

# CSC3150 Assignment 1

---

Author: Xue Haonan 120090453

## 1. Development Environment

### 1.1 Basic Information on Physical Machine

Hardware information:

```
CPU: Intel(R) Xeon(R) Silver 4210R CPU @ 2.40GHz  
RAM: 97585964 kB
```

Linux version:

```
$ cat /etc/os-release  
NAME="Arch Linux"  
PRETTY_NAME="Arch Linux"  
ID=arch  
BUILD_ID=rolling  
ANSI_COLOR="38;2;23;147;209"  
HOME_URL="https://archlinux.org/"  
DOCUMENTATION_URL="https://wiki.archlinux.org/"  
SUPPORT_URL="https://bbs.archlinux.org/"  
BUG_REPORT_URL="https://bugs.archlinux.org/"  
LOGO=archlinux-logo
```

Linux kernel version:

```
$ uname -r  
5.19.13-zen1-1-zen
```

### 1.2 Set up VM

Install vagrant and virtualbox:

```
sudo pacman -S vagrant virtualbox
```

Set up a directory and get into the directory:

```
mkdir ~/dev/csc3150 && cd ~/dev/csc3150
```

Set up vagrant configure:

```
vagrant init
```

Replace content in **Vagrantfile** with the following content:

```
Vagrant.configure("2") do |config|
  config.vm.box="cyzhu/csc3150"
  config.vm.network "public_network"
  config.vm.provider "virtualbox" do |vb|
    vb.cpus = 30
    vb.memory = "65536"
    vb.gui = false
  end
end
```

It should be noticed that vagrant only supports **cpus=32** as the maximum.

Boot the VM:

```
vagrant up
```

connect with VM

```
vagrant ssh
```

### 1.3 Set up development environment in VM

Update **apt** and **build-essential**:

```
sudo apt upgrade && sudo apt install build-essential
```

Install dependencies:

```
sudo apt-get install libncurses-dev gawk flex bison openssl libssl-dev
dkms libelf-dev libudev-dev libpci-dev libiberty-dev autoconf llvm dwarves
```

System version:

```
$ lsb_release -a
No LSB modules are available.
Distributor ID: Ubuntu
Description:    Ubuntu 16.04.7 LTS
Release:        16.04
Codename:       xenial
```

GCC version:

```
$ gcc -v
Using built-in specs.
COLLECT_GCC=gcc
COLLECT_LTO_WRAPPER=/usr/lib/gcc/x86_64-linux-gnu/5/lto-wrapper
Target: x86_64-linux-gnu
Configured with: ../src/configure -v --with-pkgversion='Ubuntu 5.4.0-6ubuntu1~16.04.12' --with-bugurl=file:///usr/share/doc/gcc-5/README.Bugs --enable-languages=c,ada,c++,java,go,d,fortran,objc,obj-c++ --prefix=/usr --program-suffix=-5 --enable-shared --enable-linker-build-id --libexecdir=/usr/lib --without-included-gettext --enable-threads=posix --libdir=/usr/lib --enable-nls --with-sysroot=/ --enable-clocale=gnu --enable-libstdcxx-debug --enable-libstdcxx-time=yes --with-default-libstdcxx-abi=new --enable-gnu-unique-object --disable-vtable-verify --enable-libmpx --enable-plugin --with-system-zlib --disable-browser-plugin --enable-java-awt=gtk --enable-gtk-cairo --with-java-home=/usr/lib/jvm/java-1.5.0-gcj-5-amd64/jre --enable-java-home --with-jvm-root-dir=/usr/lib/jvm/java-1.5.0-gcj-5-amd64 --with-jvm-jar-dir=/usr/lib/jvm-exports/java-1.5.0-gcj-5-amd64 --with-arch-directory=amd64 --with-ecj-jar=/usr/share/java/eclipse-ecj.jar --enable-objc-gc --enable-multiarch --disable-werror --with-arch-32=i686 --with-abi=m64 --with-multilib-list=m32,m64,mx32 --enable-multilib --with-tune=generic --enable-checking=release --build=x86_64-linux-gnu --host=x86_64-linux-gnu --target=x86_64-linux-gnu
Thread model: posix
gcc version 5.4.0 20160609 (Ubuntu 5.4.0-6ubuntu1~16.04.12)
```

Get Linux kernel archives and extract it:

```
cd /home/seed/work
wget https://cdn.kernel.org/pub/linux/kernel/v5.x/linux-5.10.147.tar.xz
sudo tar xvf *.tar.xz
```

Set up compile configure:

```
sudo su
cd /home/seed/work/work/linux-5.10.147
make mrproper
```

```
make clean
make menuconfig
```

Then save the config and exit

Compile and install:

```
make -j28
make modules_install
make install
```

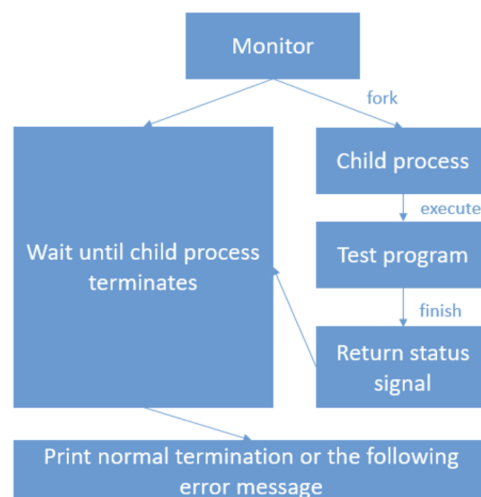
Reboot, then check kernel version:

```
$ uname -r
5.10.147
```

Done!

## 2. Task 1

The following image shows the pipeline of Task 1:



For parent process:

```
start
-> fork
-> wait until child process terminates
-> handle with signals given by child
-> terminate
```

To monitor child process, call `waitpid()` to trace the child process.

For child process:

```
start
-> fork (from parent process)
-> test program
-> return status signal
-> terminate
```

Therefore, just check `if(pid==0)` to distinguish whether the process is child process, then deal with the specific problems with `execve()`.

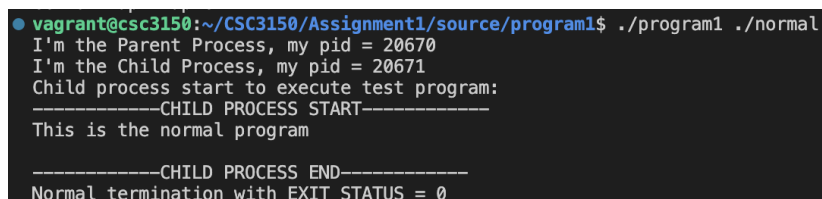
The program can handle with following signals:

- SIGABRT
- SIGALRM
- SIGBUS
- SIGCONT
- SIGCHLD
- SIGFPE
- SIGHUP
- eSIGILL
- SIGINT
- SIGKILL
- SIGPIPE
- SIGQUIT
- SIGSEGV
- SIGTSTP
- SIGTERM
- SIGTRAP

Here is the command and the sample output:

```
$ ./program1 ./normal
I'm the Parent Process, my pid = 20670
I'm the Child Process, my pid = 20671
Child process start to execute test program:
-----CHILD PROCESS START-----
This is the normal program

-----CHILD PROCESS END-----
Normal termination with EXIT STATUS = 0
```

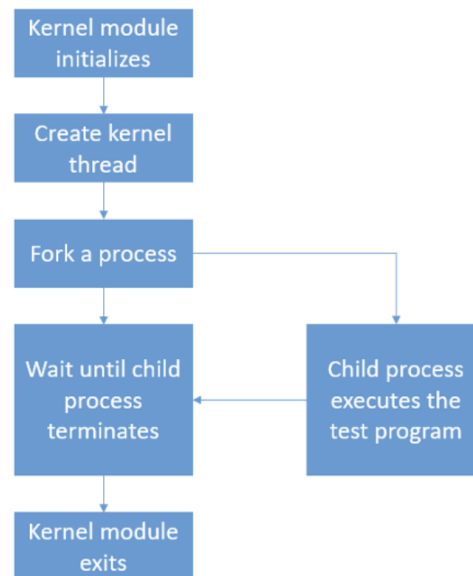


```
● vagrant@csc3150:~/CSC3150/Assignment1/source/program1$ ./program1 ./normal
I'm the Parent Process, my pid = 20670
I'm the Child Process, my pid = 20671
Child process start to execute test program:
-----CHILD PROCESS START-----
This is the normal program

-----CHILD PROCESS END-----
Normal termination with EXIT STATUS = 0
```

### 3. Task 2

The following image shows the pipeline of Task 2:



To complete the pipeline, the following functions need to be implemented:

```
int my_fork(void* argc);  
// Implement fork with kernel_clone()  
  
int my_execve(void);  
// Implement execve with do_execve()  
  
void my_wait(pid_t pid);  
// Implement wait with do_wait()
```

To call kernel function, we should apply `EXPORT_SYMBOL` to the kernel functions.

```
EXPORT_SYMBOL(function);
```

Then recompile the kernel and install

For parent process:

```
start  
-> fork with my_fork  
-> wait with my_wait until child process terminates  
-> handle with signals given by child  
-> terminate
```

For child process:

```
start
-> fork (from parent process)
-> test program with do_execve
-> return status signal
-> terminate
```

The program can handle with following signals:

- SIGABRT
- SIGALRM
- SIGBUS
- SIGCONT
- SIGCHLD
- SIGFPE
- SIGHUP
- eSIGILL
- SIGINT
- SIGKILL
- SIGPIPE
- SIGQUIT
- SIGSEGV
- SIGTSTP
- SIGTERM
- SIGTRAP

Compile the program and load it into kernel:

```
make
sudo insmod program2.ko
sudo dmesg
sudo rmmod program2.ko

# or just run
sudo bash reload.sh
```

Here is the command and the sample output:

```
[29265.460987] [program2] : Module_init Xue_Haonan 120090453
[29265.465842] [program2] : Module_init create kernel start
[29265.470719] [program2] : Kthread starts
[29265.473492] [program2] : The child process has pid = 431
[29265.473678] [program2] : Child process
[29265.479831] [program2] : The parent process has pid = 430
[29265.489168] [program2] : get signal
[29265.489169] [program2] : Child process terminated
```

```
[29265.491051] [program2] : The return signal is 126  
[29276.032379] [program2] : Module_exit
```

```
[29265.460987] [program2] : Module_init Xue_Haonan 120090453  
[29265.465842] [program2] : Module_init create kernel start  
[29265.470719] [program2] : Kthread starts  
[29265.473492] [program2] : The child process has pid = 431  
[29265.473678] [program2] : Child process  
[29265.479831] [program2] : The parent process has pid = 430  
[29265.489168] [program2] : get signal  
[29265.489169] [program2] : Child process terminated  
[29265.491051] [program2] : The return signal is 126  
[29276.032379] [program2] : Module_exit
```

## 5. What I learned?

- some Linux C APIs
- The signal and interrupt mechanism of Linux
- Getting started with kernel programming
- Several inter-process communication approaches