Chapter 1: Linux System Architecture

User Mode / Userland:
Apps (processes)

Kernel Mode / kernel-space:
OS kernel, device drivers, etc
Chapter 2: Virtual Memory
Process VAS (Virtual Address Space)

High address

Low address: 0x0
Stack Segment

High address

SP: Lowest stack address

Direction of stack growth

Stack frame of __libc_start_main

Stack frame of main

Stack frame of foo

Stack frame of bar

Stack frame of jail

Direction of stack growth
Chapter 3: Resource Limits

```
$ ulimit -f
unlimited
$ ulimit -f 2
$ ulimit -f 2
$ dd if=/dev/urandom of=tst count=2048 bs=1
2048+0 records in
2048+0 records out
2048 bytes (2.0 kB, 2.0 KiB) copied, 0.00688134 s, 298 kB/s
$ dd if=/dev/urandom of=tst count=2049 bs=1
File size limit exceeded (core dumped)
```

```
$ ulimit -a
  core file size          (blocks)  unlimited
  data seg size           (kbytes)  unlimited
  file size               (kbytes)  65536
  max locked memory       (kbytes)  65536
  max memory size         (kbytes)  unlimited
  open files              (files)   4096
  pipe size               (512 bytes) 8
  POSIX message queues    (bytes)   81920
  real-time priority      (prio)   0
  stack size              (kbytes)  8192
  cpu time                (seconds)  unlimited
  max user processes      (tasks)   unlimited
  virtual memory          (kbytes)  unlimited
  file locks              (files)   unlimited
```
<table>
<thead>
<tr>
<th>RESOURCE</th>
<th>DESCRIPTION</th>
<th>SOFT</th>
<th>HARD UNITS</th>
</tr>
</thead>
<tbody>
<tr>
<td>AS</td>
<td>address space limit</td>
<td>unlimited</td>
<td>unlimited bytes</td>
</tr>
<tr>
<td>CORE</td>
<td>max core file size</td>
<td>0</td>
<td>unlimited blocks</td>
</tr>
<tr>
<td>CPU</td>
<td>CPU time</td>
<td>unlimited</td>
<td>unlimited seconds</td>
</tr>
<tr>
<td>DATA</td>
<td>max data size</td>
<td>unlimited</td>
<td>unlimited bytes</td>
</tr>
<tr>
<td>FSIZE</td>
<td>max file size</td>
<td>2048</td>
<td>2048 blocks</td>
</tr>
<tr>
<td>LOCKS</td>
<td>max number of file locks held</td>
<td>unlimited</td>
<td>unlimited</td>
</tr>
<tr>
<td>MEMLOCK</td>
<td>max locked-in-memory address space</td>
<td>65536</td>
<td>65536 bytes</td>
</tr>
<tr>
<td>MSGQUEUE</td>
<td>max bytes in POSIX mqueues</td>
<td>819200</td>
<td>819200 bytes</td>
</tr>
<tr>
<td>NICE</td>
<td>max nice prio allowed to raise</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>NOFILE</td>
<td>max number of open files</td>
<td>2000</td>
<td>2000</td>
</tr>
<tr>
<td>NPROC</td>
<td>max number of processes</td>
<td>7741</td>
<td>7741</td>
</tr>
<tr>
<td>RSS</td>
<td>max resident set size</td>
<td>unlimited</td>
<td>unlimited pages</td>
</tr>
<tr>
<td>RTPRIO</td>
<td>max real-time priority</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>RTTIME</td>
<td>timeout for real-time tasks</td>
<td>unlimited</td>
<td>unlimited microsecs</td>
</tr>
<tr>
<td>SIGPENDING</td>
<td>max number of pending signals</td>
<td>7741</td>
<td>7741</td>
</tr>
<tr>
<td>STACK</td>
<td>max stack size</td>
<td>8388608</td>
<td>unlimited bytes</td>
</tr>
</tbody>
</table>
Chapter 4: Dynamic Memory Allocation
### Chapter 5: Linux Memory Issues

There are 91 matching records.
Displaying matches 1 through 20.

<table>
<thead>
<tr>
<th>Vuln ID</th>
<th>Summary</th>
<th>CVSS Severity</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2017-18120</td>
<td>A double-free bug in the read_gif function in gifread.c in gifsicle 1.90 allows a remote attacker to cause a denial-of-service attack or unspecified other impact via a maliciously crafted file, because last_name is mishandled, a different vulnerability than CVE-2017-1000421.</td>
<td>(not available)</td>
</tr>
<tr>
<td>CVE-2018-0101</td>
<td>A vulnerability in the Secure Sockets Layer (SSL) VPN functionality of the Cisco Adaptive Security Appliance (ASA) Software could allow an unauthenticated, remote attacker to cause a reload of the affected system or to remotely execute code. The vulnerability is due to an attempt to double free a region of memory when the webvpn feature is enabled on the Cisco ASA device. An attacker could exploit this vulnerability by sending...</td>
<td>(not available)</td>
</tr>
</tbody>
</table>
Chapter 6: Debugging Tools for Memory Issues

$ valgrind ./membugs_dbg 8

==24337== Memcheck, a memory error detector
==24337== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==24337== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==24337== Command: ./membugs_dbg 8
==24337==  uaf():165: arr = 0x521f040:aaaaaaaaaaaaaaaaaaaaaaaa
==24337==  Invalid write of size 1
==24337==   at 0x4C32E7F: strncpy (vg_replace_strmem.c:549)
==24337==     by 0x400EB5: uaf (membugs.c:173)
==24337==   at 0x4014A6: process_args (membugs.c:348)
==24337==   by 0x401574: main (membugs.c:379)
==24337==   Address 0x521f040 is 0 bytes inside a block of size 512 free'd
==24337==   at 0x4C30D1B: free (vg_replace_malloc.c:530)
==24337==     by 0x400F93: uaf (membugs.c:171)
==24337==   by 0x4014A6: process_args (membugs.c:348)
==24337==   by 0x401574: main (membugs.c:379)
==24337==  Block was alloc'd at
==24337==   at 0x4C2FB6B: malloc (vg_replace_malloc.c:299)
==24337==     by 0x400EC5: uaf (membugs.c:159)
==24337==   at 0x4014A6: process_args (membugs.c:348)
==24337==   by 0x401574: main (membugs.c:379)
==24337==
==24337==  Invalid read of size 1
==24337==   at 0x4C3202B: strlen (vg_replace_strmem.c:425)
==24337==     by 0x4E8FB2: vfprintf (in /usr/lib64/libc-2.26.so)
==24337==     by 0x4E9255: printf (in /usr/lib64/libc-2.26.so)
==24337==     by 0x400FE2: uaf (membugs.c:174)
==24337==     by 0x4014A6: process_args (membugs.c:348)
==24337==     by 0x401574: main (membugs.c:379)
### LeaksSanitizer: detected memory leaks

Direct leak of 4096 byte(s) in 1 object(s) allocated from:

```plaintext
#0 0x4d9d60 in malloc (/home/seawolf/otmp/membugs_dbg_asan+0x4d9d60)
#1 0x513ec in slyl_getpath /home/seawolf/kaiwanTECH/book_src/ch5/membugs.c:37:9
#2 0x51399 in leakage_case3 /home/seawolf/kaiwanTECH/book_src/ch5/membugs.c:55:2
#3 0x512522 in process_args /home/seawolf/kaiwanTECH/book_src/ch5/membugs.c:363:4
#4 0x512291 in main /home/seawolf/kaiwanTECH/book_src/ch5/membugs.c:375:2
#5 0x7f5f23247b96 in __libc_start_main /build/glibc-0TsEL5/glibc-2.27/csu/../csu/libc-start.c:310
```

Summary: AddressSanitizer: 4096 byte(s) leaked in 1 allocation(s).
Chapter 7: Process Credentials

<table>
<thead>
<tr>
<th>Access Category</th>
<th>Owner (U)</th>
<th>Group (G)</th>
<th>Others (O)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Permission Bits</td>
<td>r w x</td>
<td>r w x</td>
<td>r w x</td>
</tr>
<tr>
<td>Example (myfile)</td>
<td>1 1 0</td>
<td>1 1 0</td>
<td>1 0 0</td>
</tr>
</tbody>
</table>

```bash
$ ls -l $(which passwd)
-rwsr-xr-x 1 root root 54224 Aug 21 05:26 /usr/bin/passwd
```

**Caveats**

Do not use `system()` from a privileged program (a set-user-ID or set-group-ID program, or a program with capabilities) because strange values for some environment variables might be used to subvert system integrity. For example, `PATH` could be manipulated so that an arbitrary program is executed with privilege. Use the `exec(3)` family of functions instead, but not `execvp(3)` or `execvp(3)` (which also use the `PATH` environment variable to search for an executable).

`system()` will not, in fact, work properly from programs with set-user-ID or set-group-ID privileges on systems on which `/bin/sh` is bash version 2: as a security measure, bash 2 drops privileges on startup. (Debian uses a different shell, `dash(1)`, which does not do this when invoked as `sh`.)
Welcome to Seawolf-Minidv

*** IMPORTANT NOTE ***

Please read this file to gain maximum mileage.

#SEAWOLF_MINIDV_USEFULNOTES.txt

---------- To be taken with a pinch of salt. :-) ----------

Your reasoning powers are good, and you are a fairly good planner.

* Documentation: https://help.ubuntu.com
  * Management: https://landscapescanonical.com
  * Support: https://ubuntu.com/adventure

Running: /#setup bash
IP addr: 192.168.3.10

$ ps
  PID  TTY         TIME CMD
  6012 pts/0    00:00:00 bash
  6724 pts/0    00:00:00 rootsh_hack2
  6725 pts/0    00:00:00 sh
  6720 pts/0    00:00:00 bash
  6750 pts/0    00:00:00 ps
$ id -u
$ 1000
$ /rootsh_hack2
  WARNING! rootsh_hack2.c:main:36: setuid() failed!
  perror says: Operation not permitted
$ id -u
$ 1000
$ exit
$
Broadcast message from seawolf@seawolf-mindev (pts/1) (Tue Feb 20 17:12:00 2018)

Hey tty's, wassup!

Broadcast message from seawolf@seawolf-mindev (pts/1) (Tue Feb 20 17:12:00 2018)

Hey tty's, wassup!
$ sudo chown root savedset_demo
[sudo] password for seawolf:
$ sudo chmod u+s savedset_demo
$ ls -l savedset_demo
-rwsrwxr-x 1 root seawolf 13144 Feb 20 09:22 savedset_demo*
$ ./savedset_demo
t0: Init:
  RUID=1000  EUID=0
  RGID=1000  EGID=1000
  Ok, we're effectively running as root! (EUID==0)
t1: Becoming my original self!
  RUID=1000  EUID=1000
  RGID=1000  EGID=1000
t2: Switching to privileged state now...
  RUID=1000  EUID=0
  RGID=1000  EGID=1000
  Yup, we're root again!
t3: Switching back to unprivileged state now ...
  RUID=1000  EUID=1000
  RGID=1000  EGID=1000
$
Chapter 8: Process Capabilities

Transformation of capabilities during execve()

During an execve(2), the kernel calculates the new capabilities of the process using the following algorithm:

\[
\begin{align*}
P'(\text{ambient}) &= (\text{file is privileged}) \ ? \ 0 : P(\text{ambient}) \\
P'(\text{permitted}) &= (P(\text{inheritable}) \& F(\text{inheritable})) \mid \\
& \quad (F(\text{permitted}) \& \text{cap_bset}) \mid P'(\text{ambient}) \\
P'(\text{effective}) &= F(\text{effective}) \ ? \ P'(\text{permitted}) : P'(\text{ambient}) \\
P'(\text{inheritable}) &= P(\text{inheritable}) \quad \text{[i.e., unchanged]}
\end{align*}
\]

where:
- \( P \) denotes the value of a thread capability set before the execve(2)
- \( P' \) denotes the value of a thread capability set after the execve(2)
- \( F \) denotes a file capability set
- \( \text{cap_bset} \) is the value of the capability bounding set (described below).

```
$ ls -l /usr/bin/passwd /usr/bin/write /usr/bin/ping /bin/dumpcap
-rw-r-x----. 1 root wireshark 109344 Jan 19 19:45 /bin/dumpcap
-rwsr-xr-x. 1 root root 27880 Aug 4 2017 /usr/bin/passwd
-rwrxr-xr-x. 1 root root 62080 Aug 3 2017 /usr/bin/ping
-rwxr-sr-x. 1 root tty 19584 Sep 22 14:07 /usr/bin/write
$ 
```
Chapter 9: Process Execution
$ ps
  PID TTY       TIME CMD
  3396 pts/3  00:00:00 bash
  13030 pts/3  00:00:00 ps
$ bash
$ ps
  PID TTY       TIME CMD
  3396 pts/3  00:00:00 bash
  13040 pts/3  00:00:00 bash
  13087 pts/3  00:00:00 ps
$ exec ps
  PID TTY       TIME CMD
  3396 pts/3  00:00:00 bash
  13040 pts/3  00:00:00 ps
$  

[28]
Chapter 10: Process Creation

```c
switch((ret = fork())); {
    case -1: FATAL("fork failed, aborting!
    case 0: /* Child */
        printf("Child process, PID %d:\n" " return %d from fork()\n"
            , getpid(), ret);  
        foo(atoi(argv[1]));
        printf("Child process (\d) done.\n", getpid());
        exit(EXIT_SUCCESS);
    default: /* Parent */
        printf("Parent process, PID %d:\n" " return %d from fork()\n"
            , getpid(), ret);
        bar(atoi(argv[2]));
}
```

```c
switch((ret = fork())); {
    case -1: FATAL("fork failed, aborting!
    case 0: /* Child */
        printf("Child process, PID %d:\n" " return %d from fork()\n"
            , getpid(), ret);
        foo(atoi(argv[1]));
        printf("Child process (\d) done.\n", getpid());
        exit(EXIT_SUCCESS);
    default: /* Parent */
        printf("Parent process, PID %d:\n" " return %d from fork()\n"
            , getpid(), ret);
        bar(atoi(argv[2]));
}```
switch((ret = fork())) {
    case -1: FATAL("fork failed, aborting!\n");
    case 0: /* Child */
        printf("Child process, PID %d:\n" 
            " return %d from fork()\n" , getpid(), ret);
        foo(atoi(argv[1]));
        printf("Child process (%d) done, exiting ...\n", getpid());
        exit(EXIT_SUCCESS);
    default: /* Parent */
        printf("Parent process, PID %d:\n" 
            " return %d from fork()\n" , getpid(), ret);
        bar(atoi(argv[2]));
}
Parent shell PID x

```
simpsh

fork()
```

Parent shell PID x

```
simpsh

wait
```

```
simpsh

exec
```

```
ps

ps executes ...
and dies
```

Predecessor: simpsh
Child: PID y

Successor: ps
PID y

[32]
Chapter 11: Signaling - Part I

[Diagram showing the process of signaling in user and kernel space, including steps such as signal delivery, handling, and control flow.]
Default:
SA_NODEFER Cleared

Process Stack

Call Frame for signal 'n' #1

...main loop signal 'n' #1 occurs ...

Signal 'n' #2 : kept pending!
Signal 'n' #3
Signal 'n' #4
Signal 'n' #5
Signal 'n' #6

Signal Handler function for Signal 'n' (currently handling instance #1)

time (ms)
SA_NODEFER Set

Process Stack

- ...
- ...
- Call Frame for signal 'n' #1
- Call Frame for signal 'n' #2
- Call Frame for signal 'n' #3
- Call Frame for signal 'n' #4
- Call Frame for signal 'n' #5
- Call Frame for signal 'n' #6

main loop signal 'n' #1 occurs

Signal 'n' #2 : kept pending!

Signal Handler function for Signal 'n'

Signal 'n' #3
Signal 'n' #4
Signal 'n' #5
Signal 'n' #6

time (ms)
Chapter 12: Signaling - Part II
Chapter 13: Timers

```bash
$ ./runwalk_timer 5 2
************ Run Walk Timer ************
    Ver 1.0
Get moving... Run for 5 seconds
.....
*** Bzzzz!!! WALK! *** for 2 seconds
...
*** Bzzzz!!! RUN! *** for 5 seconds
.....
*** Bzzzz!!! WALK! *** for 2 seconds
...
*** Bzzzz!!! RUN! *** for 5 seconds
.....
*** Bzzzz!!! WALK! *** for 2 seconds
...
*** Bzzzz!!! RUN! *** for 5 seconds
.....
*** Bzzzz!!! WALK! *** for 2 seconds
```
Chapter 14: Multithreading with Pthreads
Part I - Essentials
pthread_create();
$ 
$ 
$ ./pthreads3 &
[1] 3906 
$ Worker thread #0 running ...
Worker thread #1 running ...
Worker thread #2 running ...

$ ps -LA|grep pthreads3
  3906  3906 pts/0  00:00:00 pthreads3 <defunct>
  3906  3907 pts/0  00:00:00 pthreads3
  3906  3908 pts/0  00:00:00 pthreads3
  3906  3909 pts/0  00:00:00 pthreads3
$ #2: work done, exiting now
#1: work done, exiting now
#0: work done, exiting now

100%  CPU-bound  100%  IO-bound
Chapter 15: Multithreading with Pthreads
Part II - Synchronization

<table>
<thead>
<tr>
<th>Time</th>
<th>Thread A on CPU 0</th>
<th>Thread B on CPU 1</th>
<th>Balance</th>
</tr>
</thead>
<tbody>
<tr>
<td>t0</td>
<td></td>
<td></td>
<td>12000</td>
</tr>
<tr>
<td>t1</td>
<td>read bal</td>
<td></td>
<td>12000</td>
</tr>
<tr>
<td>t2</td>
<td></td>
<td>read bal</td>
<td>12000</td>
</tr>
<tr>
<td>t3</td>
<td>deposit 3000</td>
<td></td>
<td>12000</td>
</tr>
<tr>
<td>t4</td>
<td></td>
<td>deposit 8000</td>
<td>12000</td>
</tr>
<tr>
<td>t5</td>
<td>update bal = 12000+3000</td>
<td></td>
<td>15000</td>
</tr>
<tr>
<td>t6</td>
<td></td>
<td>update bal = 15000+8000</td>
<td>23000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Time</th>
<th>Thread A on CPU 0</th>
<th>Thread B on CPU 1</th>
<th>Balance</th>
</tr>
</thead>
<tbody>
<tr>
<td>t0</td>
<td></td>
<td></td>
<td>12000</td>
</tr>
<tr>
<td>t1</td>
<td>read bal</td>
<td></td>
<td>12000</td>
</tr>
<tr>
<td>t2</td>
<td></td>
<td>read bal</td>
<td>12000</td>
</tr>
<tr>
<td>t3</td>
<td>deposit 3000</td>
<td></td>
<td>12000</td>
</tr>
<tr>
<td>t4</td>
<td></td>
<td>deposit 8000</td>
<td>12000</td>
</tr>
<tr>
<td>t5</td>
<td>update bal = 12000+3000</td>
<td>update bal = 12000+8000</td>
<td>15000</td>
</tr>
<tr>
<td>t6</td>
<td></td>
<td></td>
<td>20000</td>
</tr>
<tr>
<td>Time</td>
<td>Thread A on CPU 0</td>
<td>Thread B on CPU 1</td>
<td>Balance</td>
</tr>
<tr>
<td>------</td>
<td>-------------------</td>
<td>-------------------</td>
<td>---------</td>
</tr>
<tr>
<td>t0</td>
<td></td>
<td>12000</td>
<td></td>
</tr>
<tr>
<td>t1</td>
<td>read bal</td>
<td></td>
<td></td>
</tr>
<tr>
<td>t2</td>
<td>deposit 3000</td>
<td></td>
<td>12000</td>
</tr>
<tr>
<td>t3</td>
<td>update bal = 12000+3000</td>
<td></td>
<td>15000</td>
</tr>
<tr>
<td>t4</td>
<td></td>
<td>read bal</td>
<td>15000</td>
</tr>
<tr>
<td>t5</td>
<td>deposit 6000</td>
<td></td>
<td>15000</td>
</tr>
<tr>
<td>t6</td>
<td></td>
<td>update bal = 15000+8000</td>
<td>23000</td>
</tr>
<tr>
<td>Mutex Type</td>
<td>Robustness</td>
<td>Relock</td>
<td>Unlock When Not Owner</td>
</tr>
<tr>
<td>--------------</td>
<td>-------------</td>
<td>--------------</td>
<td>-----------------------</td>
</tr>
<tr>
<td>NORMAL</td>
<td>non-robust</td>
<td>deadlock</td>
<td>undefined behavior</td>
</tr>
<tr>
<td>NORMAL</td>
<td>robust</td>
<td>deadlock</td>
<td>error returned</td>
</tr>
<tr>
<td>ERRORCHECK</td>
<td>either</td>
<td>error returned</td>
<td>error returned</td>
</tr>
<tr>
<td>RECURSIVE</td>
<td>either</td>
<td>recursive (see below)</td>
<td>error returned</td>
</tr>
<tr>
<td>DEFAULT</td>
<td>non-robust</td>
<td>undefined behavior†</td>
<td>undefined behavior†</td>
</tr>
<tr>
<td>DEFAULT</td>
<td>robust</td>
<td>undefined behavior†</td>
<td>error returned</td>
</tr>
</tbody>
</table>
Chapter 16: Multithreading with Pthreads
Part III

<table>
<thead>
<tr>
<th>Interface</th>
<th>Attribute</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>asctime()</td>
<td>Thread safety</td>
<td>MT-Unsafe race:asctime locale</td>
</tr>
<tr>
<td>asctime_r()</td>
<td>Thread safety</td>
<td>MT-Safe locale</td>
</tr>
<tr>
<td>ctime()</td>
<td>Thread safety</td>
<td>MT-Unsafe race:tmbuf</td>
</tr>
<tr>
<td></td>
<td></td>
<td>race:asctime env locale</td>
</tr>
<tr>
<td>ctime_r(),</td>
<td>Thread safety</td>
<td>MT-Safe env locale</td>
</tr>
<tr>
<td>gmtime_r(),</td>
<td></td>
<td></td>
</tr>
<tr>
<td>localtime_r(),</td>
<td></td>
<td></td>
</tr>
<tr>
<td>mktime()</td>
<td></td>
<td></td>
</tr>
<tr>
<td>gmtime(),</td>
<td>Thread safety</td>
<td>MT-Unsafe race:tmbuf env locale</td>
</tr>
<tr>
<td>localtime()</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
main: pthread_cancel(threadA);

Cancellability State of threadA?

Disables

Cancellation request pending until state becomes Enabled

Enabled [default]

Cancellability Type of threadA?

Asynchronous

Cancellation takes effect

Deferred [default]

Cancellation occurs (threadA terminates) as soon as it hits the next cancellation point in it's code path
Chapter 17: CPU Scheduling on Linux

[Diagram showing the states of a CPU process: R (Runnable), Sleep, T (Stopped), Z (Zombie/Defunct), X (Dead).]

Realtime Priority
- [for SCHED_FIFO and SCHED RR tasks]

Nice value
- [for SCHED_OTHER tasks]
Chapter 18: Advanced File I/O
Process P1

1. `fd = open("myfile", O_RDWR);`
2. `mmap_base = mmap(..., fd, ...);`

Data mapped to process VR

mmap returns base address

Usermode

Kernel

VFS

Memory map the page cache data to process virtual address space

Page Cache

Page cache populated with disk data blocks

Yes

Page Cache hit? No

Allocate memory for page cache and fire off I/O requests (read from disk)

Block Driver

Storage device

Block driver: copies disk data to page cache

[ 53 ]
File I/O Essentials

Process P1

\[ \text{buf} \rightarrow \begin{array}{c}
\text{fwrite} \left( / \text{fprintf} / \text{fputc} \right) \\
\downarrow \\
\text{[write syscall]} \\
\end{array} \]

Usermode
Kernel

VFS

\[ \ldots \]
\[ \text{Kernel Block IO Layers} \]
\[ (a \text{ big black box for now}) \]
\[ \ldots \]

Block Driver

Storage device
fwrite [\ printf / fputc]

stdout buffer

[write syscall]

Usermode
Kernel

VFS

Kernel Block IO Layers
(a big black box for now)

Block Driver

Storage device
$ make mode_def
gcc -O2 -Wall -DUDEBUG -c mode_def.c -o mode_def.o

mode_def.c: In function 'main':
mode_def.c:31:7: warning: implicit declaration of function 'open'; did you mean 'popen'? [-Wimplicit-function-declaration]
  fd = open(argv[1], O_CREAT|O_RDONLY);
    ^~~~
  popen

mode_def.c:31:21: error: 'O_CREAT' undeclared (first use in this function)
  fd = open(argv[1], O_CREAT|O_RDONLY);
        ^~~~~~~

mode_def.c:31:21: note: each undeclared identifier is reported only once for each function it appears in
mode_def.c:31:29: error: 'O_RDONLY' undeclared (first use in this function)
  fd = open(argv[1], O_CREAT|O_RDONLY);
        ^~~~~~~

Makefile:113: recipe for target 'mode_def.o' failed
make: *** [mode_def.o] Error 1
Daemon Processes

It is recommended for new-style daemons to implement the following:

1. If **SIGTERM** is received, shut down the daemon and exit cleanly.

2. If **SIGHUP** is received, reload the configuration files, if this applies.

3. Provide a correct exit code from the main daemon process, as this is used by the init system to detect service errors and problems. It is recommended to follow the exit code scheme as defined in the **LSB recommendations for SysV init scripts**.

4. If possible and applicable, expose the daemon's control interface via the D-Bus IPC system and grab a bus name as last step of initialization.

5. For integration in systemd, provide a `.service` unit file that carries information about starting, stopping and otherwise maintaining the daemon. See **systemd service(5)** for details.

6. As much as possible, rely on the init system's functionality to limit the access of the daemon to files, services and other resources, i.e. in the case of systemd, rely on systemd's resource limit control instead of implementing your own, rely on systemd's privilege dropping code instead of implementing it in the daemon, and similar. See **systemd.exec(5)** for the available controls.

7. If D-Bus is used, make your daemon bus-activatable by supplying a D-Bus service activation configuration file. This has multiple advantages: your daemon may be started lazily on-demand; it may be started in parallel to other daemons requiring it — which maximizes parallelization and boot-up speed; your daemon can be restarted on failure without losing any bus requests, as the bus queues requests for activatable services. See below for details.

8. If your daemon provides services to other local processes or remote clients via a socket, it should be made socket-activatable following the scheme pointed out below. Like D-Bus activation, this enables on-demand starting of services as well as it allows improved parallelization of service start-up. Also, for state-less protocols (such as syslog, DNS), a daemon implementing socket-based activation can be restarted without losing a single request. See below for details.