HM1 Report System Process in Linux System

Name: Tong Zhen

id: 120090694

Environment Information

Linux Distribution: `Ubuntu 20.04`
Linux Kernel Version: `5.10.146`
GCC Version: 9.4.0 (Ubuntu 9.4.0-1ubuntu1~20.04.1)

▼ Linux kernel preparation

Step 1 Install a Linux kernel 5.10 from Tsinghua Source

```
$wget https://mirror.tuna.tsinghua.edu.cn/kernel/v5.x/linux-5.10.27.tar.gz
# decompress the file
$tar -xf linux-5.10.27.tar.gz
```

Step 2 copy config file from /boot

```
$cd /boot
$cp config_file
```

Step 3 get essential programs

```
# to do menuconfig
$apt-get install libncurses-dev
$apt-get install flex
$apt-get install bison
$apt-get install libssl-dev
$apt-get install libelf-dev
$apt-get install
```

Step 4 compile

```
$make mrproper
$make clean
$make menuconfig
```

```
$make bzImage -j$(nproc)
$make modules -j$(nproc)
$make –j$(nproc)

$make modules_install
$make install
$reboot
$uname -r
```

▼ **Export Symbol**

linux-5.10.146/kernel/fork.c

```
2506: EXPORT_SYMBOL(kernel_clone);
```

linux-5.10.146/fs/exec.c

```
2013: EXPORT_SYMBOL(do_execve);
```

linux-5.10.146/fs/namei.c

```
212: EXPORT_SYMBOL(getname);
250: EXPORT_SYMBOL(getname_kernel);
```

linux-5.10.146/kernel/exit.c

```
1482: EXPORT_SYMBOL(do_wait);
```

Task 1

Fork a child process

When the program1 is running, we create a child process by calling `fork()`

```
pid_t pid = fork();
```

`pid_t` is the process descriptor, which is essentially an int：

- returns a negative number on failure
- returns two values on success: 0 and the child process ID

After we build the child process, the kernel will do the following things for us

1. Allocate new memory blocks and kernel data structures to the child process
2. Copy parts of the parent process's data structure (dataspace, stack, etc.) to the child process
3. Add the child process to the system process list
4. fork returns and then starts scheduling

Execute program

Code begins from the fork function and is shared between parent and child, as it is executed by both parent and child. The child gets a copy of the parent's data space, heap, and stack.

We identify the child process by the variable `pid == 0` Need to mention that, the child knows if its `pid` is 0 or not so that it knows if it was created successfully. We can also get the system's real pid by `getpid()`. In the child process, we can get file string data from the parent process memory and execute by calling `execute()`

Receive signal

In testing executable files compiled by c programs, different signals are returned to the parent process by calling `raise()` when meeting abnormal, and normal signal `SIGCHLD` will return by `exit(SIGCHLD)`

In the parent process, we use the `waitpid()` , because the parent process knows the pid of its child, it will assign the child status signal to `status`

```
int status;
waitpid(pid, &status, WUNTRACED);
```

We set the `WUNTRACED` because it is possible that the child will be killed/stopped.

Refered to the macros in <wait.h> we may get these signals.

```
 1) SIGHUP       2) SIGINT       3) SIGQUIT      4) SIGILL       5) SIGTRAP
 6) SIGABRT      7) SIGBUS       8) SIGFPE       9) SIGKILL     10) SIGUSR1
11) SIGSEGV     12) SIGUSR2     13) SIGPIPE     14) SIGALRM     15) SIGTERM
16) SIGSTKFLT   17) SIGCHLD     18) SIGCONT     19) SIGSTOP     20) SIGTSTP
21) SIGTTIN     22) SIGTTOU     23) SIGURG      24) SIGXCPU     25) SIGXFSZ
26) SIGVTALRM   27) SIGPROF     28) SIGWINCH    29) SIGIO       30) SIGPWR
31) SIGSYS
```

Signal print

Macros defined in <waitflags.h> can be used to analyze the status retu

If the child process is normal, we check it by `WIFEXITED()`, and it will return true.

If the child process is stopped, we check it by `WIFSTOPPED()`, and it will return true.

If the child process is failed, we check it by `WIFSIGNALED()`, and it will return true.

Finally, whatever happened, we use the `exit(0)` to end the whole process.

Program output

see Appendix 1

Task 2

Create a kernel thread to run `my_fork()`

In the `program2_inti()` function, we need first to create a kernel thread, we use the `kthread_create(threadfn, data, namefmt, arg...)` The `threadfn` is my_fork(), the data is set as NULL, and the namefmt is “my_thread”. This create is a packaging for the `kernel_clone()`.

When the new task is created, we need to do the process by wake_up_process(task)

Fork a child process in kernel mode

To fork a process using `kernel_clone()`, we first need to set the args in the `kernel_clone_args` struct.

```
struct kernel_clone_args kargs = {
      .flags = SIGCHLD,
      .exit_signal = SIGCHLD,
      .stack = &my_exec,
      .stack_size = 0,
      .parent_tid = NULL,
      .child_tid = NULL,
  };
  pid_t pid = kernel_clone(&kargs);
```

`my_exec()` is the function we want to do in the child process, and we wait the signal by using the `my_wait()` function.

```
my_wait(pid);
```

Execute the given test program

For convenience, we set the path of the test as

```
const char path[] = "/tmp/test";
```

We use the `do_execve()` function in linux kernel.

```
do_execve(struct *filename, struct *argv, struct *envp)
```

Need to mention that, if we want to get the filename sturct, `getname()` is not convenient, therefore we used the `getnamekernel()` in linux 5.10.x kernel.

Another thing worth to mention is that argv and envp are set to `NULL`.

Wait for the child process and capture signal

We build the waiting function in `void my_wait(struct task_struct *pid)` , it takes in the child process pid. Actually, what we do is just to capture the signal by creating a `wait_opts` struct and use the `do_wait(&wo)` in the linux kernel.

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

  struct waitid_info *wo_info;
  int wo_stat;
  struct rusage *wo_rusage;

  wait_queue_entry_t child_wait;
  int notask_error;
};
```

Import argument here is the status, it will be sent in the do_wait() and later be assign to a signal that can be analyzed in the `output_info()` function.

Signal analysis

In the `output_info()` function, we referred to the *<waitstatus.h> the GNU C Library*. It can do a transfermation.

```
#define __WEXITSTATUS(status) (((status)&0xff00) >> 8)

/* If WIFSIGNALED(STATUS), the terminating signal.  */
#define __WTERMSIG(status) ((status)&0x7f)

/* If WIFSTOPPED(STATUS), the signal that stopped the child.  */
#define __WSTOPSIG(status) __WEXITSTATUS(status)
```

```
/* Nonzero if STATUS indicates normal termination.  */
#define __WIFEXITED(status) (__WTERMSIG(status) == 0)

/* Nonzero if STATUS indicates termination by a signal.  */
#define __WIFSIGNALED(status) (((signed char)(((status)&0x7f) + 1) >> 1) > 0)

/* Nonzero if STATUS indicates the child is stopped.  */
#define __WIFSTOPPED(status) (((status)&0xff) == 0x7f)
```

Program output

see Appendix 2

Bonus

Data structure for a process

Like a node in a tree, every process is with a pid number its parent and children. We define the `Proc`

```
struct Proc {
    unsigned int pid;
    char comm[LN];
    unsigned char state;
    unsigned int ppid;      // the number of the parent pid
    char name[SN];          // the name of the process
    int ccnt;               // the number of the child process
    struct Proc *chrilden[];    // child process as a array
};
```

Visit process

we use the `get_proc()` function to access the /proc.

```
struct dirent {
      ino_t          d_ino;       /* inode number */
      off_t          d_off;       /* not an offset; see NOTES */
      unsigned short d_reclen;    /* length of this record */
      unsigned char  d_type;   /*type of file; not supported by all  filesystem types */

      char           d_name[256]; /* filename */
};
```

When we enter the /proc dirctory, we can view those process files

```
root@vagrant:/proc# ls
1     1175  1242  1299  1320  1489  19    2188  24    47   52   54   58   673   806   9328  9592
10    1181  1245  13    1321  1490  2     2197  2425  48   529  540  59   676   8523  9338  9603
11    1182  1250  1301  1322  1495  20    2259  25    49   53   542  596  677   8553  9483  acpi
1117  1183  1257  1303  133   15    205   2285  278   493  530  544  6    68    8554  9582  asound
1127  1186  1266  1305  1332  1514  206   23    3     496  531  547  60   683   8583  9583  buddyinfo
1128  1191  1282  1306  1389  1527  21    2315  306   50   532  548  61   69    8597  9587  bus
1136  12    1292  1309  1390  1528  2113  2316  321   51   536  552  62   766   8618  9589  cgroups
1168  1223  1294  1318  14    16    2143  2368  4     516  537  56   63   7999  9     9590  cmdline
1173  1239  1295  1319  1486  18    2144  2379  46    518  538  570  67   8     9314  9591  consoles
```

After each use of `readdir()`, it will read the next file. readdir is to read all the files in the directory in sequence, one at a time. We always want the file open with a number.

Set process information

When we get the right process file, use the `set_proc(char *pid, sturct Proc *proc)` we will set the set the path and name.

Generate tree

Iterate the `Proc` in `ProcList` . In each iteration, iterate the `ProcList` again from the first element. we asign the child to the parent if **cp->ppid == pp->pid.**

```
for (int i = 1; i < ProcN; i++) {
    struct Proc *cp = ProcList[i];

    for (int j = 0; j < ProcN; j++) {
      struct Proc *pp = ProcList[j];
      if (cp->ppid == pp->pid) {
        pp->chrilden[pp->ccnt] = cp;
        pp->ccnt++;
      }
    }
  }
```

Show tree

We use dfs recursion to show the tree sturcture.

- when the child is the first in the child list, use┬─to connect
- when the child is the middle in the child list, use ├─to connect
- when the child is the last in the child list, use" └─to connect

Output

see Appendix 3

Run instruction

Task 1

```
$make
$./program1 ./abort
$./task1.sh
```

Task 2

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

Bonus

```
$make
$./pstree
```

Appendix

▼ Appendix 1

```
Process start to fork
I'm the Parent Process, my pid = 4457
I'm the Child Process, my pid = 4458
Child process start to execute test program:
------------CHILD PROCESS START------------
This is the SIGABRT program

Parent process receives the SIGCHLD signal
CHILD PROCESS FAILED:6

Process start to fork
I'm the Parent Process, my pid = 4460
I'm the Child Process, my pid = 4461
Child process start to execute test program:
------------CHILD PROCESS START------------
This is the SIGALRM program

Parent process receives the SIGCHLD signal
CHILD PROCESS FAILED:14

Process start to fork
I'm the Parent Process, my pid = 4462
I'm the Child Process, my pid = 4463
Child process start to execute test program:
------------CHILD PROCESS START------------
This is the SIGBUS program

Parent process receives the SIGCHLD signal
CHILD PROCESS FAILED:7

Process start to fork
I'm the Parent Process, my pid = 4465
I'm the Child Process, my pid = 4466
Child process start to execute test program:
------------CHILD PROCESS START------------
This is the SIGFPE program

Parent process receives the SIGCHLD signal
CHILD PROCESS FAILED:8
```

```
Process start to fork
I'm the Parent Process, my pid = 4468
I'm the Child Process, my pid = 4469
Child process start to execute test program:
------------CHILD PROCESS START------------
This is the SIGHUP program

Parent process receives the SIGCHLD signal
CHILD PROCESS FAILED:1

Process start to fork
I'm the Parent Process, my pid = 4470
I'm the Child Process, my pid = 4471
Child process start to execute test program:
------------CHILD PROCESS START------------
This is the SIGILL program

Parent process receives the SIGCHLD signal
CHILD PROCESS FAILED:4

Process start to fork
I'm the Parent Process, my pid = 4473
I'm the Child Process, my pid = 4474
Child process start to execute test program:
------------CHILD PROCESS START------------
This is the SIGINT program

Parent process receives the SIGCHLD signal
CHILD PROCESS FAILED:2

Process start to fork
I'm the Parent Process, my pid = 4475
I'm the Child Process, my pid = 4476
Child process start to execute test program:
------------CHILD PROCESS START------------
This is the SIGKILL program

Parent process receives the SIGCHLD signal
CHILD PROCESS FAILED:9
```

```
Process start to fork
I'm the Parent Process, my pid = 4477
I'm the Child Process, my pid = 4478
Child process start to execute test program:
------------CHILD PROCESS START------------
This is the normal program

------------CHILD PROCESS END------------
Parent process receives the SIGCHLD signal
Normal termination with EXIT STATUS = 0

Process start to fork
I'm the Parent Process, my pid = 4479
I'm the Child Process, my pid = 4480
Child process start to execute test program:
------------CHILD PROCESS START------------
This is the SIGPIPE program

Parent process receives the SIGCHLD signal
CHILD PROCESS FAILED:13

Process start to fork
I'm the Parent Process, my pid = 4481
I'm the Child Process, my pid = 4482
Child process start to execute test program:
------------CHILD PROCESS START------------
This is the SIGQUIT program

Parent process receives the SIGCHLD signal
CHILD PROCESS FAILED:3

Process start to fork
I'm the Parent Process, my pid = 4484
I'm the Child Process, my pid = 4485
Child process start to execute test program:
------------CHILD PROCESS START------------
This is the SIGSEGV program

Parent process receives the SIGCHLD signal
CHILD PROCESS FAILED:11

Process start to fork
I'm the Parent Process, my pid = 4487
I'm the Child Process, my pid = 4488
Child process start to execute test program:
------------CHILD PROCESS START------------
This is the SIGSTOP program

Parent process receives the SIGCHLD signal
CHILD PROCESS STOPPED:19
```

```
Process start to fork
I'm the Parent Process, my pid = 4489
I'm the Child Process, my pid = 4490
Child process start to execute test program:
------------CHILD PROCESS START------------
This is the SIGTERM program

Parent process receives the SIGCHLD signal
CHILD PROCESS FAILED:15

Process start to fork
I'm the Parent Process, my pid = 4491
I'm the Child Process, my pid = 4492
Child process start to execute test program:
------------CHILD PROCESS START------------
This is the SIGTRAP program

Parent process receives the SIGCHLD signal
CHILD PROCESS FAILED:5
```

▼ Appendix 2

```
[ 4751.297127] [program2] : Module_init {name: TONG ZHEN} {id: 120090694}
[ 4751.297570] [program2] : module_init create kthread start
[ 4751.301342] [program2] : module_init kthread start
[ 4751.301411] [program2] : The child process has pid = 6616
[ 4751.301418] [program2] : This is the parent process, pid = 6614
[ 4751.301429] [program2] : child process
[ 4751.592172] [program2] : wo_stat = 134
[ 4751.592190] [program2] : get SIGABRT signal
[ 4751.592200] [program2] : child process is aborted.
[ 4751.592209] [program2] : the return signal is 6
[ 4844.113269] [program2] : Module_exit
[ 4844.121220] sudo (6691) used greatest stack depth: 11864 bytes left
[ 4851.767822] [program2] : Module_init {name: TONG ZHEN} {id: 120090694}
[ 4851.767871] [program2] : module_init create kthread start
[ 4851.768712] [program2] : module_init kthread start
[ 4851.768747] [program2] : The child process has pid = 7268
[ 4851.768751] [program2] : This is the parent process, pid = 7267
[ 4851.768758] [program2] : child process
[ 4853.771297] [program2] : wo_stat = 14
[ 4853.771313] [program2] : get SIGALRM signal
[ 4853.771323] [program2] : child process is exited by an alarm.
[ 4853.771331] [program2] : the return signal is 14
[ 4879.089322] [program2] : Module_exit
[ 4886.184288] [program2] : Module_init {name: TONG ZHEN} {id: 120090694}
[ 4886.184335] [program2] : module_init create kthread start
[ 4886.186658] [program2] : module_init kthread start
[ 4886.186684] [program2] : The child process has pid = 7850
[ 4886.186688] [program2] : This is the parent process, pid = 7847
[ 4886.187601] [program2] : child process
[ 4888.188094] [program2] : wo_stat = 14
[ 4888.188115] [program2] : get SIGALRM signal
[ 4888.188117] [program2] : child process is exited by an alarm.
[ 4888.188120] [program2] : the return signal is 14
[ 4919.653128] [program2] : Module_exit
[ 4926.874623] [program2] : Module_init {name: TONG ZHEN} {id: 120090694}
[ 4926.874664] [program2] : module_init create kthread start
[ 4926.875578] [program2] : module_init kthread start
[ 4926.875600] [program2] : The child process has pid = 8474
[ 4926.875621] [program2] : This is the parent process, pid = 8472
[ 4926.875625] [program2] : child process
[ 4927.609549] [program2] : wo_stat = 135
[ 4927.609560] [program2] : get SIGBUS signal
[ 4927.609565] [program2] : child process is exited by bus error.
[ 4927.609567] [program2] : the return signal is 7
[ 4941.663950] [program2] : Module_exit
```

```
[ 4941.663950] [program2] : Module_exit
[ 4947.261761] [program2] : Module_init {name: TONG ZHEN} {id: 120090694}
[ 4947.261836] [program2] : module_init create kthread start
[ 4947.263746] [program2] : module_init kthread start
[ 4947.263774] [program2] : The child process has pid = 9065
[ 4947.263777] [program2] : This is the parent process, pid = 9064
[ 4947.263783] [program2] : child process
[ 4948.173779] [program2] : wo_stat = 136
[ 4948.173785] [program2] : get SIGFPE signal
[ 4948.173787] [program2] : child process is exited by computation error.
[ 4948.173789] [program2] : the return signal is 8
[ 4965.904978] [program2] : Module_exit
[ 4970.690373] [program2] : Module_init {name: TONG ZHEN} {id: 120090694}
[ 4970.693823] [program2] : module_init create kthread start
[ 4970.693899] [program2] : module_init kthread start
[ 4970.693927] [program2] : The child process has pid = 9656
[ 4970.693929] [program2] : This is the parent process, pid = 9655
[ 4970.698807] [program2] : child process
[ 4970.702579] [program2] : wo_stat = 1
[ 4970.702587] [program2] : get SIGHUP signal
[ 4970.702590] [program2] : child process is hung up.
[ 4970.702593] [program2] : the return signal is 1
[ 5016.115639] kworker/dying (64) used greatest stack depth: 11840 bytes left
[ 5053.694683] [program2] : Module_exit
```

```
[ 5060.049881] [program2] : Module_init {name: TONG ZHEN} {id: 120090694}
[ 5060.049975] [program2] : module_init create kthread start
[ 5060.050461] [program2] : module_init kthread start
[ 5060.050508] [program2] : The child process has pid = 10248
[ 5060.050514] [program2] : This is the parent process, pid = 10247
[ 5060.050524] [program2] : child process
[ 5061.196849] [program2] : wo_stat = 132
[ 5061.196862] [program2] : get SIGILL signal
[ 5061.196867] [program2] : child process is stopped by ill-formed instruction.
[ 5061.196873] [program2] : the return signal is 4
[ 5119.645974] [program2] : Module_exit
```

```
[ 5210.292359] [program2] : Module_init {name: TONG ZHEN} {id: 120090694}
[ 5210.292473] [program2] : module_init create kthread start
[ 5210.293384] [program2] : module_init kthread start
[ 5210.293493] [program2] : The child process has pid = 11476
[ 5210.293501] [program2] : This is the parent process, pid = 11475
[ 5210.293821] [program2] : child process
[ 5210.297855] [program2] : wo_stat = 2
[ 5210.297867] [program2] : get SIGINT signal
[ 5210.297873] [program2] : child process is interupted.
[ 5210.297878] [program2] : the return signal is 2
```

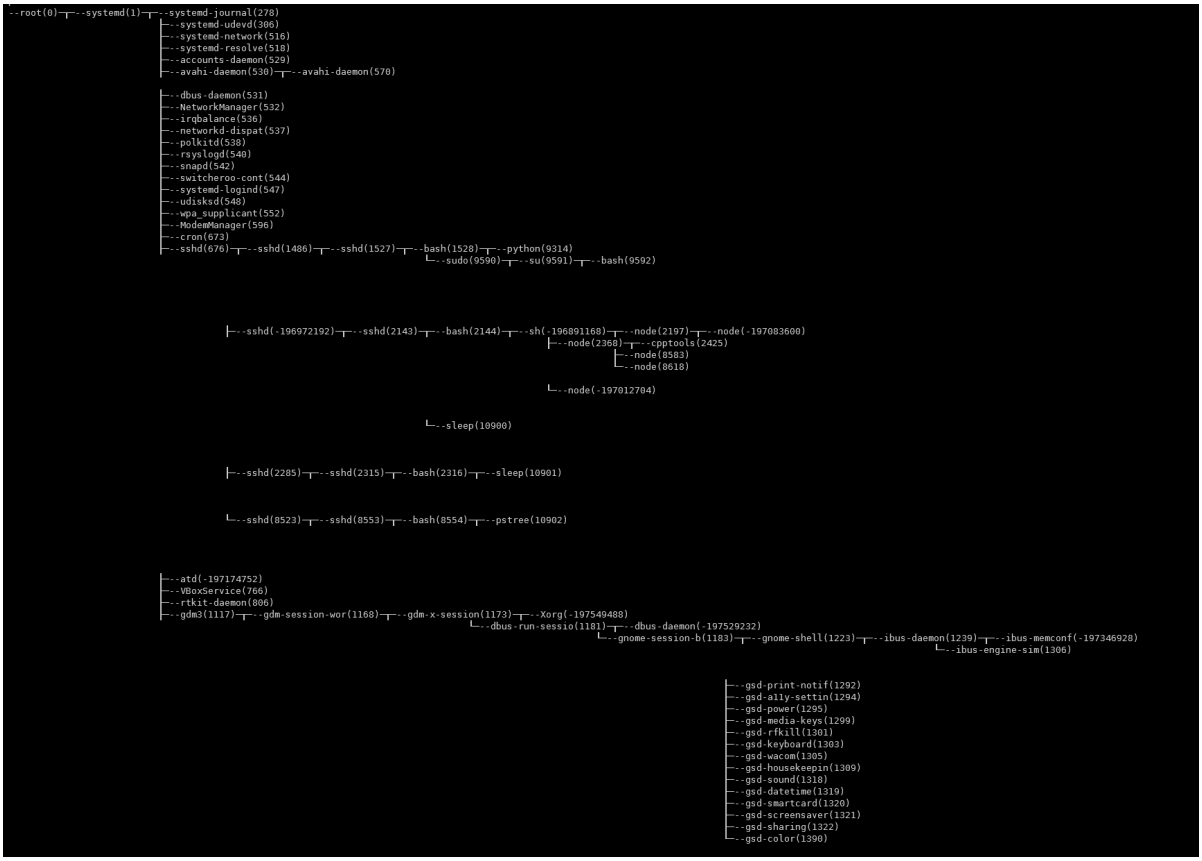
```
[ 5236.995330] [program2] : Module_init {name: TONG ZHEN} {id: 120090694}
[ 5236.995511] [program2] : module_init create kthread start
[ 5237.002913] [program2] : module_init kthread start
[ 5237.002980] [program2] : The child process has pid = 12063
[ 5237.002990] [program2] : This is the parent process, pid = 12061
[ 5237.010065] [program2] : child process
[ 5237.014693] [program2] : wo_stat = 9
[ 5237.014709] [program2] : get SIGKILL signal
[ 5237.014716] [program2] : child process is killed.
[ 5237.014722] [program2] : the return signal is 9
vagrant@vagrant:~/CSC3150/HM_1/source/program2$
```

```
[ 5237.014709] [program2] : get SIGKILL signal
[ 5237.014716] [program2] : child process is killed.
[ 5237.014722] [program2] : the return signal is 9
[ 5266.139476] [program2] : Module_exit
[ 5271.985383] [program2] : Module_init {name: TONG ZHEN} {id: 120090694}
[ 5271.985704] [program2] : module_init create kthread start
[ 5271.986292] [program2] : module_init kthread start
[ 5271.986430] [program2] : The child process has pid = 12663
[ 5271.986441] [program2] : This is the parent process, pid = 12662
[ 5271.995530] [program2] : child process
[ 5271.998897] [program2] : wo_stat = 0
[ 5271.998906] [program2] : Normal termination with EXIT STATUS = 0
[ 5301.801693] [program2] : Module_exit
[ 5308.215925] [program2] : Module_init {name: TONG ZHEN} {id: 120090694}
[ 5308.216066] [program2] : module_init create kthread start
[ 5308.216187] [program2] : module_init kthread start
[ 5308.216257] [program2] : The child process has pid = 13248
[ 5308.216266] [program2] : This is the parent process, pid = 13247
[ 5308.219046] [program2] : child process
[ 5308.221420] [program2] : wo_stat = 13
[ 5308.221428] [program2] : get SIGPIPE signal
[ 5308.221432] [program2] : child process is trying to access a broken pipe.
[ 5308.221435] [program2] : the return signal is 13
[ 5421.419548] [program2] : Module_exit
[ 5427.395215] [program2] : Module_init {name: TONG ZHEN} {id: 120090694}
[ 5427.396624] [program2] : module_init create kthread start
[ 5427.398712] [program2] : module_init kthread start
[ 5427.398741] [program2] : The child process has pid = 13884
[ 5427.398744] [program2] : This is the parent process, pid = 13881
[ 5427.405869] [program2] : child process
[ 5428.129271] [program2] : wo_stat = 131
[ 5428.129277] [program2] : get SIGQUIT signal
[ 5428.129279] [program2] : child process is quitted.
[ 5428.129282] [program2] : the return signal is 3
```

```
[ 5630.479991] [program2] : Module_init {name: TONG ZHEN} {id: 120090694}
[ 5630.480029] [program2] : module_init create kthread start
[ 5630.481056] [program2] : module_init kthread start
[ 5630.481087] [program2] : The child process has pid = 14578
[ 5630.481091] [program2] : This is the parent process, pid = 14576
[ 5630.481101] [program2] : child process
[ 5631.241720] [program2] : wo_stat = 139
[ 5631.241730] [program2] : get SIGSEGV signal
[ 5631.241734] [program2] : child process is exited by segmentation fault.
[ 5631.241737] [program2] : the return signal is 11
[ 5668.202522] [program2] : Module_exit
```

```
[ 5739.624631] [program2] : Module_init {name: TONG ZHEN} {id: 120090694}
[ 5739.624671] [program2] : module_init create kthread start
[ 5739.625709] [program2] : module_init kthread start
[ 5739.625737] [program2] : The child process has pid = 15788
[ 5739.625741] [program2] : This is the parent process, pid = 15786
[ 5739.628941] [program2] : child process
[ 5739.631058] [program2] : wo_stat = 15
[ 5739.631067] [program2] : get SIGTERM signal
[ 5739.631073] [program2] : child process is terminated.
[ 5739.631075] [program2] : the return signal is 15
[ 5748.802085] [program2] : Module_exit
[ 5753.222761] [program2] : Module_init {name: TONG ZHEN} {id: 120090694}
[ 5753.222821] [program2] : module_init create kthread start
[ 5753.225050] [program2] : module_init kthread start
[ 5753.225120] [program2] : The child process has pid = 16374
[ 5753.225125] [program2] : This is the parent process, pid = 16370
[ 5753.227746] [program2] : child process
[ 5754.051296] [program2] : wo_stat = 133
[ 5754.051306] [program2] : get SIGTRAP signal
[ 5754.051310] [program2] : child process is trapped.
[ 5754.051314] [program2] : the return signal is 5
[ 5757.743386] [program2] : Module_exit
vagrant@vagrant:~/CSC3150/HM_1/source/program2$
```

▼ Appendix 3


--root(0)─┬--systemd(1)─┬--systemd-journal(278)
├--systemd-udevd(306)
├--systemd-network(516)
├--systemd-resolve(518)
├--accounts-daemon(529)
├--avahi-daemon(530)─┬--avahi-daemon(570)
├--dbus-daemon(531)
├--NetworkManager(532)
├--irqbalance(536)
├--networkd-dispat(537)
├--polkitd(538)
├--rsyslogd(540)
├--snapd(542)
├--switcheroo-cont(544)
├--systemd-logind(547)
├--udisksd(548)
├--wpa_supplicant(552)
├--ModemManager(596)
├--cron(673)
├--sshd(676)─┬--sshd(1486)─┬--sshd(1527)─┬--bash(1528)─┬--python(9314)
└--sudo(9590)─┬--su(9591)─┬--bash(9592)
├--sshd(-196972192)─┬--sshd(2143)─┬--bash(2144)─┬--sh(-196891168)─┬--node(2197)─┬--node(-197083600)
├--node(2368)─┬--cpptools(2425)
├--node(8583)
└--node(8618)
└--node(-197012704)
└--sleep(10900)
├--sshd(2285)─┬--sshd(2315)─┬--bash(2316)─┬--sleep(10901)
└--sshd(8523)─┬--sshd(8553)─┬--bash(8554)─┬--pstree(10902)
├--atd(-197174752)
├--VBoxService(766)
├--rtkit-daemon(806)
├--gdm3(1117)─┬--gdm-session-wor(1168)─┬--gdm-x-session(1173)─┬--Xorg(-197549488)
└--dbus-run-sessio(1181)─┬--dbus-daemon(-197529232)
└--gnome-session-b(1183)─┬--gnome-shell(1223)─┬--ibus-daemon(1239)─┬--ibus-memconf(-197346928)
└--ibus-engine-sim(1306)
├--gsd-print-notif(1292)
├--gsd-a11y-settin(1294)
├--gsd-power(1295)
├--gsd-media-keys(1299)
├--gsd-rfkill(1301)
├--gsd-keyboard(1303)
├--gsd-wacom(1305)
├--gsd-housekeepin(1309)
├--gsd-sound(1318)
├--gsd-datetime(1319)
├--gsd-smartcard(1320)
├--gsd-screensaver(1321)
├--gsd-sharing(1322)
└--gsd-color(1390)


├──systemd(1127)─┬─┬──(sd-pam)(-197590000)
│ ├──dbus-daemon(1136)
│ └──pulseaudio(1266)
├──at-spi-bus-laun(-197417824)─┬─┬──dbus-daemon(1191)
├──ibus-x11(1245)
├──ibus-portal(1250)
├──at-spi2-registr(1257)
├──gjs(1282)
├──gsd-printer(1332)
├──colord(1389)
├──systemd(1489)─┬─┬──(sd-pam)(-197215264)
│ ├──pulseaudio(1495)
│ └──dbus-daemon(1514)
├──cpptools-srv(9483)
└──cpptools-srv(9735)
└──kthreadd(2)─┬─┬──rcu_gp(-198309088)
├──rcu_par_gp(4)
├──kworker/0:0H-events_highpri(6)
├──kworker/0:1H-events_highpri(8)
├──mm_percpu_wq(9)
├──ksoftirqd/0(10)
├──rcu_sched(11)
├──migration/0(12)
├──cpuhp/0(13)
├──cpuhp/1(14)
├──migration/1(15)
├──ksoftirqd/1(16)
├──kworker/1:0H-kblockd(18)
├──kdevtmpfs(19)
├──netns(20)
├──kauditd(21)
├──oom_reaper(23)
├──writeback(24)
├──kcompactd0(25)
├──kblockd(46)
├──ata_sff(47)
├──md(48)
├──rpciod(49)
├──kworker/u5:0(50)
├──xprtiod(51)
├──cfg80211(52)
├──kswapd0(53)
├──nfsiod(54)
├──acpi_thermal_pm(56)
├──scsi_eh_0(58)
├──scsi_tmf_0(59)
├──scsi_eh_1(60)
├──scsi_tmf_1(61)
├──scsi_eh_2(62)
├──scsi_tmf_2(63)
├──kworker/1:1H-events_highpri(67)
├──kworker/0:2-events(68)
├──ipv6_addrconf(69)
├──kdmflush(133)
├──jbd2/dm-0-8(205)
├──ext4-rsv-conver(206)
├──iprt-VBoxWQueue(321)
├──jbd2/sda2-8(493)
├──ext4-rsv-conver(496)
├──kworker/1:3-mm_percpu_wq(683)
├──kworker/1:1-cgroup_destroy(7999)
├──kworker/0:0-kdmflush(10057)
├──kworker/u4:1-ext4-rsv-conversion(10095)
├──kworker/u4:2-events_unbound(10147)
└──kworker/u4:0-events_unbound(10882)