Report

Qingshuo Guo 121090151

October 2022

1 design of my program

1.1 task1

In user mode, using function **fork** to fork a child process to execute the test program.

Using the pid returned from **fork** to determine it is a child process or a parent process.

If it is the child process, simply use **execve(argv[1],argv,NULL)** to execute the child process.

If it is the parent process use **waitpid(pid, &status, WUNTRACED)** to wait for the child process and receive the SIGCHLD signal.

Use **WIFEXITED, WIFSIGNALED, WIFSTOPPED** to solve the SIGCHLD signal and use **WEXITSTATUS, WTERMSIG, WSTOPSIG** correspondingly to get the integer value of the signal in order to determine which kind of signal is raised.

1.2 task2

Extern function **kernel_clone, do_execve, getname_kernel, do_wait** for the kernel source code.

When program2.ko being initialized, use **kthread_create(&my_fork, NULL, "my_thread")** to create a kernel thread and run my_fork function.

In wy_fork, use the kernel_clone to invoke my_exec and get the child_pid.

I find the structure of kernel_clone_args on the linux website in order to correctly set parameter for the kernel_clone_args in my own program.

```
struct kernel_clone_args args = {
        .flags          = (clone_flags & ~CSIGNAL),
        .pidfd          = parent_tidptr,
        .child_tid      = child_tidptr,
        .parent_tid     = parent_tidptr,
        .exit_signal    = (clone_flags & CSIGNAL),
        .stack          = newsp,
        .tls            = tls_val,
};
```

Figure 1: figure - 1

Then use **task_pid_nr** to get the pid of parent process.

Then call my_wait wait until child process terminates, using the pid as parameter of my_wait.

During my_exec, use **getname_kernel, do_execve** to caught the signal.

I found the structure of wait_opts in /kernel/exec.c

```
struct wait_opts {
    enum pid_type       wo_type;
    int         wo_flags;
    struct pid      *wo_pid;

    struct siginfo __user   *wo_info;
    int __user      *wo_stat;
    struct rusage __user    *wo_rusage;

    wait_queue_t        child_wait;
    int         notask_error;
};
```

Figure 2: figure - 2

I created my own wait_opts in my program as the parameter of do_wait.

Similiar to task 1, I check those functions such as **WIFEXITED, WIFSIGNALED, WIFSTOPPED** to deal with signals in sys/wait.h and I wirte a "my" version of those fuctions.

The following steps is same as program1.

```
int my_WAIT_INT(int status)
{
    return (*(__const int *)&(status));
}
int my_WEXITSTATUS(int status)
{
    return (((status)&0xff00) >> 8);
}
int my_WTERMSIG(int status)
{
    return ((status)&0x7f);
}
int my_WSTOPSIG(int status)
{
    return my_WEXITSTATUS(status);
}
int my_WIFEXITED(int status)
{
    return (my_WTERMSIG(status) == 0);
}
int my_WIFSIGNALED(int status)
{
    return (((signed char)(((status)&0x7f) + 1) >> 1) > 0);
}
int my_WIFSTOPPED(int status)
{
    return (((status)&0xff) == 0x7f);
}
```

Figure 3: figure - 3

1.3 bonus

According to hint, we can get the statues of all process in the dictionary /proc/pid/status. We can get the pid ppid NSgtid of process and name from

this dictionary.

Be build the tree structure use linked list according to those information.

The root of the tree structure is process systemd.

For a process whose pid is equal to its NSgtid it is a process whose father is its ppid.

For a thread whose pid is not equal to its NSgtid it is a thread whose father is its NSgtid.

We can build a tree structure in this way.

Then the pstree can be printed out.

I make option -p -c -U -A PID in this task.

2 set up of environment

Follow the instruction of tutorial 1 to set up my VM box and establish ssh connnection to the server.

Download the kernel source code from tsinghua mirror to my VM box.

Tar the kernel source code under dictionary /home/seed/work.

Use $sudo apt-get install to install all the tools I need.

Copy config from /boot to /home/seed/work/KERNEL_FILE

use $make menuconfig to load config flie under current dictionary. Save the configure and exit.

From /kernel/fork.c export the symbol of kernel_clone

From /fs/exec.c export the symbol of do_execve

From /kernel/exit.c export the symbol of do_wait

From /fs/namei.c export the symbol of getname_kernel

Use $make to compile all the c program.

Use $make modules_install to install kernel_modules.

Use $make install to install kernel.

Reboot my VM box and finish.

3 screen shoot of my program output

3.1 task1

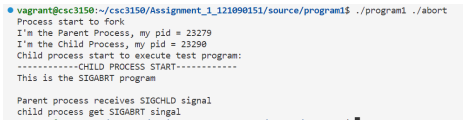

Figure 4: abort-task1

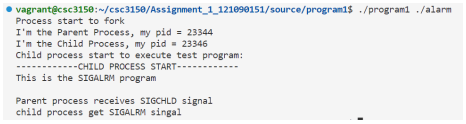

Figure 5: alarm-task1

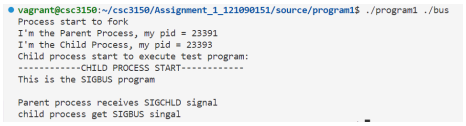

Figure 6: bus-task1

Figure 7: floating-task1

Figure 8: hangup-task1

Figure 9: illegal_instr-task1

Figure 10: interrupt-task1

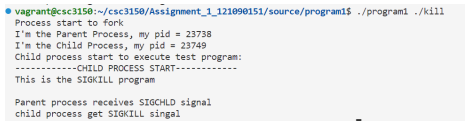

Figure 11: kill-task1

Figure 12: normal-task1

Figure 13: pipe-task1

Figure 14: quit-task1

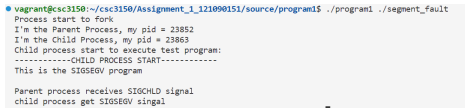

Figure 15: segment_fault-task1

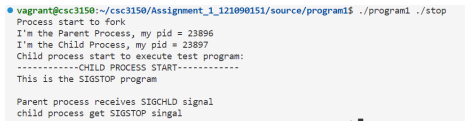

Figure 16: stop-task1

```
vagrant@csc3150:~/csc3150/Assignment_1_121090151/source/program1$ ./program1 ./terminate
Process start to fork
I'm the Parent Process, my pid = 23935
I'm the Child Process, my pid = 23937
Child process start to execute test program:
------------CHILD PROCESS START------------
This is the SIGTERM program

Parent process receives SIGCHLD signal
child process get SIGTERM singal
```

Figure 17: terminate-task1

```
vagrant@csc3150:~/csc3150/Assignment_1_121090151/source/program1$ ./program1 ./trap
Process start to fork
I'm the Child Process, my pid = 23981
Child process start to execute test program:
I'm the Parent Process, my pid = 23976
------------CHILD PROCESS START------------
This is the SIGTRAP program

Parent process receives SIGCHLD signal
child process get SIGTRAP singal
```

Figure 18: trap-task1

3.2 task2

```
[28551.132407] [program2] : Module_exit
[28561.826923] [program2] : Module_init {Guo Qingshuo} {121090151}
[28561.829391] [program2] : module_init create kthread start
[28561.831622] [program2] : module_init kthread start
[28561.833851] [program2] : The child process has pid = 29712
[28561.833852] [program2] : This is the parent process, pid = 29709
[28561.833935] [program2] : child process
[28561.933965] [program2] : get SIGABRT singal
[28561.934660] [program2] : child process terminated
[28561.935395] [program2] : The return signal is 6
```

Figure 19: abort-task2

```
[29144.532901] [program2] : Module_exit
[29152.027276] [program2] : Module_init {Guo Qingshuo} {121090151}
[29152.029838] [program2] : module_init create kthread start
[29152.031976] [program2] : module_init kthread start
[29152.034245] [program2] : The child process has pid = 32156
[29152.034245] [program2] : This is the parent process, pid = 32154
[29152.034280] [program2] : child process
[29154.041868] [program2] : get SIGALRM singal
[29154.042730] [program2] : child process terminated
[29154.043660] [program2] : The return signal is 14
```

Figure 20: alarm-task2

```
[29488.114118] [program2] : Module_exit
[29493.482761] [program2] : Module_init {Guo Qingshuo} {121090151}
[29493.485281] [program2] : module_init create kthread start
[29493.487339] [program2] : module_init kthread start
[29493.489633] [program2] : The child process has pid = 357
[29493.489634] [program2] : This is the parent process, pid = 355
[29493.489697] [program2] : child process
[29493.584855] [program2] : get SIGBUS singal
[29493.585786] [program2] : child process terminated
[29493.586784] [program2] : The return signal is 7
```

Figure 21: bus-task2

```
[29606.993364] [program2] : Module_exit
[29611.909052] [program2] : Module_init {Guo Qingshuo} {121090151}
[29611.911024] [program2] : module_init create kthread start
[29611.912638] [program2] : module_init kthread start
[29611.914495] [program2] : The child process has pid = 967
[29611.914496] [program2] : This is the parent process, pid = 964
[29611.914527] [program2] : child process
[29612.018212] [program2] : get SIGFPE singal
[29612.019298] [program2] : child process terminated
[29612.020421] [program2] : The return signal is 8
```

Figure 22: floating-task2

```
[29724.744081] [program2] : Module_exit
[29729.899917] [program2] : Module_init {Guo Qingshuo} {121090151}
[29729.901745] [program2] : module_init create kthread start
[29729.904476] [program2] : module_init kthread start
[29729.906481] [program2] : The child process has pid = 2272
[29729.906482] [program2] : This is the parent process, pid = 2270
[29729.906491] [program2] : child process
[29729.912668] [program2] : get SIGHUP singal
[29729.914490] [program2] : child process terminated
[29729.916110] [program2] : The return signal is 1
```

Figure 23: hangup-task2

```
[29834.060774] [program2] : Module_exit
[29840.498026] [program2] : Module_init {Guo Qingshuo} {121090151}
[29840.500479] [program2] : module_init create kthread start
[29840.502916] [program2] : module_init kthread start
[29840.505797] [program2] : The child process has pid = 2889
[29840.505797] [program2] : This is the parent process, pid = 2887
[29840.505813] [program2] : child process
[29840.583674] [program2] : get SIGILL singal
[29840.584626] [program2] : child process terminated
[29840.585751] [program2] : The return signal is 4
```

Figure 24: illegal_instr-task2

```
[29935.813377] [program2] : Module_exit
[29939.798668] [program2] : Module_init {Guo Qingshuo} {121090151}
[29939.801096] [program2] : module_init create kthread start
[29939.803260] [program2] : module_init kthread start
[29939.805308] [program2] : The child process has pid = 3483
[29939.805309] [program2] : This is the parent process, pid = 3480
[29939.805361] [program2] : child process
[29939.811860] [program2] : get SIGINT singal
[29939.813426] [program2] : child process terminated
[29939.815218] [program2] : The return signal is 2
```

Figure 25: interrupt-task2

```
[30116.349104] [program2] : Module_exit
[30120.944650] [program2] : Module_init {Guo Qingshuo} {121090151}
[30120.947082] [program2] : module_init create kthread start
[30120.949358] [program2] : module_init kthread start
[30120.951363] [program2] : The child process has pid = 4086
[30120.951364] [program2] : This is the parent process, pid = 4084
[30120.951419] [program2] : child process
[30120.958565] [program2] : get SIGKILL singal
[30120.960730] [program2] : child process terminated
[30120.962507] [program2] : The return signal is 9
```

Figure 26: kill-task2

```
[30288.345552] [program2] : Module_exit
[30294.261209] [program2] : Module_init {Guo Qingshuo} {121090151}
[30294.263578] [program2] : module_init create kthread start
[30294.265914] [program2] : module_init kthread start
[30294.267919] [program2] : The child process has pid = 4687
[30294.267920] [program2] : This is the parent process, pid = 4685
[30294.267980] [program2] : child process
[30294.274579] [program2] : child process exit normally
[30294.276427] [program2] : The return signal is 0
```

Figure 27: normal-task2

```
[30442.064382] [program2] : Module_exit
[30447.213716] [program2] : Module_init {Guo Qingshuo} {121090151}
[30447.216213] [program2] : module_init create kthread start
[30447.218353] [program2] : module_init kthread start
[30447.221130] [program2] : The child process has pid = 5268
[30447.221130] [program2] : This is the parent process, pid = 5266
[30447.221142] [program2] : child process
[30447.228223] [program2] : get SIGPIPE singal
[30447.229946] [program2] : child process terminated
[30447.231831] [program2] : The return signal is 13
```

Figure 28: pipe-task2

```
[30599.338658] [program2] : Module_exit
[30611.455336] [program2] : Module_init {Guo Qingshuo} {121090151}
[30611.457638] [program2] : module_init create kthread start
[30611.459741] [program2] : module_init kthread start
[30611.461941] [program2] : The child process has pid = 5935
[30611.461942] [program2] : This is the parent process, pid = 5933
[30611.461956] [program2] : child process
[30611.537359] [program2] : get SIGQUIT singal
[30611.539020] [program2] : child process terminated
[30611.540732] [program2] : The return signal is 3
```

Figure 29: quit-task2

```
30768.464865] [program2] : Module_exit
30773.596499] [program2] : Module_init {Guo Qingshuo} {121090151}
30773.598819] [program2] : module_init create kthread start
30773.601059] [program2] : module_init kthread start
30773.603036] [program2] : The child process has pid = 6526
30773.603037] [program2] : This is the parent process, pid = 6524
30773.603167] [program2] : child process
30773.703120] [program2] : get SIGSEGV singal
30773.704222] [program2] : child process terminated
30773.705287] [program2] : The return signal is 11
```

Figure 30: segment_fault-task2

```
[30926.322527] [program2] : Module_exit
[30933.916849] [program2] : Module_init {Guo Qingshuo} {121090151}
[30933.919303] [program2] : module_init create kthread start
[30933.921441] [program2] : module_init kthread start
[30933.925523] [program2] : The child process has pid = 7714
[30933.925524] [program2] : This is the parent process, pid = 7712
[30933.925702] [program2] : child process
[30933.932124] [program2] : get SIGSTOP singal
[30933.933692] [program2] : child process terminated
[30933.935674] [program2] : The return signal is 19
```

Figure 31: stop-task2

```
[31139.155224] [program2] : Module_exit
[31148.158323] [program2] : Module_init {Guo Qingshuo} {121090151}
[31148.160365] [program2] : module_init create kthread start
[31148.161936] [program2] : module_init kthread start
[31148.164343] [program2] : The child process has pid = 8919
[31148.164344] [program2] : This is the parent process, pid = 8917
[31148.164430] [program2] : child process
[31148.174547] [program2] : get SIGTERM singal
[31148.176053] [program2] : child process terminated
[31148.177825] [program2] : The return signal is 15
```

Figure 32: terminate-task2

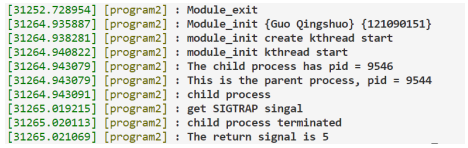

Figure 33: trap-task2

4 what I learned from this assignment

Read a lot of the underlying code related to the kernel. Understand the basic implement of the functions such as fork and wait. Get deeper perspective of the process and thread. Learned some kernel programming. Learned how to compile kernel and insert/remove my kernel