Прикладные физико-технические и компьютерные методы исследований

Семинар 4

https://dl.dropboxusercontent.com/u/96739039/infa_s04.pdf

Прошлый семинар

- Типы контекстов процесса
- Создание процесса – fork()
- **pid_t wait(int* status);**
- Побитовые операции x & 255, x % 256, x >> 1
- Изменение пользовательского контекста процесса – системный вызов ***exec***

Средства коммуникации процессов

- Сигнальные
- Канальные (потоковая модель, сообщения)
- Разделяемая память

Отличие между fprintf, fscanf и read, write

- Потоковая коммуникация может быть осуществляться между процессом и диском (файлом).
- В fprintf, fscanf есть некоторая структура сообщений.
- В linux fprintf, fscanf реализованы с помощью read, write

Файловый дескриптор

Таблица файловых дескрипторов (часть системного контекста):

- информация об используемых процессом файлах
- информация о потоковых линиях связи, соединяющих разные процессы.

индекс этого массива = файловый дескриптор

Некоторые файловые дескрипторы сразу ассоциируются со стандартными потоками ввода-вывода:

0 - поток ввода (из терминала)

1 - поток вывода (в терминал)

2 - поток для вывода ошибок

Как открыть файл?

Системный вызов open возвращает файловый дескриптор , либо -1.

```
#include <fcntl.h>
int open(char* path, int flags, int mode);
flags = O_RDONLY O_WRONLY O_RDWR
O_CREAT O_EXCL O_APPEND
mode & ~umask
```

Чтение из файла

size_t read(int fd, void* addr, size_t nbytes)

- годится для файлов, pipe , FIFO и socket
- что возвращает и что принимает?
- если для файла read вернул 0, то файл прочитан до конца

Закрытие файла

int close(int fd);

- Сброс информации из буфера
- Удаление fd из таблицы файловых дискрипторов.

Пример использования write

- http://acm.mipt.ru/twiki/bin/view/Cintro/WebHome/osstud.zip либо
- https://dl.dropboxusercontent.com/u/96739039/s03ex.zip

Упражнение 1

Написать программу, считывающую файл и выводящую результат на экран кусками по N символов.

Pipe

- pipe (в отличие от файла нельзя читать повторно) реализован в виде циклического буфера

```
int fd[2];
pipe(fd);
```

- fd[0] - выход, только чтение
- fd[1] - вход, только запись
- общаться могут только родственные процессы
- к памяти pip’a нельзя обратиться напрямую, т.к. она находится в адресном пространстве ОС.

Пример использования Pipe

- Из учебника

Pipe между процессом-родителем и процессом-потомком

- Из учебника

Особенности read/write для pipe

Из-за ограниченности размера и задержек чтения/записи **может прочитаться или записаться информации меньше**, чем вы хотите.

Особенность блокирующего системного вызова read:

если pipe пустой, и есть открытые на запись концы у какого-нибудь родственного процесса, то системный вызов ждет появления в pipe какой-либо информации, поэтому в примерах закрывали неиспользуемый конец pipe.

Аналогично для write, если нет процессов открытых для чтения.

Упражнение 2

- Определить размер pipe пользуясь особенностями read/write

Упражнение 3. Двунаправленная связь между родственными процессами

- Почему сложно обойтись одним pipe?
- Используем два разных pipe