**Bài 1: File in/out**

**Mô hình Universality I/O:** tất cả các file đều có thể open,read,write, quy định các lập trình viên phải tạo ra các file system và device driver có system call input/output

**I. Thao tác file trong Linux**

**-int open(pathname, flag, mode)**

**-int close(fd);**

**- size\_t read(fd,void\* buffer, count):**

**-wrie**

**II- File I/O Buffering**

**-**Kernel buffer giúp hệ thống chạy nhanh hơn thay vì đọc, ghi lên tục trên ô cứng

- Đồng bộ kerel buffer theo 2 cơ chế Synchronizied I/O data/file

- Các syscall đồng bộ:

+ int fsync();

+int fdatasync

+void sync:

+ Cờ O\_SYNC or ) O\_DSYNC

**III- Con trỏ file**

- Mỗi file đang mở, kernel duy trì 1 giá trị file offset, là vị trí hiện tại file mà lời gọi read, write tiếp theo sẽ bắt đầu.

**System call đổi file offset**

-**off\_t lseek(int fd,off\_t offset, whence)** : thay đổi file offset 1 giá trị offset byte tính từ điểm whence

+whence: có 3 giá trij SEEK\_SET, SEEK\_CUR, SEEK\_END;

+ nó chỉ thay đổi giá trị offset lưu ở kernel file đó, ko truy xuất đến bộ nhớ vật lý

+ KO dùng cho file đặc biệt pipe, FIFO, socket và terminal`

**IV- I/O Multiplexing -select**

**-** Với đọc ghi I/o thông thường, thì system call I/O sẽ block cho đến khi có dữ liệu=> khi truy xuất quản lý nhiều file, sẽ ko truy xuất đc các file khác khi đang chờ đọc 1 file

- 2 hướng giải quyết này:

+ Nonbloking: thêm cờ O\_NONBLOCK. Nếu file chưa sẵn sàng=> trả giá trị lỗi, sẽ làm việc khác và quay lại thăm dò sau. Nếu thăm dò ít, => thời gian trễ thao tác file sẽ dài, nếu quá dầy thì lại tốn CPU

+ Tạo threading thăm dò: từ tiến trình cha. => số file lớn, số thread lớn=> phức tạp tốn tài nguyên

**=> I/O mutiplexing ra đời giải quyết: poll, select**

**1. Select**

**-**Gom tất cả các file muốn theo dõi vào 1 tập sẽ block chương trình cho đến khi 1 hoặc nhiều file sẵn sàng, trả về số file sẵn sàng, 0 nếu timout và -1 lỗi

+ 3 tập read,write,except

+đối số nfds: số file lớn nhất trong 3 tệp +1

**2. Poll**

**Bài 2: Tiến trình**

**1. Thao tác tiến trình con**

**- Tạo mới: fork() trả về**

+0 đang ở tiến trình con

+>0 tiến trình cha

+(-1) lỗi

Sau khi copy thì các phần vùng giữa cha, con có mỗi liên hệ như sau:

+ text segment( chưas mã thực thi): đc dùng chung

+ head, stack: theo kĩ thuật copy on write

- **Chạy chương trình mới**

**+ các lệnh exec..**

**- Kết thúc tiến trình: exit()**

**2. Quản lý tiến trình con**

**-Hàm wait: trả về pid của iến trình con kết thúc hoặc -1 nếu lỗi**

**+** block cho đến khi 1 tiến trình con bị kết thúc

**3. Lập lịch**

**-**Phân loại theo đặc điểm hoatj động : **I/O Bound process vs Processor -Bound process**

- Phan loai theo : **Realtime và Normal(interactive vs batch process)**

**Prioty trong Linux có 1 bang duy nhat từ 0-139 vs:**

+ 0-99: cho realtime process (realtime value)

+ -20 →19 cho normal process( nice value)

**- Giá trị PR= NI+20, realtime->rt**

**-Kernal priority= 100+20+NI**

**2. Scheduling Policy**

**-Giúp OS quyết định xem tiến trình đc chạy bao lâu và khi nào nhường quyền cho đứa khác**

**-** Đc chia làm 2 loại

+ Với realtime: **SCHED\_RR vs SCHED\_FIFO**

**+** Với normal: SCHED\_NORMAL, SCHED\_BATCH, SCHED\_IDLE

Bộ lập lịch chung **Completely Fair Scheduler(CFS)**

**Bài 4: Thread**

**1. Tạo mới thread**

**- pthread\_create: trả về 0 nếu thành công, còn lại là lỗi**

**- pthread\_exit(NULL): thoát threading**

**2. Quản lý thread**