
LUT
University
Operating Systems and System Programming
LAND
OF THE
CURIOUS

Experiment 3—Process state

- The process states in Linux and their mutual transitions
- Observe the process states

Experiment 3—Process state

- TASK_RUNNING (R)
  - A process that is running or waiting to run in the runnable process queue is in this state.
- TASK_INTERRUPTIBLE (S)
  - The process in the state of waiting for resources will be woken up when the waiting resources are available. It can also be awakened by other processes or kernels with signals and interrupts and enter the ready state.
- TASK_UNINTERRUPTIBLE (D)
  - The process in the state of waiting for resources will be woken up when the waiting resources are available. It cannot be woken up by other processes or kernels through signals and interrupts.

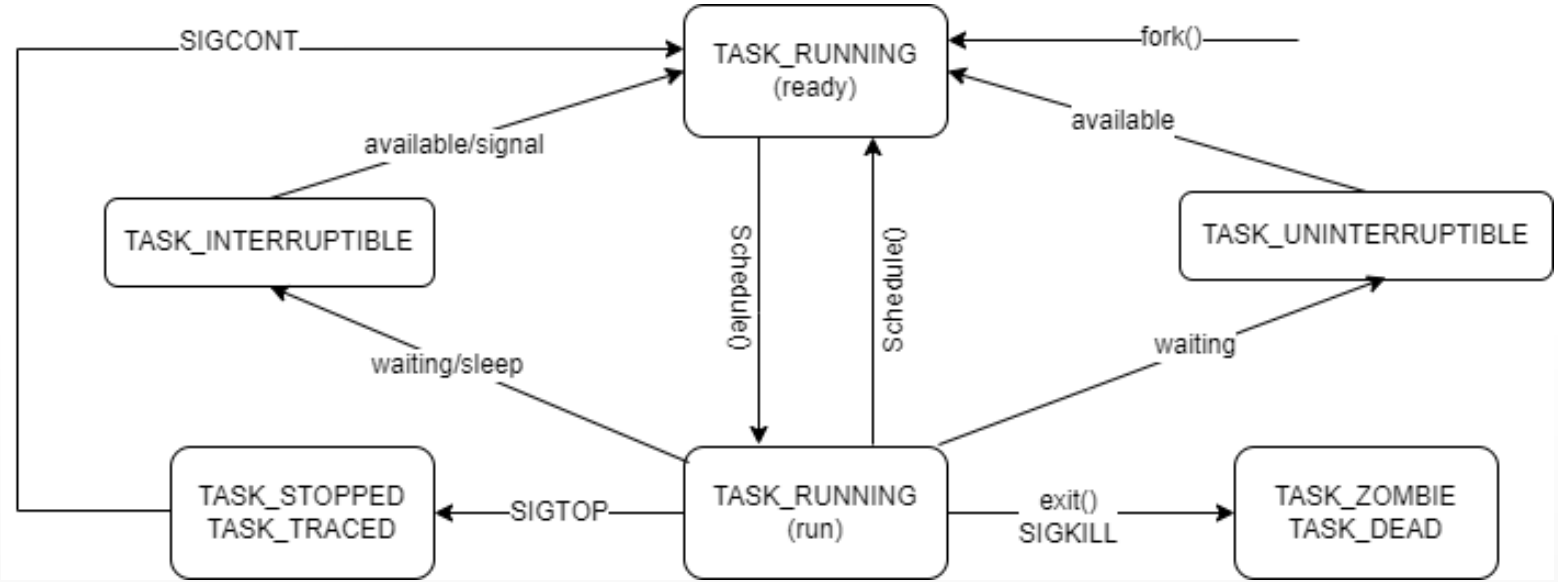
Experiment 3—Process state

- TASK_STOP/TASK_TRACED (T)
  - When the process receives the signal SIGSTOP, SIGTSTP, SIGTTIN or SIGTTOU, it will enter the suspended state. A SIGCONT signal can be sent to it to make the process transition to a runnable state.
  - When a process is being traced, it is in this special state.
- TASK_DEAD-EXIT_ZOMBIE (Z)
  - The process is stopped but not yet dead. It is a transitional state before the process finishes running. Although resources such as memory and files have been released at this time, some data structures of this process (such as task_struct) are still reserved in the kernel to wait for the parent process to recycle.

Experiment 3—Process state

- TASK_DEAD-EXIT_DEAD (Z)
  - The last state before the process dies, indicating that the parent process has obtained the accounting information of the process, and the process can be destroyed. This state is very transient and almost impossible to capture with the ps command

Experiment 3—Process state



Process state transition

Experiment 3—run_status.c

```
#include <stdio.h>
#include <unistd.h>
int main()
{
    int i=0,j=0,k=0;
    for(i=0;i<1000000;i++)
    {
        for(j=0;j<10000000;j++)
        {
            k++;
            k--;
        }
    }
}
```

Experiment 3—Process state

- Observe the running state
  - gcc –o run_status run_status.c
  - ./run_status &
  - ps -ax | grep run_status
  - ps -ax | grep run_status |grep -v grep

Experiment 3—Process state

```
[root@localhost example]# gcc -o run_status run_status.c
[root@localhost example]# ./run_status &
[1] 4962
[root@localhost example]# ps ax ¦ grep run_status
 4964 tty1     R+     0:00 grep run_status
[1]+  Exit 196                ./run_status
[root@localhost example]# _
```

Experiment 3—Process state

- Observe the “T” state
  - gcc –o run_status run_status.c
  - ./run_status &
  - kill –SIGSTOP pid
  - kill –SIGCONT pid

Experiment 3—Process state

```
[root@localhost code]# kill -SIGSTOP 6350

[1]+  Stopped                 ./run_status
[root@localhost code]# ps ax | grep run_status
 6350 tty1     T      0:24 ./run_status
 6356 tty1     S+     0:00 grep run_status
```

```
[root@localhost code]# kill -SIGCONT 6350
[root@localhost code]# ps ax | grep run_status
 6350 tty1     R      0:26 ./run_status
 6358 tty1     S+     0:00 grep run_status
```

Experiment 3—interruptible_status.c

```
#include <stdio.h>
#include <unistd.h>
int main()
{
        sleep(60);
        return 0;

}
```

Experiment 3—Process state

- Observe the “S” state

```
[root@localhost example]# gcc -o interruptible_status interruptible_status.c
[root@localhost example]# ./interruptible_status & ps ax | grep interruptible_st
atus
[1] 4351
 4351 tty1     S      0:00 ./interruptible_status
 4353 tty1     S+     0:00 grep interruptible_status
[root@localhost example]# kill -9 4351
[root@localhost example]# ps ax | grep interruptible_status
 4357 tty1     S+     0:00 grep interruptible_status
[1]+  Killed                  ./interruptible_status
[root@localhost example]# ps ax | grep interruptible_status
 4359 tty1     S+     0:00 grep interruptible_status
[root@localhost example]# _
```

Experiment 3—uninter_status.c

```
#include <stdio.h>
#include <unistd.h>
int main()
{
        if(vfork()==0){
            sleep(60);
            return 0;
      }

}
```

Experiment 3—Process state

- Observe the “D” state
  - old kernel

```
[root@localhost example]# gcc -o uninter_status uninter_status.c
[root@localhost example]# ./uninter_status & ps ax | grep uninter_status
[1] 4619
 4619 tty1     D      0:00 ./uninter_status
 4621 tty1     S+     0:00 grep uninter_status
 4622 tty1     S      0:00 ./uninter_status
[root@localhost example]# kill -9 4619
[root@localhost example]# ps ax | grep uninter_status
 4619 tty1     D      0:00 ./uninter_status
 4622 tty1     S      0:00 ./uninter_status
 4624 tty1     S+     0:00 grep uninter_status
[root@localhost example]# _
```

Experiment 3—Process state

- Observe the “D” state
  - newer kernel

```
zsh@DEEP-2022JVUNCQ:~/os$ ./uninter &
[1] 377
zsh@DEEP-2022JVUNCQ:~/os$ ps -ax | grep uninter
  377 pts/1    D      0:00 ./uninter
  378 pts/1    S      0:00 ./uninter
  380 pts/1    S+     0:00 grep --color=auto uninter
zsh@DEEP-2022JVUNCQ:~/os$ kill -9 377
[1]+  Killed                  ./uninter
zsh@DEEP-2022JVUNCQ:~/os$ ps -ax | grep uninter
  378 pts/1    S      0:00 ./uninter
  382 pts/1    S+     0:00 grep --color=auto uninter
zsh@DEEP-2022JVUNCQ:~/os$
```

Experiment 3—zombie_status.c

```
#include <stdio.h>
#include <unistd.h>
int main()
{
        if(fork()){
         sleep(60);
    }
}
```

Experiment 3—Process state

- Observe the “Z” state
- Run the ps command within 60s to check the running status.

```
[root@localhost example]# gcc -o zombie_status zombie_status.c
[root@localhost example]# ./zombie_status & ps -ax | grep zombie_status
[1] 5039
Warning: bad syntax, perhaps a bogus '-'? See /usr/share/doc/procps-3.2.7/FAQ
 5039 tty1     S      0:00 ./zombie_status
 5041 tty1     R+     0:00 grep zombie_status
 5042 tty1     Z      0:00 [zombie_status] <defunct>
[root@localhost example]# _
```