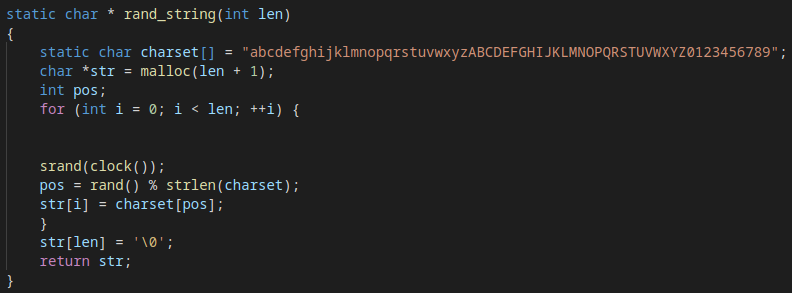
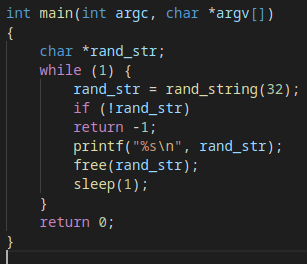
**VD1:**

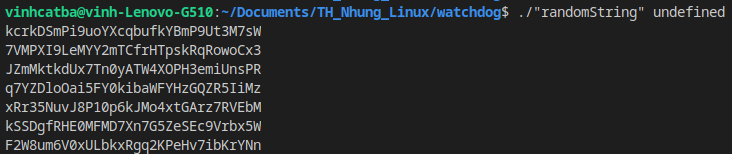
- Hàm rand\_string trả về con trỏ kiểu char, tham số truyền vào là độ dài chuỗi. Hàm này có tác dụng là tạo ra một chuỗi ngẫu nhiên có độ dài là len, lưu chuỗi đó vào con trỏ str và trả về str. Quá trình tạo chuỗi ngẫu nhiên là tạo ra 1 ví trí ngẫu nhiên từ chuỗi gốc charset, sau đó gán kí tự ở vị trí đó của chuỗi gốc vào chuối mới.



- Hàm main thực hiện tạo chuỗi ngẫu nhiên qua gọi hàm rand\_string, gán vào biến con trỏ rand\_str, in ra nó và giải phóng. Sau đó đợi 1s và thực hiện lặp lại.



Biên dịch và chạy:



**VD2:**

- code chương trình tham khảo từ phụ lục 1 giáo trình.

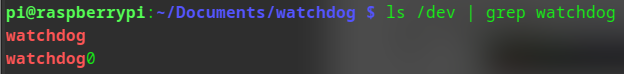
- Do máy laptop ubuntu không có watchdog nên chương trình được chạy trên raspberry pi.

B1: SSH vào ras:

ssh pi@raspberrypi.local

B2: Kiểm tra xem ras có watchdog timer không:

ls /dev | grep watchdog

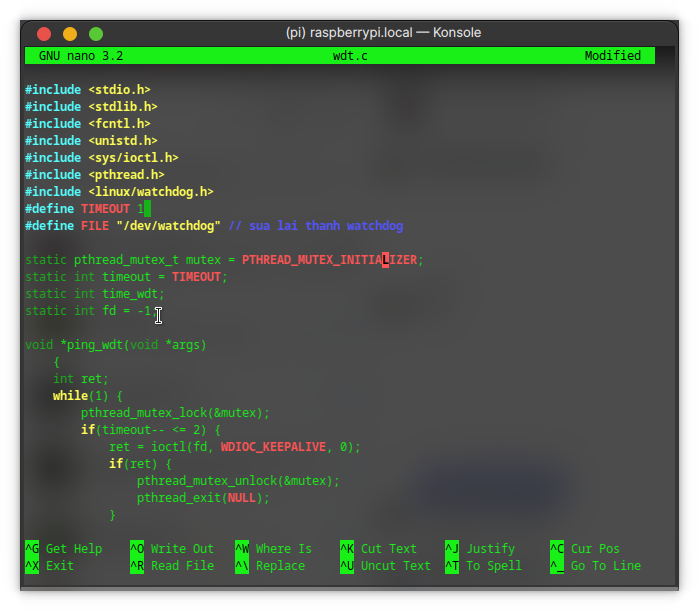


Có 2 watchdog timer là watchdog và watchdog0

B3: tạo file .c và viết code:

cd ~/Documents && mkdir watchdog && cd watchdog  
nano wdt.c

Sau đó viết code. Code được tham khảo từ phụ lục 1 giáo trình, sửa #define FILE thành “dev/watchdog” thay vì “dev/watchdog1” ban đầu. Ấn Ctrl + X → Y → Enter để lưu và thoát.

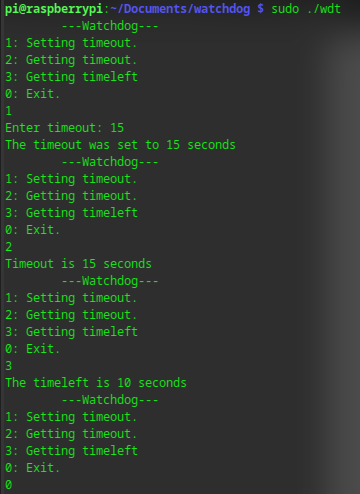


B4: Compile chương trình, sử dụng cờ -lpthread do chương trình có dùng pthread:

gcc -o wdt wdt.c -lpthread

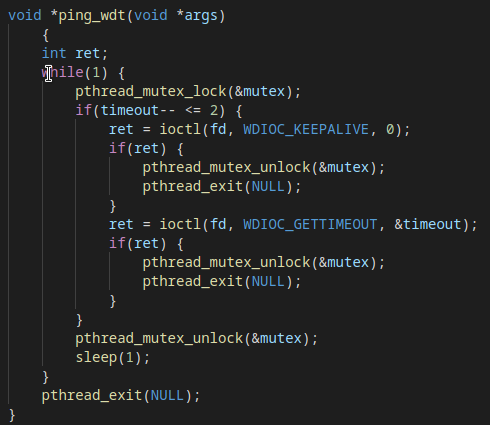
B5: Chạy chương trình với quyền sudo

sudo ./wdt

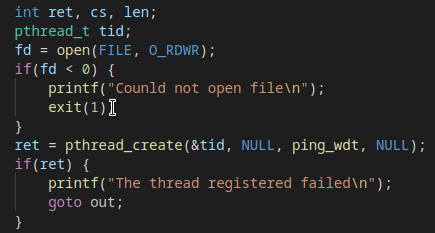


Giải thích code:

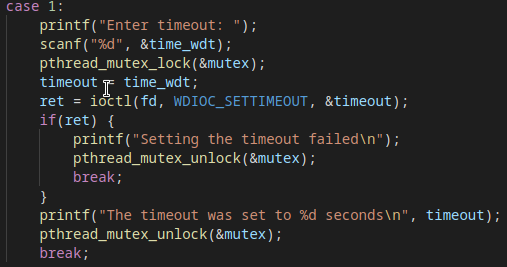
- Hàm ping\_wdt liên tục lấy timeout từ watchdog timer, nếu timeout < 2 thì gọi system call ioctl(fd, WDIOC\_KEEPALIVE, 0) để restart wdt.



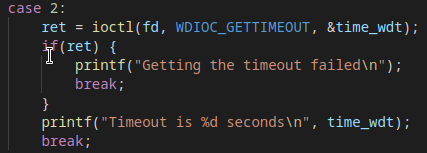
- Hàm main mở file /dev/watchdog và tạo thread ping\_wdt



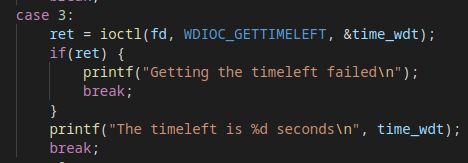
- case 1: Sử dụng system call ioctl(fd, WDIOC\_SETTIMEOUT, &timeout) để set timeout cho wdt qua biến timeout.



- case 2: Sử dụng system call ioctl(fd, WDIOC\_GETTIMEOUT, &time\_wdt) để lấy timeout hiện tại và cho vào biến time\_wdt, rồi in ra màn hình



- case 3: Sử dụng system call ioctl(fd, WDIOC\_GETTIMELEFT, &time\_wdt) để lấy thời gian còn lại của wdt, lưu vào biến time\_wdt và in ra màn hình



Code VD1:

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <time.h>

static char \* rand\_string(int len)

{

static char charset[] = "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789";

char \*str = malloc(len + 1);

int pos;

for (int i = 0; i < len; ++i) {

srand(clock());

pos = rand() % strlen(charset);

str[i] = charset[pos];

}

str[len] = '\0';

return str;

}

int main(int argc, char \*argv[])

{

char \*rand\_str;

while (1) {

rand\_str = rand\_string(32);

if (!rand\_str)

return -1;

printf("%s\n", rand\_str);

free(rand\_str);

sleep(1);

}

return 0;

}

Code VD2:

#include <stdio.h>

#include <stdlib.h>

#include <fcntl.h>

#include <unistd.h>

#include <sys/ioctl.h>

#include <pthread.h>

#include <linux/watchdog.h>

#define TIMEOUT 1

#define FILE "/dev/watchdog1"

static pthread\_mutex\_t mutex = PTHREAD\_MUTEX\_INITIALIZER;

static int timeout = TIMEOUT;

static int time\_wdt;

static int fd = -1;

void \*ping\_wdt(void \*args)

{

int ret;

while(1) {

pthread\_mutex\_lock(&mutex);

if(timeout-- <= 2) {

ret = ioctl(fd, WDIOC\_KEEPALIVE, 0);

if(ret) {

pthread\_mutex\_unlock(&mutex);

pthread\_exit(NULL);

}

ret = ioctl(fd, WDIOC\_GETTIMEOUT, &timeout);

if(ret) {

pthread\_mutex\_unlock(&mutex);

pthread\_exit(NULL);

}

}

pthread\_mutex\_unlock(&mutex);

sleep(1);

}

pthread\_exit(NULL);

}

int main()

{

int ret, cs, len;

pthread\_t tid;

fd = open(FILE, O\_RDWR);

if(fd < 0) {

printf("Counld not open file\n");

exit(1);

}

ret = pthread\_create(&tid, NULL, ping\_wdt, NULL);

if(ret) {

printf("The thread registered failed\n");

goto out;

}

while(1) {

printf("\t---Watchdog---\n");

printf("1: Setting timeout.\n");

printf("2: Getting timeout.\n");

printf("3: Getting timeleft\n");

printf("0: Exit.\n");

scanf("%d", &cs);

switch(cs) {

case 1:

printf("Enter timeout: ");

scanf("%d", &time\_wdt);

pthread\_mutex\_lock(&mutex);

timeout = time\_wdt;

ret = ioctl(fd, WDIOC\_SETTIMEOUT, &timeout);

if(ret) {

printf("Setting the timeout failed\n");

pthread\_mutex\_unlock(&mutex);

break;

}

printf("The timeout was set to %d seconds\n", timeout);

pthread\_mutex\_unlock(&mutex);

break;

case 2:

ret = ioctl(fd, WDIOC\_GETTIMEOUT, &time\_wdt);

if(ret) {

printf("Getting the timeout failed\n");

break;

}

printf("Timeout is %d seconds\n", time\_wdt);

break;

case 3:

ret = ioctl(fd, WDIOC\_GETTIMELEFT, &time\_wdt);

if(ret) {

printf("Getting the timeleft failed\n");

break;

}

printf("The timeleft is %d seconds\n", time\_wdt);

break;

case 0:

goto out;

default:A

printf("The option is invalid\n");

break;

}

}

out:

len = write(fd, "V", sizeof("V"));

if(len < 0)

goto out;

close(fd);

return ret;

}