I. File I/O

BT1. Viết một chương trình mở một file bất kì và sử dụng cờ O_APPEND, sau đó thực hiện seek về đầu file rồi ghi một vài dữ liệu vào file đó. Dữ liệu sẽ xuất hiện ở vị trí nào của file và tại sao lại như vậy?

➔ **Trả lời:**

Dữ liệu sẽ xuất hiện ở cuối file, do khi mở file bằng O_APPEND, tất cả dữ liệu sẽ được ghi vào cuối file bất kể file poiter đang trỏ đến chỗ nào. Nếu thực hiện seek từ đầu file, nó sẽ lấy vị trí file poiter mới làm đầu file tức là vị trí kết thúc của file cũ.

BT2. Sau mỗi lệnh write dưới đây, Cho biết nội dung sẽ được ghi vào file nào, nội dung là gì và giải thích tại sao?

```
fd1 = open(file, O_RDWR | O_CREAT | O_TRUNC, S_IRUSR | S_IWUSR);

fd2 = open(file, O_RDWR);

fd3 = open(file, O_RDWR);

write(fd1, "Hello,", 6);

write(fd2, "world", 6);

lseek(fd2, 0, SEEK_SET);

write(fd1, "HELLO,", 6);

write(fd3, "Gidday", 6);
```

➔ **Trả lời:**

Các cờ trong mã code:

+ S_IRUSR: 00400 - user has read permission
+ S_IWUSR: 00200 - user has write permission

+ O_TRUNC: Nếu file đã tồn tại, là file thông thường và chế độ truy cập cho phép ghi (tức là O_RDWR hoặc O_WRONLY) nó sẽ xóa hết dữ liệu của file. Nếu file là FIFO hoặc terminal device file, cờ O_TRUNC bị bỏ qua.
+ SEEK_SET: di chuyển vị trí con trỏ file lên đầu file.

Chương trình thực hiện theo các bước sau:

write(fd1, "Hello,", 6);	Tạo file, ghi "Hello," vào file.
write(fd2, "world", 6);	File lúc này đã tồn tại, có quyền đọc ghi nên O_TRUNC sẽ xóa tất cả dữ liệu của file ("Hello,") và ghi vào file nội dung "world".
lseek(fd2, 0, SEEK_SET);	Offset = 0, di chuyển con trỏ lên vị trí đầu file.
write(fd1, "HELLO,", 6);	File lúc này đã tồn tại, có quyền đọc ghi nên O_TRUNC sẽ xóa tất cả dữ liệu của file ("world") và ghi vào chữ "HELLO,".
write(fd3, "Gidday", 6);	Mở file với offset = 0, chữ Gidday được ghi vào đầu file.

➔ Đầu ra: **"GiddayHELLO,"**

BT3. Viết một chương trình có só lượng command-line arguments là 3, Có dạng như sau:

$ ./example_program filename num-bytes [r/w] "Hello"

Trong đó:

1. example_grogram: Tên file thực thi
2. filename: Tên file
3. num-bytes: Số byte muốn read/write
4. [r/w]: r -> Thực hiện đọc từ filename và in ra màn hình

w -> Thực hiện ghi vào filename

5. "Hello": Nội dung bất kì muốn read/write vào filename

II. Process

BT1. Giả sử rằng một parent process đã thiết lập một handler cho SIGCHLD và cũng block tín hiệu này. Sau đó, một trong các child process của nó thoát ra và parent process sau đó thực hiện wait() để thu thập trạng thái của child process. Điều gì xảy ra khi parent process bỏ chặn SIGCHLD? Viết một chương trình để xác minh câu trả lời.

➔ **Trả lời:**

Sau khi bỏ chặn SIGCHLD, signal được gửi tới parent process.

BT2. Điều gì xảy ra khi chúng ta sử dụng họ hàm execute (execl/exevp/exepv)? Giải thích và viết một chương trình để xác minh câu trả lời. Sau đó hãy cho biết bản chất khi lệnh system() hoạt động như thế nào.

Họ hàm execute thay thế process đang chạy hiện tại thành một process mới, chiếm một số tài nguyên của process cũ. Khi sử dụng các hàm execute, hệ thống sẽ thay thế vùng virtual memory mapping của process hiện thành thành của process mới, copy các data argv và envp đã được nhập trong execute vào vùng virtual memory mapping mới.

system (command): tạo ra một tiến trình con gọi một lệnh shell để thực thi lệnh.

- B1: Kiểm tra command = NULL -> shell không tồn tại, trả về ":" – không làm gì cả.
- B2: Block SIGCHLD, Ignore SIGINT và SIGQUIT -> tạo tiến trình con bằng lệnh fork.
  - childpid trả về -1: Lỗi
  - trả về 0: tiến trình con là exec command. Lúc này sử dụng hàm execl để thực hiện lệnh command. Nếu execl không gọi được tiến trình con thì thoát ra ngoài.
  - khác: tiến trình cha đang đợi tiến trình con chấm dứt. (waitpid)
- B3: Unlock SIGCHLD và khôi phục lại SIGINT, SIGQUIT.

BT3. Trình bày cách sử dụng 2 hàm dup()/dup2(). Viết chương trình minh họa cho câu trả lời.

➔ **Trả lời:**

int dup (int oldfd); / oldfd: old file descriptor whose copy is to be created. */*

dup () tạo ra một bản sao file descriptor. Nó sử dụng bộ descriptor chưa sử dụng được đánh số thấp nhất cho descriptor mới. Nếu bản sao được tạo thành công, thì file descriptor gốc và bản sao có thể được sử dụng thay thế cho nhau. Cả hai đều tham chiếu đến cùng một file descriptor đang mở và do đó dùng chung các cờ trạng thái file và các offset.

int dup2(int oldfd, int newfd);

/ oldfd: old file descriptor*

*newfd new file descriptor which is used by dup2() to create a copy. */*

Nếu descriptor *newfd* đang được mở thì nó sẽ trở về trạng thái đóng trước khi được mở ra vào lần tiếp theo. Nếu oldfd không phải là một file descriptor hợp lệ, khi lệnh gọi xảy ra lỗi, newfd sẽ không được đóng lại. Ngược lại, nếu oldfd hợp lệ và newfd = oldfd thì dup2() sẽ không làm gì cả và hệ thống return newfd.

BT4. Debug là một công việc quan trọng trong việc lập trình do đó hãy tìm hiểu về segmentation fault, core dumped và cho biết chúng là gì? Viết một chương trình tái hiện lại lỗi này. Sau khi tái hiện được lỗi, tìm hiểu về gdb và trình bày các bước fix cho lỗi này.

➔ **Trả lời:**

Khi chương trình chạy, nó có quyền truy cập vào một số phần trong memory. Biến cục bộ thì lưu trong stack; các hàm cấp phát động (malloc ()) thì được lưu trong heap. Nếu ta truy cập ngoài các khu vực vùng nhớ này - > xảy ra lỗi ***segmentation fault***.

Có một số lỗi dẫn đến segmentation fault là: tham chiếu NULL, tham chiếu con trỏ chưa được khởi tạo, tham chiếu một con trỏ đã được giải phóng, vượt ra khỏi vùng nhớ cho sẵn, cố viết vào cuối một mảng hoặc là tràn ngăn xếp (stackoverflow).

Core dumped: Lỗi xảy ra do truy cập vùng nhớ không được phép truy cập, như khi một đoạn mã cố gắng thực hiện thao tác đọc và ghi ở vị trí chỉ đọc trong bộ nhớ hoặc khối bộ nhớ đã được giải phóng.

GDB cho phép ta xem những gì đang diễn ra trên 'bên trong' một chương trình trong khi nó thực thi.

- Lấy thông tin (a query processor) như trạng thái các thanh ghi, địa chỉ các ô nhớ mà CPU đang xử lý ở 1 thời điểm nào đó.
- Ánh xạ symbol (a symbol resolver): ánh xạ trạng thái tại 1 điểm đang chạy đến vị trí trên source code tương ứng (file nào, dòng nào). Đây là chức năng được dùng phổ biến nhất, giúp developer dễ dàng xác định đoạn source code chưa đúng.
- Một trình thông dịch biểu thức (a expression interpreter) chuyển các biểu thức do developer viết ra thành điều kiện hay action tương ứng tác động lên chương trình đang chạy.
- Dừng có điều kiện thông qua các breakpoint.
- Sửa trạng thái chương trình như thay đổi giá trị các biến, các thanh ghi rồi cho phép chạy tiếp với trạng thái mới (một số trường hợp giúp nhảy qua các đoạn lỗi hoặc chưa ổn định).

Ví dụ: Trong chương trình trên, người dùng cố gán giá trị cho một con trỏ NULL. Điều này sẽ xảy ra lỗi.

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

int main ()
{
    int *p = NULL;
    *p = 1;

    printf("%d",*p);

    return 0;
}
```

```
vkahhh@ubuntu:~/dzs/topic6$ gcc -o -Wextra gdb_example gdb_example.c
vkahhh@ubuntu:~/dzs/topic6$ ./gdb_example
Segmentation fault (core dumped)
```

(Sử dụng cờ -Wextra cũng không có warning)

```
vkahhh@ubuntu:~/dzs/topic6$ gcc -o -g gdb_example gdb_example.c
vkahhh@ubuntu:~/dzs/topic6$ gdb gdb_example
GNU gdb (Ubuntu 8.1.1-0ubuntu1) 8.1.1
Copyright (C) 2018 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from gdb_example...(no debugging symbols found)...done.
(gdb)
```

```
(gdb) r
Starting program: /home/vkahhh/dzs/topic6/gdb_example

Program received signal SIGSEGV, Segmentation fault.
0x000055555555465e in main () at gdb_example.c:8
8           *p = 1;
(gdb) bt
#0  0x000055555555465e in main () at gdb_example.c:8
(gdb) frame 0
#0  0x000055555555465e in main () at gdb_example.c:8
8           *p = 1;
(gdb) print p
$1 = (int *) 0x0
```

⇨ Lỗi ở dòng thứ 8 (*p = 1)

III. Thread

BT1. Viết một chương trình để chứng minh rằng các thread khác nhau trong cùng một process có thể có các tập hợp signal đang chờ xử lý (set of pending signal) khác nhau,được trả về bằng sigpending(). Bạn có thể làm điều này bằng cách sử dụng pthread_kill() để gửi các tín hiệu khác nhau đến hai thread khác nhau đã bị block các tín hiệu này, và sau với mỗi thread gọi sigpending() hãy hiển thị thông tin về các tín hiệu đang chờ xử lý.

IV. Signal

BT1. Viết chương trình block tín hiệu SIGINT và sau đó in ra signal mask của process hiện tại.

BT2. Realtime signal và standard signal là gì? Phân biệt sự khác nhau giữa chúng.

➔ Trả lời:

- Standard signal: được kernel sử dụng để xử lý các sự kiện của tiến trình, được đánh số từ 1 đến 31 và có thêm các signal như SIGCHLD, SIGTERM, ... Trong standard signals không có đối số, thông điệp hay thông tin đi kèm nào. Standard signals cũng không thể xử lý hàng đợi, một signal chỉ có thể pending (sau đó được gửi đi) 1 lần.

- Realtime signal: được đánh số từ 32-64. Linux cho phép gửi nhiều yêu cầu của 1 Realtime signal cho cùng 1 process, các signal sẽ được gửi nhiều lần. Khi gửi một realtime signal, ta có thể có thể đính kèm với tín hiệu một dữ liệu, có thể là một số nguyên hoặc giá trị con trỏ. Signal handler ở process nhận sẽ truy xuất dữ liệu này. Nếu nhiều realtime signal được pending, signal có số nhỏ nhất sẽ được ưu tiên gửi đi trước. Realtime signal không được đánh số như standard signal mà được viết kiểu: (SIGRTMIN + n) -> realtime signal thứ n + 1.

So sánh:

Standard Signal	**Realtime Signal**
Không có dữ liệu đi kèm	Được đính kém dữ liệu
1 signal được pending 1 lần	Nhiều signal được pending và ưu tiên gửi theo thứ tự từ số nhỏ nhất đến lớn nhất
Chỉ có SIGUSR1 và SIGUSR2 hỗ trợ application-defined.	Nhiều signal hỗ trợ

V. Pipe

BT1. Viết một chương trình sử dụng hai pipe để cho phép giao tiếp hai chiều giữa hai tiến trình cha và con. Parent process lặp lại việc đọc dữ liệu văn bản từ bàn phím và sử dụng một trong các pipe để gửi văn bản đến child process, child process chuyển nó thành chữ hoa và gửi lại cho parent process qua pipe còn lại. Parent process đọc dữ liệu trả về từ child process và in ra màn hình trước khi lặp lại quá trình một lần nữa.

VI. Shared Memory

BT1. Viết một chương trình sử dụng shared memory segment để gửi đi dữ liệu là các cặp tên-giá trị (name-value). Bạn sẽ cần cung cấp một vài API cho phép tạo tên mới, sửa đổi tên hiện có, xóa tên hiện có và truy xuất giá trị được liên kết với tên.