



BAHIR DAR UNIVERSITY

BIT

FACULTY OF COMPUTING

DEPARTMENT OF SOFTWARE ENGINEERING (SED)

COURSE: Operating System and System Programming

NAME

SHALOM WUBU

NUMBER ID

1308438

SUBMISSION DATE: 26/08/2014 E.C

Instructors Name: Wendimu Baye

INTRODUCTION

What is System call?

System call is like a bridge or interface that connects the process and an operating system. They are mostly available as assembly language instructions. System calls are usually made when a process in user mode requires access to a resource. Then it requests the kernel to provide the resource via a system call.

In general, system calls are required in the following situations.

- If a file system requires the creation or deletion of files. Reading and writing from files also require a system call. (File management)
- Creation and management of new processes. (Process control)
- Network connections also require system calls. This includes sending and receiving packets.
- Access to a Hardware Devices such as a printer, scanner etc. requires a system call. (Device management).

Get process resource limits

Name

getrlimit – get resource limits.

getrlimit() functions

getrlimit() is type of system call that returns the calling process's resource limits. A resource limit comprises two values: one that specifies the current (soft) limit and the other that specifies the maximum (hard) limit.

Why getrlimit functions?

As system resources are limited, each process is constrained by limitations imposed by the operating system. The limit function commands (a built-in commands included in the Bourne shell on the command line allows the user to view and adjust current system restrictions available to the shell and the processes it starts.

Getrlimit function is very useful to return the resource limits as the system resources are limited. Each resource has a soft and hard limit associated with it.

How do they work?

Each call to either getrlimit() identifies a specific resource to be operated upon as well as a resource limit. A resource limit is represented by an rlimit structure The rlim_cur member specifies the current or soft limit and the rlim_max member specifies the maximum or hard

limit. Soft limits may be changed by a process to any value that is less than or equal to the hard limit. A process may (irreversibly) lower its hard limit to any value that is greater than or equal to the soft limit. Only a process with appropriate privileges can raise a hard limit. Both hard and soft limits can be changed in a single call to `setrlimit()` subject to the constraints described above. The value `RLIM_INFINITY`, defined in `<sys/resource.h>`, shall be considered to be larger than any other limit value. If a call to `getrlimit()` returns `RLIM_INFINITY` for a resource, it means the implementation shall not enforce limits on that resource.

The resources that are limited are:

RLIMIT_CORE

The maximum size of a dump of memory (in bytes) allowed for the process. A value of 0 (zero) prevents file creation. Dump file creation will stop at this limit.

RLIMIT_CPU

The maximum amount of CPU time (in seconds) allowed for the process. If the limit is exceeded, a `SIGXCPU` signal is sent to the process and the process is granted a small CPU time extension to allow for signal generation and delivery. If the extension is used up, the process is terminated with a `SIGKILL` signal.

RLIMIT_DATA

The maximum size of the break value for the process, in bytes. In this implementation, this resource always has a hard and soft limit value of `RLIM_INFINITY`.

RLIMIT_FSIZE

The maximum file size (in bytes) allowed for the process. A value of 0 (zero) prevents file creation. If the size is exceeded, a `SIGXFSZ` signal is sent to the process. If the process is blocking, catching, or ignoring `SIGXFSZ`, continued attempts to increase the size of a file beyond the limit will fail with an `errno` of `EFBIG`.

RLIMIT_MEMLIMIT

The maximum amount of usable storage above the 2gigabyte bar (in 1megabyte segments) that can be allocated.

RLIMIT_NOFILE

The maximum number of open file descriptors allowed for the process. This number is one greater than the maximum value that may be assigned to a newly created descriptor. (That is, it is one-based.) Any function that attempts to create a new file descriptor beyond the limit will fail with an `EMFILE` `errno`.

RLIMIT_STACK

The maximum size of the stack for a process, in bytes. Note that in z/OS® UNIX services, the stack is a per-thread resource. In this implementation, this resource always has a hard and soft limit value of RLIM_INFINITY. A call to setrlimit() to set this resource to any value other than RLIM_INFINITY will fail with an errno of EINVAL

RLIMIT_AS

The maximum address space size for the process, in bytes.

Synopsis

```
int getrlimit(int resource, struct rlimit *rlim);
```

Each resource has an associated soft and hard limit.

- **soft limit:** The soft limit is the actual limit enforced by the kernel for the corresponding resource.
- **hard limit:** The hard limit acts as a ceiling for the soft limit.

The two limits are defined by the following structure

```
struct rlimit {
    rlim_t rlim_cur; /* Soft limit */
    rlim_t rlim_max; /* Hard limit (ceiling for rlim_cur) */
};
```

```
int getrlimit(int resource, struct rlimit *rlim);
```

the parameters of getrlimits are int resource and struct rlimit*rlim.

```
#include <stdio.h>
#include <sys/resource.h>
#include <string.h>
#include <errno.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
int main() {
    struct rlimit old_lim, lim, new_lim;
```

```
    if( getrlimit(RLIMIT_NOFILE, &old_lim) == 0)
```

```

printf("Old limits -> soft limit= %ld \t"
      " hard limit= %ld \n", old_lim.rlim_cur,
      old_lim.rlim_max);

else
printf(stderr, "%s\n", strerror(errno));
lim.rlim_cur = 3;
lim.rlim_max = 1024;
if(setrlimit(RLIMIT_NOFILE, &lim) == -1)
fprintf(stderr, "%s\n", strerror(errno));
if( getrlimit(RLIMIT_NOFILE, &new_lim) == 0)
printf("New limits -> soft limit= %ld "
      "\t hard limit= %ld \n", new_lim.rlim_cur,
      new_lim.rlim_max);

else
fprintf(stderr, "%s\n", strerror(errno));
return 0;

```

The flag RLIMIT_NOFILE indicates that we want to restrict the number of files that that will be created. Thus the created number of files can't more than the specified value.

output

```

shalom@shalom-VirtualBox:~$ gcc hello.c -o test
shalom@shalom-VirtualBox:~$ ./test
Old limits -> soft limit= 1024    hard limit= 1048576
New limits -> soft limit= 3      hard limit= 1024
shalom@shalom-VirtualBox:~$

```

```

#include <sys/resource.h>
#include <sys/time.h>
#include <unistd.h>
#include <stdio.h>

int main ()
{
    struct rlimit rl;

    // First get the time limit on CPU
    getrlimit (RLIMIT_CPU, &rl);

    printf("\n Default value is : %lld\n", (long long int)rl.rlim_cur);

    // Change the time limit
    rl.rlim_cur = 1;

    // Now call setrlimit() to set the

```

```
// changed value.
setrlimit (RLIMIT_CPU, &rl);

// Again get the limit and check
getrlimit (RLIMIT_CPU, &rl);

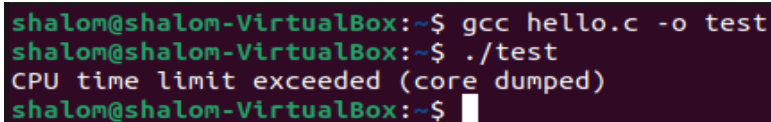
printf("\n Default value now is : %lld\n", (long long int)rl.rlim_cur);

// Simulate a long time consuming work
while (1);

return 0;
}
```

The RLIMIT_CPU flag gives the maximum number of processes to be processed.

output



```
shalom@shalom-VirtualBox:~$ gcc hello.c -o test
shalom@shalom-VirtualBox:~$ ./test
CPU time limit exceeded (core dumped)
shalom@shalom-VirtualBox:~$
```

```
#include <sys/resource.h>
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>

int main (int argc, char *argv[])
{
    struct rlimit limit;

    /* Set the file size resource limit. */
    limit.rlim_cur = 65535;
    limit.rlim_max = 65535;
    if (setrlimit(RLIMIT_FSIZE, &limit) != 0) {
        printf("setrlimit() failed with errno=%d\n", errno);
        exit(1);
    }

    /* Get the file size resource limit. */
    if (getrlimit(RLIMIT_FSIZE, &limit) != 0) {
        printf("getrlimit() failed with errno=%d\n", errno);
        exit(1);
    }
}
```

```
printf("The soft limit is %llu\n", limit.rlim_cur);  
printf("The hard limit is %llu\n", limit.rlim_max);  
exit(0);}
```

Output

```
shalom@shalom-VirtualBox:~$ gcc hello.c -o test  
shalom@shalom-VirtualBox:~$ ./test  
The soft limit is 65535  
The hard limit is 65535  
shalom@shalom-VirtualBox:~$
```

The flag `RLIMIT_FSIZE` indicates the maximum number of file descriptors that can be opened by a process

```
#include <sys/resource.h>  
int main (void)  
{  
    struct rlimit limit;  
  
    getrlimit (RLIMIT_STACK, &limit);  
    printf ("\nStack Limit = %ld and %ld max\n", limit.rlim_cur, limit.rlim_max);  
}
```

Output

```
shalom@shalom-VirtualBox:~$ gcc hello.c -o test  
shalom@shalom-VirtualBox:~$ ./test  
  
Stack Limit = 8388608 and -1 max  
shalom@shalom-VirtualBox:~$
```

`RLIMIT_STACK` is used to express the maximum size of the initial thread's size

```

#include
<stdio.h>

#include <stdint.h>
#include <sys/resource.h>

void print_rlimit(struct rlimit *r, const char *name) {
    int64_t cur;          /* Soft limit */
    int64_t max;          /* Hard limit */
    cur = r->rlim_cur;
    max = r->rlim_max;
    printf("RLIMIT_%s :rlim_cur => %#llx, :rlim_max => %#llx\n",
           name, cur, max);
}

int main(int argc, char *argv[]) {
    struct rlimit rlim;
    int resources[] = {RLIMIT_CORE, RLIMIT_CPU, RLIMIT_DATA, RLIMIT_FSIZE,
                       RLIMIT_MEMLOCK, RLIMIT_NOFILE, RLIMIT_NPROC, RLIMIT_RSS,
                       RLIMIT_STACK};
    const char *names[] = {"CORE", "CPU", "DATA", "FSIZE",
                           "MEMLOCK", "NOFILE", "NPROC", "RSS",
                           "STACK"};

    int n = sizeof(resources)/sizeof(resources[0]);
    int i;
    for (i = 0; i < n; i++) {
        getrlimit(resources[i], &rlim);
        print_rlimit(&rlim, names[i]);
    }
    return 0;
}

```

output

```

shalom@shalom-VirtualBox:~$ gcc hello.c -o test
shalom@shalom-VirtualBox:~$ ./test
RLIMIT_CORE :rlim_cur => 0, :rlim_max => 0xffffffffffffffff¥nRLIMIT_CPU :rlim_c
ur => 0xffffffffffffffff, :rlim_max => 0xffffffffffffffff¥nRLIMIT_DATA :rlim_cu
r => 0xffffffffffffffff, :rlim_max => 0xffffffffffffffff¥nRLIMIT_FSIZE :rlim_cu
r => 0xffffffffffffffff, :rlim_max => 0xffffffffffffffff¥nRLIMIT_MEMLOCK :rlim_
cur => 0x1d69f000, :rlim_max => 0x1d69f000¥nRLIMIT_NOFILE :rlim_cur => 0x400, :
rlim_max => 0x100000¥nRLIMIT_NPROC :rlim_cur => 0x39c6, :rlim_max => 0x39c6¥nRL
IMIT_RSS :rlim_cur => 0xffffffffffffffff, :rlim_max => 0xffffffffffffffff¥nRLIM
IT_STACK :rlim_cur => 0x800000, :rlim_max => 0xffffffffffffffff¥nshalom@shalom-
VirtualBox:~$

```

These are the default values because we didn't set any values.

