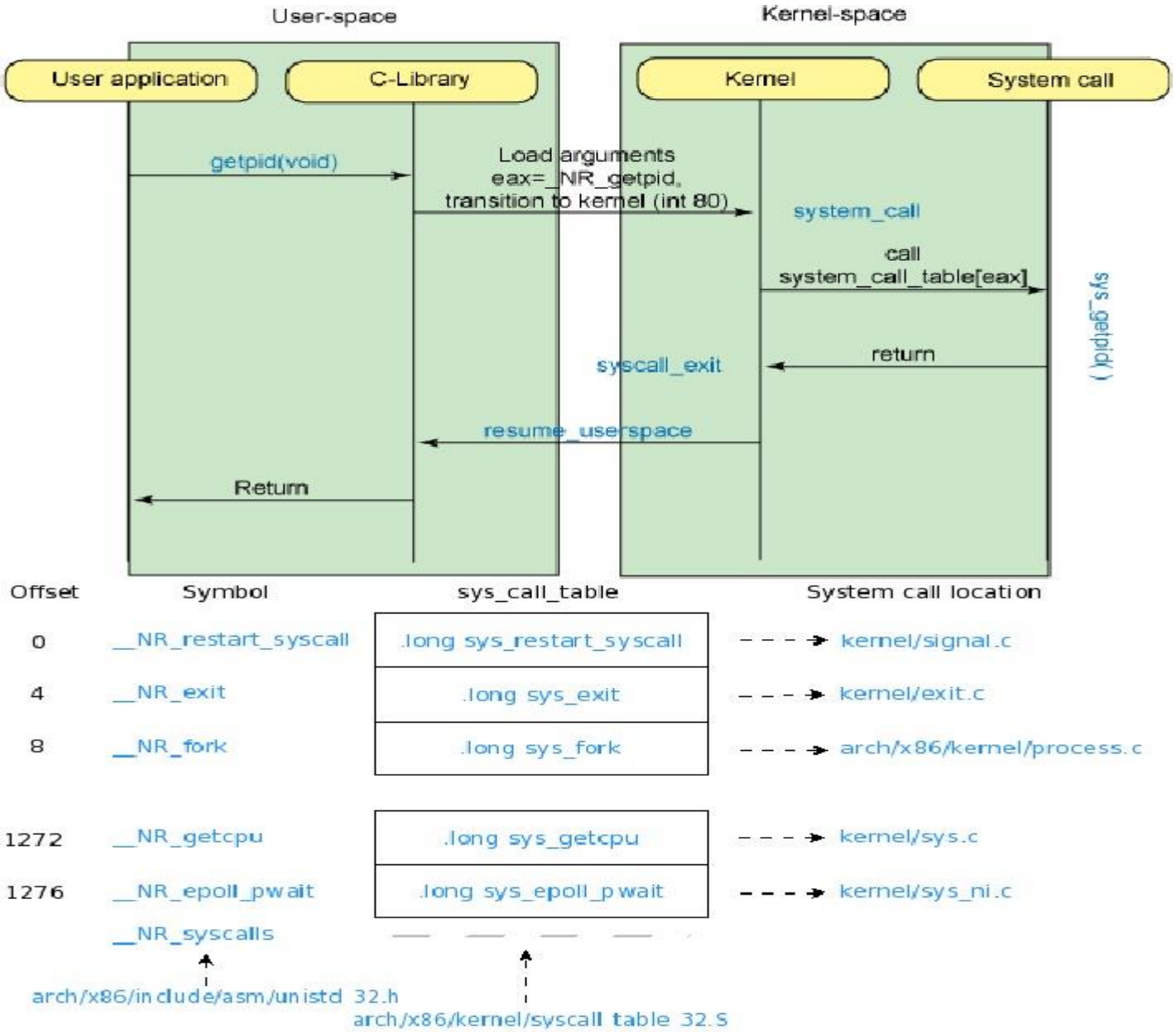
Номер системного вызова хранится в регистре eax

Параметры вызова хранятся в регистрах регистры ebx, ecx, edx, esi, edi или в памяти, если их больше 5

Дескриптор прерывания содержит:

- селектор сегмента — селектор сегмента кода ядра __KERNEL_CS;
- смещение — указатель на обработчик системного вызова system_call();
- тип — число 15, означающее, что исключение имеет тип «ловушка», не запрещает аппаратные прерывания;
- DPL (Descriptor Privilege Level, уровень привилегий дескриптора) — число 3, означающее, что процессам режима пользователя разрешено вызывать обработчик исключений

Таблица системных вызовов



Алгоритм запуска системной функции

алгоритм syscall {

найти запись в таблице системных функций, соответствующую указанному номеру функции;

определить количество параметров, передаваемых функции; скопировать параметры из адресного пространства задачи в пространство ядра;

сохранить текущий контекст для аварийного завершения; запустить в ядре исполняемый код системной функции;

если (во время выполнения функции произошла ошибка) { установить номер ошибки в регистре сохраненного регистрового контекста задачи;

} в противном случае занести возвращаемые функцией значения в регистр сохраненного регистрового контекста задачи;

}

Таймеры и управление временем

Системный таймер — это программируемое аппаратное устройство, которое генерирует аппаратное прерывание с фиксированной частотой *(*1000 Гц, 1 млсек*)*.

Обработчик этого прерывания, который называется *прерыванием таймера, запрещает аппаратные прерывания, обновляет значение* системного времени и выполняет периодические действия:

- Обновление значения времени работы системы (uptime).
- Обновление значения абсолютного времени (time of day).
- Для многопроцессорных систем выполняется балансировка очередей выполнения.
- Проверка, не израсходовал ли текущий процесс свой квант времени, и если израсходовал, то выполнятся планирование выполнения нового процесса.
- Выполнение обработчиков всех динамических таймеров, для которых истек период времени.
- Обновление статистики по использованию процессорного времени и других ресурсов.

Системные вызовы времени

Время задается с помощью типа time_t;

time_t **time**(time_t *tloc) - текущее календарное значение времени в секундах;

double **difftime**(time_t time1, time_t time2) -разность в секундах time1-time2;

struct tm ***gmtime**(const time_t timeval) – прямое преобразование времени в структуру

time_t **mktime**(struct tm *timeptr) – обратное преобразование времени

Struct tm {

int tm_sec Секунды, 0–61 (2 високосныt секунды)

int tm_min Минуты, 0–59

int tm_hour Часы, 0–23

int tm_mday День в месяце, 1–31

int tm_mon Месяц в году, 0–11

int tm_year Годы, начиная с 1900 г.

int tm_wday День недели, 0–6 (воскресенье соответствует 0)

int tm_yday День в году, 0–365

int tm_isdst Действующее летнее время

}

Системные вызовы времени

char ***ctime**(const time_t *timeval) - строковое представление даты и времени;

size_t **strftime**(char *s, size_t maxsize, const char *format, struct tm *timeptr) – форматированное представление даты и времени;

clock_t **clock** () - количество временных тактов, прошедших с начала запуска программы (в секундах clock () / CLOCKS_PER_SEC, количество тиков в секунду);

clock_t times(struct tms **buf*);

struct **tms** {

time_t tms_utime; число тактов выполнения процесса в режиме задачи

time_t tms_stime; число тактов выполнения процесса в режиме ядра

time_t tms_cutime; число тактов выполнения потомков в режиме задачи

time_t tms_cstime; число тактов выполнения потомков в режиме ядра

};

Программные интервальные таймеры

Установка интервальных таймеров

int setitimer(int *which***, const struct itimerval ****value***, struct itimerval ****ovalue***);**

which=

- **ITIMER_REAL** подает сигнал **SIGALRM**, когда значение таймера становится равным 0, уменьшается постоянно
- **ITIMER_VIRTUAL** подает сигнал **SIGVTALRM**, когда значение таймера становится равным 0, уменьшается во время работы программы процесса
- **ITIMER_PROF** подает сигнал **SIGPROF**, когда значение таймера становится равным 0, уменьшается во время работы программы и ядра

struct itimerval {
struct timeval it_interval; следующее значение таймера
struct timeval it_value; текущее значение таймера
};
struct timeval {
long tv_sec; секунды
long tv_usec; микросекунды
};

Значения таймеров уменьшаются от величины *it_value* до нуля, подается сигнал, и значения вновь устанавливаются равными *it_interval*.

Посылка процессу сигнала SIGALRM через *seconds* секунд

unsigned int alarm(unsigned int *seconds*);

Запуск периодических процессов с помощью интервальных таймеров

```
#include <sys/time.h>
#include <signal.h>
void F( )
{ /* функция, которая периодически запускается*/
}
main()
{ struct itimerval  A,B;
   struct sigaction act;
    /* установка реакции на сигнал SIGALRM*/
        act.sa_handler=F;
     sigemptyset(&act.sa_mask);
        act.sa_flags=0;
 sigaction(SIGALRM,&act,0);
 /* установка периода перезапуска программы*/
A.it_interval.tv_sec=0;
A.it_interval.tv_usec=100000; // период перезапуска 100 мс
A.it_value.tv_usec=0;
A.it_value.tv_sec=2; // первый запуск через 2 сек
setitimer(ITIMER_REAL, &A,  &B);
for (;;) pause(); /* бесконечный цикл ожидания  сигнала SIGALRM*/
}
```

Запуск периодических процессов с помощью сервиса CRON

Демон cron запускается процессом init в момент запуска системы. После запуска, он ежеминутно просматривает файл */etc/crontab*, каталог */etc/cron.d/* и каталог с пользовательскими таблицами заданий (*/var/spool/cron/crontabs*) , в которых содержатся информация о периодичности запуска команд и запускает команды, когда значения полей совпадают с текущим временем.

Начальный пользовательский конфигурационный файл создается командой **crontab** <*имя файла расписания*>.

Затем его можно отредактировать просмотреть и удалить соответственно командами **crontab *–e***, **crontab *–l*** и **crontab *-r***.

1-6/2 * * * * command запуск в 1,3 и 5 минуты

Задание 1 к лабораторной работе 6

Создайте пользовательский файл конфигурации сервиса *cron*, в котором содержаться команды периодического запуска одной из программ, разработанных в предыдущих лабораторных работах. Результаты работы этой программы должны выводиться или переадресоваться в файл.

Задание 2 к лабораторной работе 6

Напишите периодическую программу, в которой период запуска и количество запусков должны задаваться в качестве ее параметров. При каждом очередном запуске программа должна порождать новый процесс, который выводить на экран свой идентификатор, дату и время старта. Программа и ее дочерний процесс должны быть заблокированы от завершения при нажатии клавиш Ctrl/z. После завершения дочернего процесса программа должна вывести на экран информацию о времени своей работы и дочернего процесса.

Пример использования системных функций времени

```
#include <sys/types.h>
#include <sys/times.h>
extern long times();
main() { int i;
/* tms - имя структуры данных, состоящей из 4 элементов */
struct tms pb1,pb2; long pt1,pt2; pt1 = times(&pb1);
for (i = 0; i < 10; i++) if (fork() == 0) child(i);
for (i = 0; i < 10; i++) wait((int*) 0); pt2 = times(&pb2);
printf("процесс-родитель: реальное время %u в режиме задачи %u в режиме
    ядра %u потомки: в режиме задачи %u в режиме ядра %u\n", pt2 -
    pt1,pb2.tms_utime - pb1.tms_utime, pb2.tms_stime - pb1.tms_stime,
    pb2.tms_cutime - pb1.tms_cutime, pb2.tms_cstime - pb1.tms_cstime);
}
child(int n) { int i; struct tms cb1,cb2; long t1,t2; t1 = times(&cb1);
for (i = 0; i < 10000; i++) ; t2 = times(&cb2);
printf("потомок %d: реальное время %u в режиме задачи %u в режиме ядра
    %u\n",n,t2 - t1, cb2.tms_utime - cb1.tms_utime, cb2.tms_stime - cb1.tms_stime);
    exit();
}
```