Прикладные физико-технические и компьютерные методы исследований

Семинар 10

https://dl.dropboxusercontent.com/u/96739039/infa_s10.pdf

Memory mapped files

- Отображать в виртуальное адресное пространство процесса можно не только части RAM, но и файлы из HDD.
- Можно использовать в качестве аналога разделяемой памяти.

Memory mapped files (+)

- более быстрый доступ к памяти, т.к. нет необходимости считывать результат в промежуточный буфер (системные вызовы read, write).

http://stackoverflow.com/questions/192527/what-are-the-advantages-of-memory-mapped-files

- Удобно при необходимости (Random Access)
- Возможность работы с очень большими файлами («ленивая» загрузка, нет проблемы фрагментации)

Memory mapped files (-)

- Размер отображаемого файла зависит от архитектуры: 32-битные ОС не могут отображать файлы > 4Gb.
- *False sharing*: помимо кэшей процессора есть ещё *page cache.* Если несколько процессов одновременно меняют один файл, и у одного из них какая-то страницы была в кэше, то перед внесением изменения необходимо сначала синхронизовать кэши, что накладно.
- Проблемно работать в многопоточном/многопроцессном режиме.

Применение

- Загрузка процессов в память (исполняемого кода).
- Доступ из нескольких процессов к одному участку памяти.

Пример использования

https://dl.dropboxusercontent.com/u/96739039/s10e01.c

Упражнение 1

- Дан большой текстовый файл в несколько десятков Mb (можно сгенерировать программно).
- Зашифровать его, циклически сдвинув каждый символ на N в таблице ASCII.
- Сравнить время работы в случае использования системных вызовов read/write или ф-й scanf/printf.

Прерывания

- Аппаратные прерывания (непрерывный опрос состояния устройства – polling…накладно, если скорости работы процессора и устройства сильно отличаются) + дополнительная линия.
- Исключения
- Программные прерывания

Прерывания

- Перед началом обработки прерывания сохраняется текущее состояние.
- Происходят *асинхронно,* т.е. могут произойти в любой момент исполнения программы
- Могут возникнуть только ***между*** выполнением процессором некоторых команд.

Исключения

- Возникают ***во время выполнения*** процессором некоторой команды (деление на 0, обращение к отсутствующей странице памяти)

Программные прерывания

- Возникают ***после выполнения*** специальных команд, как правило, для выполнения привилегированных действий внутри системных вызовов. Возникают абсолютно предсказуемо.

Сигналы

- Обработка аппаратных прерываний выполняется ОС (пользователю не доверяют…)
- Обработка исключений и программных прерываний ложится на пользователя через механизм сигналов.

Сигналы

- Типы сигналов: 1 … 31

Процесс может получить сигнал от:

- Hardware (при возникновении исключительной ситуации)
- Другого процесса, выполнившего системный вызов передачи сигнала
- Операционной системы (при наступлении некоторых событий)
- Терминала (при нажатии определенной комбинации клавиш)
- Системы управления заданиями (при выполнении команды **kill** - мы рассмотрим ее позже)

Реакция на сигналы

- Принудительно проигнорировать сигнал (*9 – SIGKILL нельзя проигнорировать*).
- Произвести обработку по умолчанию (проигнорировать, остановить процесс (перевести в состояние ожидания до получения другого специального сигнала), либо завершить работу с образование core файла или без него).
- Выполнить обработку сигнала, специфицированную пользователем

Кто кому может послать сигнал?

- В реальности довольно громоздкая система правил вводящая понятия *группы процессов, сеансов, лидеров группы* и т.д.
- Мы для простоты рассмотрим только очень частный случай, когда обмениваться сообщениями могут только родственные процессы.

Как отправить сигнал

- Из терминала **kill [-signal] [--] pid**
- Из программы

```
#include <sys/types.h>
#include <signal.h>

int kill(pid_t pid, int signal);
```

Изменение поведения процесса при получении сигнала

- ***void signal(SIGINT, SIG_IGN);***

Игнорировать сигнал SIGINT (отправляется при нажатии ctrl+c из терминала).

- **SIG_DFL –** вернуть значение по умолчанию.
- Функция пользователя

```
void* my_handler(int nsig) {
        <обработка сигнала>
        // nsig – номер сигнала, т.е. одну ф-ю можно использовать
        // для обработки нескольких сигналов
}

int main() {
        ...
        (void)signal(SIGHUP, my_handler);
        ...
}
```

Пример 1

```
#include <signal.h>

int main(void){
        /* Выставляем реакцию процесса на
сигнал SIGINT на      игнорирование */

        (void)signal(SIGINT, SIG_IGN);

        /*Начиная с этого места, процесс будет
игнорировать          возникновение сигнала SIGINT */

        while(1);
        return 0;
}
```

Упражнения 2

- Добиться того, чтобы программа не реагировала на внешние воздействия, т.е. и на сигнал SIGQUIT.
- Задать пользовательский обработчик сигнала.

P.S.

Чтобы посмотреть список запущенных процессов в терминале набираем ps –ej

В новом терминале узнаём pid и отправляем ему SIGKILL из консоли.

Упражнение 3 (РТ/ФУПМ - чемпион)

- Два процесса <-> две трибуны, скандирующие <факультет> (1 процесс), потом «чемпион» (2 процесс), потом снова 1й процесс и т.д.
- Вторая трибуна должна кричать только после того, как прокричит первая и наоборот. Синхронизация трибун производится через сигналы SIGUSR1, SIGUSR2.
- Смоделировать скандирование через родственные процессы.

Упражнение 4 (домашнее)

- Реализовать побитовую передачу информации между процессами с помощью SIGUSR1, SIGUSR2.