哈爾濱工業大學

计算机系统

大作业

题　　目　程序人生-Hello’s P2P

专　　业　计算机

学　　号　1180301020

班　　级　1803010

学　　生　李文潇

指 导 教 师　史先俊

计算机科学与技术学院

2019 年 12 月

摘　要

摘要是论文内容的高度概括，应具有独立性和自含性，即不阅读论文的全文，就能获得必要的信息。摘要应包括本论文的目的、主要内容、方法、成果及其理论与实际意义。摘要中不宜使用公式、结构式、图表和非公知公用的符号与术语，不标注引用文献编号，同时避免将摘要写成目录式的内容介绍。

关键词：关键词 1；关键词 2；……；

（摘要 0 分，缺失-1 分，根据内容精彩称都酌情加分 0-1 分）

目 录

第 1 章 概述

1.1 Hello 简介

P2P 是指 from program to process。在 linux 系统中，源文件 hello.c 经 cpp 的预处理生成文本文件 hello.i，经 ccl 编译生成汇编文件 hello.s，经 id 链接生成可执行程序 hello。在输入命令 (./hello) 启动该程序后，在 shell 里 OS 为其 fork，创建子进程，就成为了 process。

020 是指 from zero to zero。hello 程序开始运行成为进程后，OS 为其 execve，映射虚拟内存，调用程序入口时开始载入物理内存，进入 main 函数执行目标代码，然后分配时间片，执行逻辑控制流，执行正确的 IO 管理与信号处理来保证程序的正常运行，以及与用户的交互。hello 程序运行结束后，父进程回收该子进程，内核删除进程产生的相关数据和分配的结构，恢复程序执行前的状态，实现 020。

1.2 环境与工具

Intel(R)_Core(TM)_i7-7700_CPU_@_3.60GHz

Ubuntu18.04.1 LTS

1.3 中间结果

hello.i 预处理后的文本文件，加载了头文件，进行了宏替换，完成条件编译

hello.s 编译后的汇编文件

hello.o 汇编后的可重定位目标文件，将汇编语言翻译成机器语言指令，并将指令打包成可重定位目标文件。

hello 链接产生的可执行目标文件

hello.elf hello 的 ELF 格式文件

1.4 本章小结

本章主要介绍了 hello 文件的 P2P 和 020 过程，旨在使人初步了解程序在计算机

系统中是如何被执行的，对计算机程序执行的原理及有关文件的作用、来源进行阐述，有利于加深人们对计算机相关知识和相关用语的理解。

（第 1 章 0.5 分）

第 2 章 预处理

2.1 预处理的概念与作用

概念：预处理器 cpp 根据以字符#开通的命令修改原始.c 文件，加载头文件，将引用的所有库展开合并为一个文本文件，进行宏替换，完成条件编译。

作用：（1）展开所有宏定义，处理#define x y 格式的语句，用 y 替代 x；

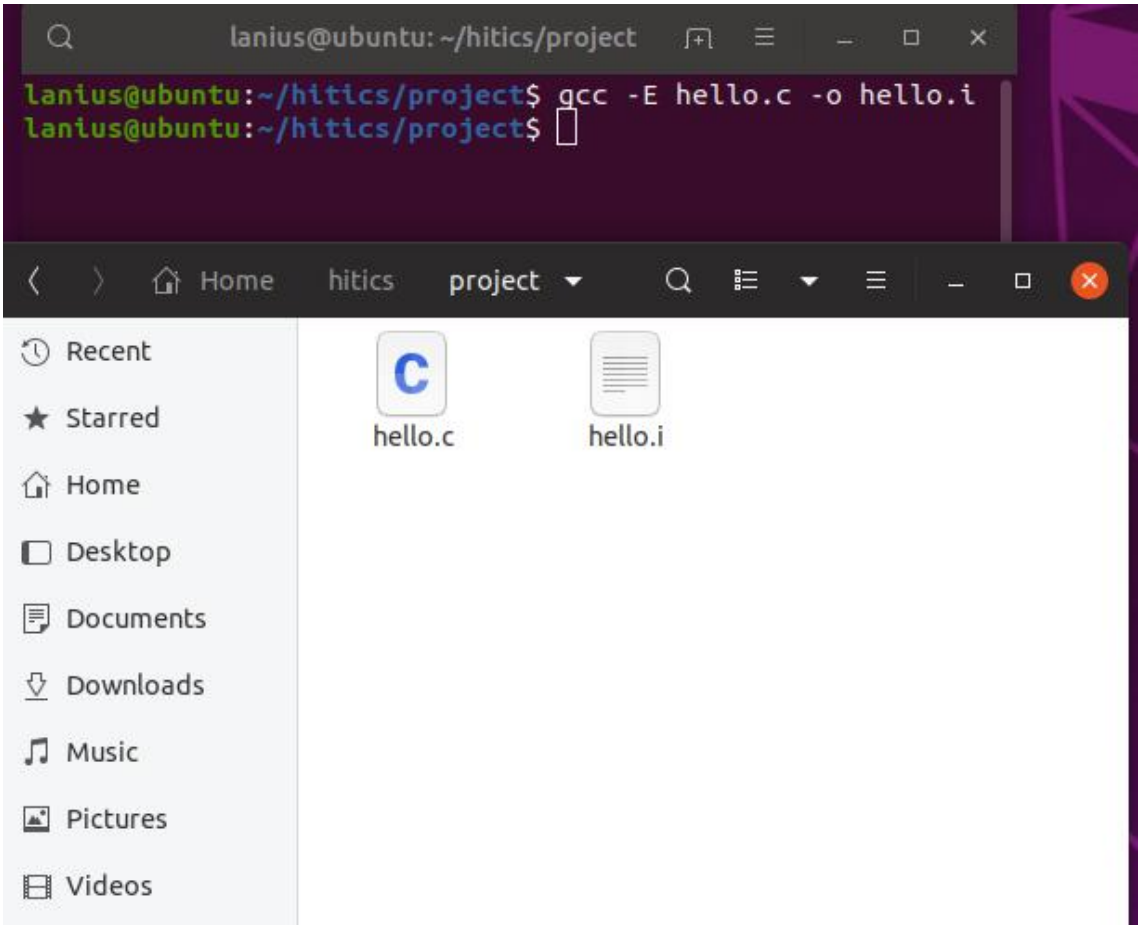
（2）加载头文件，将#include 格式的头文件插入到预编译指令的位置；

（3）完成条件编译，处理#if，#ifdef 格式的语句；

（4）删除注释；

2.2 在 Ubuntu 下预处理的命令

gcc -E -o hello.i hello.c



Hello.i 代码如下

```
# 1 "hello.c"
# 1 "<built-in>"
# 1 "<command-line>"
# 31 "<command-line>"
# 1 "/usr/include/stdc-predef.h" 1 3 4
# 32 "<command-line>" 2
# 1 "hello.c"


# 1 "/usr/include/stdio.h" 1 3 4
# 27 "/usr/include/stdio.h" 3 4
# 1 "/usr/include/x86_64-linux-gnu/bits/libc-header-start.h" 1 3 4
# 33 "/usr/include/x86_64-linux-gnu/bits/libc-header-start.h" 3 4
# 1 "/usr/include/features.h" 1 3 4
# 446 "/usr/include/features.h" 3 4
# 1 "/usr/include/x86_64-linux-gnu/sys/cdefs.h" 1 3 4
# 452 "/usr/include/x86_64-linux-gnu/sys/cdefs.h" 3 4
# 1 "/usr/include/x86_64-linux-gnu/bits/wordsize.h" 1 3 4
# 453 "/usr/include/x86_64-linux-gnu/sys/cdefs.h" 2 3 4
# 1 "/usr/include/x86_64-linux-gnu/bits/long-double.h" 1 3 4
# 454 "/usr/include/x86_64-linux-gnu/sys/cdefs.h" 2 3 4
# 447 "/usr/include/features.h" 2 3 4
# 470 "/usr/include/features.h" 3 4
# 1 "/usr/include/x86_64-linux-gnu/gnu/stubs.h" 1 3 4
# 10 "/usr/include/x86_64-linux-gnu/gnu/stubs.h" 3 4
# 1 "/usr/include/x86_64-linux-gnu/gnu/stubs-64.h" 1 3 4
# 11 "/usr/include/x86_64-linux-gnu/gnu/stubs.h" 2 3 4
# 471 "/usr/include/features.h" 2 3 4
# 34 "/usr/include/x86_64-linux-gnu/bits/libc-header-start.h" 2 3 4
# 28 "/usr/include/stdio.h" 2 3 4



# 1 "/usr/lib/gcc/x86_64-linux-gnu/8/include/stddef.h" 1 3 4
# 216 "/usr/lib/gcc/x86_64-linux-gnu/8/include/stddef.h" 3 4

# 216 "/usr/lib/gcc/x86_64-linux-gnu/8/include/stddef.h" 3 4
typedef long unsigned int size_t;
# 34 "/usr/include/stdio.h" 2 3 4


# 1 "/usr/lib/gcc/x86_64-linux-gnu/8/include/stdarg.h" 1 3 4
# 40 "/usr/lib/gcc/x86_64-linux-gnu/8/include/stdarg.h" 3 4
```

```
49. typedef __builtin_va_list __gnuc_va_list;
50. # 37 "/usr/include/stdio.h" 2 3 4
51. 
52. # 1 "/usr/include/x86_64-linux-gnu/bits/types.h" 1 3 4
53. # 27 "/usr/include/x86_64-linux-gnu/bits/types.h" 3 4
54. # 1 "/usr/include/x86_64-linux-gnu/bits/wordsize.h" 1 3 4
55. # 28 "/usr/include/x86_64-linux-gnu/bits/types.h" 2 3 4
56. # 1 "/usr/include/x86_64-linux-gnu/bits/timesize.h" 1 3 4
57. # 29 "/usr/include/x86_64-linux-gnu/bits/types.h" 2 3 4
58. 
59. 
60. typedef unsigned char __u_char;
61. typedef unsigned short int __u_short;
62. typedef unsigned int __u_int;
63. typedef unsigned long int __u_long;
64. 
65. 
66. typedef signed char __int8_t;
67. typedef unsigned char __uint8_t;
68. typedef signed short int __int16_t;
69. typedef unsigned short int __uint16_t;
70. typedef signed int __int32_t;
71. typedef unsigned int __uint32_t;
72. 
73. typedef signed long int __int64_t;
74. typedef unsigned long int __uint64_t;
75. 
76. 
77. 
78. 
79. 
80. 
81. typedef __int8_t __int_least8_t;
82. typedef __uint8_t __uint_least8_t;
83. typedef __int16_t __int_least16_t;
84. typedef __uint16_t __uint_least16_t;
85. typedef __int32_t __int_least32_t;
86. typedef __uint32_t __uint_least32_t;
87. typedef __int64_t __int_least64_t;
88. typedef __uint64_t __uint_least64_t;
89. 
90. 
91. 
92. typedef long int __quad_t;
93. typedef unsigned long int __u_quad_t;
94. 
95. 
96. 
```

```
97.
98.
99.
100.
101. typedef long int __intmax_t;
102. typedef unsigned long int __uintmax_t;
103. # 141 "/usr/include/x86_64-linux-gnu/bits/types.h" 3 4
104. # 1 "/usr/include/x86_64-linux-gnu/bits/typesizes.h" 1 3 4
105. # 142 "/usr/include/x86_64-linux-gnu/bits/types.h" 2 3 4
106. # 1 "/usr/include/x86_64-linux-gnu/bits/time64.h" 1 3 4
107. # 143 "/usr/include/x86_64-linux-gnu/bits/types.h" 2 3 4
108.
109.
110. typedef unsigned long int __dev_t;
111.typedef unsigned int __uid_t;
112. typedef unsigned int __gid_t;
113. typedef unsigned long int __ino_t;
114. typedef unsigned long int __ino64_t;
115. typedef unsigned int __mode_t;
116. typedef unsigned long int __nlink_t;
117. typedef long int __off_t;
118. typedef long int __off64_t;
119. typedef int __pid_t;
120. typedef struct { int __val[2]; } __fsid_t;
121. typedef long int __clock_t;
122. typedef unsigned long int __rlim_t;
123. typedef unsigned long int __rlim64_t;
124. typedef unsigned int __id_t;
125. typedef long int __time_t;
126. typedef unsigned int __useconds_t;
127. typedef long int __suseconds_t;
128.
129. typedef int __daddr_t;
130. typedef int __key_t;
131.
132.
133. typedef int __clockid_t;
134.
135.
136. typedef void * __timer_t;
137.
138.
139. typedef long int __blksize_t;
140.
141.
142.
143.
144. typedef long int __blkcnt_t;
```

```
145. typedef long int __blkcnt64_t;
146. 
147. 
148. typedef unsigned long int __fsblkcnt_t;
149. typedef unsigned long int __fsblkcnt64_t;
150. 
151. 
152. typedef unsigned long int __fsfilcnt_t;
153. typedef unsigned long int __fsfilcnt64_t;
154. 
155. 
156. typedef long int __fsword_t;
157. 
158. typedef long int __ssize_t;
159. 
160. 
161. typedef long int __syscall_slong_t;
162. 
163. typedef unsigned long int __syscall_ulong_t;
164. 
165. 
166. 
167. typedef __off64_t __loff_t;
168. typedef char *__caddr_t;
169. 
170. 
171. typedef long int __intptr_t;
172. 
173. 
174. typedef unsigned int __socklen_t;
175. 
176. 
177. 
178. 
179. typedef int __sig_atomic_t;
180. # 39 "/usr/include/stdio.h" 2 3 4
181. # 1 "/usr/include/x86_64-linux-gnu/bits/types/__fpos_t.h" 1 3 4
182. 
183. 
184. 
185. 
186. # 1 "/usr/include/x86_64-linux-gnu/bits/types/__mbstate_t.h" 1 3 4
187. # 13 "/usr/include/x86_64-linux-gnu/bits/types/__mbstate_t.h" 3 4
188. typedef struct
189. {
190.   int __count;
191.   union
192.   {
```

```
193.     unsigned int __wch;
194.     char __wchb[4];
195.   } __value;
196. } __mbstate_t;
197. # 6 "/usr/include/x86_64-linux-gnu/bits/types/__fpos_t.h" 2 3 4
198.
199.
200.
201.
202. typedef struct _G_fpos_t
203. {
204.   __off_t __pos;
205.   __mbstate_t __state;
206. } __fpos_t;
207. # 40 "/usr/include/stdio.h" 2 3 4
208. # 1 "/usr/include/x86_64-linux-gnu/bits/types/__fpos64_t.h" 1 3 4
209. # 10 "/usr/include/x86_64-linux-gnu/bits/types/__fpos64_t.h" 3 4
210. typedef struct _G_fpos64_t
211. {
212.   __off64_t __pos;
213.   __mbstate_t __state;
214. } __fpos64_t;
215. # 41 "/usr/include/stdio.h" 2 3 4
216. # 1 "/usr/include/x86_64-linux-gnu/bits/types/__FILE.h" 1 3 4
217.
218.
219.
220. struct _IO_FILE;
221. typedef struct _IO_FILE __FILE;
222. # 42 "/usr/include/stdio.h" 2 3 4
223. # 1 "/usr/include/x86_64-linux-gnu/bits/types/FILE.h" 1 3 4
224.
225.
226.
227. struct _IO_FILE;
228.
229.
230. typedef struct _IO_FILE FILE;
231. # 43 "/usr/include/stdio.h" 2 3 4
232. # 1 "/usr/include/x86_64-linux-gnu/bits/types/struct_FILE.h" 1 3 4
233. # 35 "/usr/include/x86_64-linux-gnu/bits/types/struct_FILE.h" 3 4
234. struct _IO_FILE;
235. struct _IO_marker;
236. struct _IO_codecvt;
237. struct _IO_wide_data;
238.
239.
240.
```

```
241.
242. typedef void _IO_lock_t;
243.
244.
245.
246.
247.
248. struct _IO_FILE
249. {
250.   int _flags;
251.
252.
253.   char *_IO_read_ptr;
254.   char *_IO_read_end;
255.   char *_IO_read_base;
256.   char *_IO_write_base;
257.   char *_IO_write_ptr;
258.   char *_IO_write_end;
259.   char *_IO_buf_base;
260.   char *_IO_buf_end;
261.
262.
263.   char *_IO_save_base;
264.   char *_IO_backup_base;
265.   char *_IO_save_end;
266.
267.   struct _IO_marker *_markers;
268.
269.   struct _IO_FILE *_chain;
270.
271.   int _fileno;
272.   int _flags2;
273.   __off_t _old_offset;
274.
275.
276.   unsigned short _cur_column;
277.   signed char _vtable_offset;
278.   char _shortbuf[1];
279.
280.   _IO_lock_t *_lock;
281.
282.
283.
284.
285.
286.
287.
288.   __off64_t _offset;
```

```
289.
290.   struct _IO_codecvt *_codecvt;
291.   struct _IO_wide_data *_wide_data;
292.   struct _IO_FILE *_freeres_list;
293.   void *_freeres_buf;
294.   size_t __pad5;
295.   int _mode;
296.
297.   char _unused2[15 * sizeof (int) - 4 * sizeof (void *) - sizeof (size_t)];
298. };
299. # 44 "/usr/include/stdio.h" 2 3 4
300. # 52 "/usr/include/stdio.h" 3 4
301. typedef __gnuc_va_list va_list;
302. # 63 "/usr/include/stdio.h" 3 4
303. typedef __off_t off_t;
304. # 77 "/usr/include/stdio.h" 3 4
305. typedef __ssize_t ssize_t;
306.
307.
308.
309.
310.
311.
312. typedef __fpos_t fpos_t;
313. # 133 "/usr/include/stdio.h" 3 4
314. # 1 "/usr/include/x86_64-linux-gnu/bits/stdio_lim.h" 1 3 4
315. # 134 "/usr/include/stdio.h" 2 3 4
316.
317.
318.
319. extern FILE *stdin;
320. extern FILE *stdout;
321. extern FILE *stderr;
322.
323.
324.
325.
326.
327.
328. extern int remove (const char *__filename) __attribute__ ((__nothrow__ , __le
    af__));
329.
330. extern int rename (const char *__old, const char *__new) __attribute__ ((__no
    throw__ , __leaf__));
331.
332.
333.
334. extern int renameat (int __oldfd, const char *__old, int __newfd,
```

```
335.        const char *__new) __attribute__ ((__nothrow__ , __leaf__));
336. # 173 "/usr/include/stdio.h" 3 4
337. extern FILE *tmpfile (void) ;
338. # 187 "/usr/include/stdio.h" 3 4
339. extern char *tmpnam (char *__s) __attribute__ ((__nothrow__ , __leaf__)) ;
340.
341.
342.
343.
344. extern char *tmpnam_r (char *__s) __attribute__ ((__nothrow__ , __leaf__)) ;
345. # 204 "/usr/include/stdio.h" 3 4
346. extern char *tempnam (const char *__dir, const char *__pfx)
347.      __attribute__ ((__nothrow__ , __leaf__)) __attribute__ ((__malloc__)) ;
348.
349.
350.
351.
352.
353.
354.
355. extern int fclose (FILE *__stream);
356.
357.
358.
359.
360. extern int fflush (FILE *__stream);
361. # 227 "/usr/include/stdio.h" 3 4
362. extern int fflush_unlocked (FILE *__stream);
363. # 246 "/usr/include/stdio.h" 3 4
364. extern FILE *fopen (const char *__restrict __filename,
365.       const char *__restrict __modes) ;
366.
367.
368.
369.
370. extern FILE *freopen (const char *__restrict __filename,
371.         const char *__restrict __modes,
372.         FILE *__restrict __stream) ;
373. # 279 "/usr/include/stdio.h" 3 4
374. extern FILE *fdopen (int __fd, const char *__modes) __attribute__ ((__nothrow
    __ , __leaf__)) ;
375. # 292 "/usr/include/stdio.h" 3 4
376. extern FILE *fmemopen (void *__s, size_t __len, const char *__modes)
377.   __attribute__ ((__nothrow__ , __leaf__)) ;
378.
379.
380.
```

```
381.
382. extern FILE *open_memstream (char **__bufloc, size_t *__sizeloc) __attribute_
     _ ((__nothrow__ , __leaf__)) ;
383.
384.
385.
386.
387.
388. extern void setbuf (FILE *__restrict __stream, char *__restrict __buf) __attr
     ibute__ ((__nothrow__ , __leaf__));
389.
390.
391.
392. extern int setvbuf (FILE *__restrict __stream, char *__restrict __buf,
393.       int __modes, size_t __n) __attribute__ ((__nothrow__ , __leaf__));
394.
395.
396.
397.
398. extern void setbuffer (FILE *__restrict __stream, char *__restrict __buf,
399.         size_t __size) __attribute__ ((__nothrow__ , __leaf__));
400.
401.
402. extern void setlinebuf (FILE *__stream) __attribute__ ((__nothrow__ , __leaf_
     _));
403.
404.
405.
406.
407.
408.
409.
410. extern int fprintf (FILE *__restrict __stream,
411.       const char *__restrict __format, ...);
412.
413.
414.
415.
416. extern int printf (const char *__restrict __format, ...);
417.
418. extern int sprintf (char *__restrict __s,
419.       const char *__restrict __format, ...) __attribute__ ((__nothrow__));
420.
421.
422.
423.
424.
425. extern int vfprintf (FILE *__restrict __s, const char *__restrict __format,
426.        __gnuc_va_list __arg);
```

```
427.
428.
429.
430.
431. extern int vprintf (const char *__restrict __format, __gnuc_va_list __arg);
432.
433. extern int vsprintf (char *__restrict __s, const char *__restrict __format,
434.        __gnuc_va_list __arg) __attribute__ ((__nothrow__));
435.
436.
437.
438. extern int snprintf (char *__restrict __s, size_t __maxlen,
439.        const char *__restrict __format, ...)
440.       __attribute__ ((__nothrow__)) __attribute__ ((__format__ (__printf__, 3,
     4)));
441.
442. extern int vsnprintf (char *__restrict __s, size_t __maxlen,
443.         const char *__restrict __format, __gnuc_va_list __arg)
444.       __attribute__ ((__nothrow__)) __attribute__ ((__format__ (__printf__, 3,
     0)));
445. # 379 "/usr/include/stdio.h" 3 4
446. extern int vdprintf (int __fd, const char *__restrict __fmt,
447.        __gnuc_va_list __arg)
448.       __attribute__ ((__format__ (__printf__, 2, 0)));
449. extern int dprintf (int __fd, const char *__restrict __fmt, ...)
450.       __attribute__ ((__format__ (__printf__, 2, 3)));
451.
452.
453.
454.
455.
456.
457.
458. extern int fscanf (FILE *__restrict __stream,
459.      const char *__restrict __format, ...) ;
460.
461.
462.
463.
464. extern int scanf (const char *__restrict __format, ...) ;
465.
466. extern int sscanf (const char *__restrict __s,
467.       const char *__restrict __format, ...) __attribute__ ((__nothrow__ , __le
     af__));
468.
469.
470.
471.
472.
```

```
473.
474. extern int fscanf (FILE *__restrict __stream, const char *__restrict __format,
     ...) __asm__ ("" "__isoc99_fscanf")
475.
476.                                ;
477. extern int scanf (const char *__restrict __format, ...) __asm__ ("" "__isoc99
   _scanf")
478.                               ;
479. extern int sscanf (const char *__restrict __s, const char *__restrict __forma
   t, ...) __asm__ ("" "__isoc99_sscanf") __attribute__ ((__nothrow__ , __leaf__))

480.
481.                         ;
482. # 432 "/usr/include/stdio.h" 3 4
483. extern int vfscanf (FILE *__restrict __s, const char *__restrict __format,
484.       __gnuc_va_list __arg)
485.      __attribute__ ((__format__ (__scanf__, 2, 0))) ;
486.
487.
488.
489.
490.
491. extern int vscanf (const char *__restrict __format, __gnuc_va_list __arg)
492.      __attribute__ ((__format__ (__scanf__, 1, 0))) ;
493.
494.
495. extern int vsscanf (const char *__restrict __s,
496.       const char *__restrict __format, __gnuc_va_list __arg)
497.       __attribute__ ((__nothrow__ , __leaf__)) __attribute__ ((__format__ (__s
   canf__, 2, 0)));
498.
499.
500.
501.
502. extern int vfscanf (FILE *__restrict __s, const char *__restrict __format, __
   gnuc_va_list __arg) __asm__ ("" "__isoc99_vfscanf")
503.
504.
505.
506.      __attribute__ ((__format__ (__scanf__, 2, 0))) ;
507. extern int vscanf (const char *__restrict __format, __gnuc_va_list __arg) __a
   sm__ ("" "__isoc99_vscanf")
508.
509.      __attribute__ ((__format__ (__scanf__, 1, 0))) ;
510. extern int vsscanf (const char *__restrict __s, const char *__restrict __form
   at, __gnuc_va_list __arg) __asm__ ("" "__isoc99_vsscanf") __attribute__ ((__not
   hrow__ , __leaf__))
511.
512.
513.
```

```
514.      __attribute__ ((__format__ (__scanf__, 2, 0)));
515. # 485 "/usr/include/stdio.h" 3 4
516. extern int fgetc (FILE *__stream);
517. extern int getc (FILE *__stream);
518.
519.
520.
521.
522.
523. extern int getchar (void);
524.
525.
526.
527.
528.
529.
530. extern int getc_unlocked (FILE *__stream);
531. extern int getchar_unlocked (void);
532. # 510 "/usr/include/stdio.h" 3 4
533. extern int fgetc_unlocked (FILE *__stream);
534. # 521 "/usr/include/stdio.h" 3 4
535. extern int fputc (int __c, FILE *__stream);
536. extern int putc (int __c, FILE *__stream);
537.
538.
539.
540.
541.
542. extern int putchar (int __c);
543. # 537 "/usr/include/stdio.h" 3 4
544. extern int fputc_unlocked (int __c, FILE *__stream);
545.
546.
547.
548.
549.
550.
551.
552. extern int putc_unlocked (int __c, FILE *__stream);
553. extern int putchar_unlocked (int __c);
554.
555.
556.
557.
558.
559.
560. extern int getw (FILE *__stream);
561.
```

```
562.
563. extern int putw (int __w, FILE *__stream);
564.
565.
566.
567.
568.
569.
570.
571. extern char *fgets (char *__restrict __s, int __n, FILE *__restrict __stream)

572.      ;
573. # 603 "/usr/include/stdio.h" 3 4
574. extern __ssize_t __getdelim (char **__restrict __lineptr,
575.                               size_t *__restrict __n, int __delimiter,
576.                               FILE *__restrict __stream) ;
577. extern __ssize_t getdelim (char **__restrict __lineptr,
578.                             size_t *__restrict __n, int __delimiter,
579.                             FILE *__restrict __stream) ;
580.
581.
582.
583.
584.
585.
586.
587. extern __ssize_t getline (char **__restrict __lineptr,
588.                           size_t *__restrict __n,
589.                           FILE *__restrict __stream) ;
590.
591.
592.
593.
594.
595.
596.
597. extern int fputs (const char *__restrict __s, FILE *__restrict __stream);
598.
599.
600.
601.
602.
603. extern int puts (const char *__s);
604.
605.
606.
607.
608.
```

```
609.
610. extern int ungetc (int __c, FILE *__stream);
611.
612.
613.
614.
615.
616.
617. extern size_t fread (void *__restrict __ptr, size_t __size,
618.        size_t __n, FILE *__restrict __stream) ;
619.
620.
621.
622.
623. extern size_t fwrite (const void *__restrict __ptr, size_t __size,
624.         size_t __n, FILE *__restrict __s);
625. # 673 "/usr/include/stdio.h" 3 4
626. extern size_t fread_unlocked (void *__restrict __ptr, size_t __size,
627.          size_t __n, FILE *__restrict __stream) ;
628. extern size_t fwrite_unlocked (const void *__restrict __ptr, size_t __size,
629.           size_t __n, FILE *__restrict __stream);
630.
631.
632.
633.
634.
635.
636.
637. extern int fseek (FILE *__stream, long int __off, int __whence);
638.
639.
640.
641.
642. extern long int ftell (FILE *__stream) ;
643.
644.
645.
646.
647. extern void rewind (FILE *__stream);
648. # 707 "/usr/include/stdio.h" 3 4
649. extern int fseeko (FILE *__stream, __off_t __off, int __whence);
650.
651.
652.
653.
654. extern __off_t ftello (FILE *__stream) ;
655. # 731 "/usr/include/stdio.h" 3 4
656. extern int fgetpos (FILE *__restrict __stream, fpos_t *__restrict __pos);
```

```
657.
658.
659.
660.
661. extern int fsetpos (FILE *__stream, const fpos_t *__pos);
662. # 757 "/usr/include/stdio.h" 3 4
663. extern void clearerr (FILE *__stream) __attribute__ ((__nothrow__ , __leaf__))
     ;
664.
665. extern int feof (FILE *__stream) __attribute__ ((__nothrow__ , __leaf__)) ;
666.
667. extern int ferror (FILE *__stream) __attribute__ ((__nothrow__ , __leaf__)) ;
668.
669.
670.
671. extern void clearerr_unlocked (FILE *__stream) __attribute__ ((__nothrow__ ,
     __leaf__));
672. extern int feof_unlocked (FILE *__stream) __attribute__ ((__nothrow__ , __lea
     f__)) ;
673. extern int ferror_unlocked (FILE *__stream) __attribute__ ((__nothrow__ , __l
     eaf__)) ;
674.
675.
676.
677.
678.
679.
680.
681. extern void perror (const char *__s);
682.
683.
684.
685.
686.
687. # 1 "/usr/include/x86_64-linux-gnu/bits/sys_errlist.h" 1 3 4
688. # 26 "/usr/include/x86_64-linux-gnu/bits/sys_errlist.h" 3 4
689. extern int sys_nerr;
690. extern const char *const sys_errlist[];
691. # 782 "/usr/include/stdio.h" 2 3 4
692.
693.
694.
695.
696. extern int fileno (FILE *__stream) __attribute__ ((__nothrow__ , __leaf__)) ;
697.
698.
699.
```

```
700.
701. extern int fileno_unlocked (FILE *__stream) __attribute__ ((__nothrow__ , __l
   eaf__)) ;
702. # 800 "/usr/include/stdio.h" 3 4
703. extern FILE *popen (const char *__command, const char *__modes) ;
704.
705.
706.
707.
708.
709. extern int pclose (FILE *__stream);
710.
711.
712.
713.
714.
715. extern char *ctermid (char *__s) __attribute__ ((__nothrow__ , __leaf__));
716. # 840 "/usr/include/stdio.h" 3 4
717. extern void flockfile (FILE *__stream) __attribute__ ((__nothrow__ , __leaf__)
   );
718.
719.
720.
721. extern int ftrylockfile (FILE *__stream) __attribute__ ((__nothrow__ , __leaf
   __)) ;
722.
723.
724. extern void funlockfile (FILE *__stream) __attribute__ ((__nothrow__ , __leaf
   __));
725. # 858 "/usr/include/stdio.h" 3 4
726. extern int __uflow (FILE *);
727. extern int __overflow (FILE *, int);
728. # 873 "/usr/include/stdio.h" 3 4
729.
730. # 7 "hello.c" 2
731. # 1 "/usr/include/unistd.h" 1 3 4
732. # 27 "/usr/include/unistd.h" 3 4
733.
734. # 202 "/usr/include/unistd.h" 3 4
735. # 1 "/usr/include/x86_64-linux-gnu/bits/posix_opt.h" 1 3 4
736. # 203 "/usr/include/unistd.h" 2 3 4
737.
738.
739.
740. # 1 "/usr/include/x86_64-linux-gnu/bits/environments.h" 1 3 4
741. # 22 "/usr/include/x86_64-linux-gnu/bits/environments.h" 3 4
742. # 1 "/usr/include/x86_64-linux-gnu/bits/wordsize.h" 1 3 4
743. # 23 "/usr/include/x86_64-linux-gnu/bits/environments.h" 2 3 4
744. # 207 "/usr/include/unistd.h" 2 3 4
```

```
# 226 "/usr/include/unistd.h" 3 4
# 1 "/usr/lib/gcc/x86_64-linux-gnu/8/include/stddef.h" 1 3 4
# 227 "/usr/include/unistd.h" 2 3 4



typedef __gid_t gid_t;



typedef __uid_t uid_t;
# 255 "/usr/include/unistd.h" 3 4
typedef __useconds_t useconds_t;



typedef __pid_t pid_t;



typedef __intptr_t intptr_t;



typedef __socklen_t socklen_t;
# 287 "/usr/include/unistd.h" 3 4
extern int access (const char *__name, int __type) __attribute__ ((__nothrow__ , __leaf__)) __attribute__ ((__nonnull__ (1)));
# 304 "/usr/include/unistd.h" 3 4
extern int faccessat (int __fd, const char *__file, int __type, int __flag)
     __attribute__ ((__nothrow__ , __leaf__)) __attribute__ ((__nonnull__ (2))) ;
# 334 "/usr/include/unistd.h" 3 4
extern __off_t lseek (int __fd, __off_t __offset, int __whence) __attribute__ ((__nothrow__ , __leaf__));
# 353 "/usr/include/unistd.h" 3 4
extern int close (int __fd);

```

```
791.
792.
793.
794.
795. extern ssize_t read (int __fd, void *__buf, size_t __nbytes) ;
796.
797.
798.
799.
800.
801. extern ssize_t write (int __fd, const void *__buf, size_t __n) ;
802. # 376 "/usr/include/unistd.h" 3 4
803. extern ssize_t pread (int __fd, void *__buf, size_t __nbytes,
804.         __off_t __offset) ;
805.
806.
807.
808.
809.
810.
811. extern ssize_t pwrite (int __fd, const void *__buf, size_t __n,
812.          __off_t __offset) ;
813. # 417 "/usr/include/unistd.h" 3 4
814. extern int pipe (int __pipedes[2]) __attribute__ ((__nothrow__ , __leaf__)) ;
815. # 432 "/usr/include/unistd.h" 3 4
816. extern unsigned int alarm (unsigned int __seconds) __attribute__ ((__nothrow_
    _ , __leaf__));
817. # 444 "/usr/include/unistd.h" 3 4
818. extern unsigned int sleep (unsigned int __seconds);
819.
820.
821.
822.
823.
824.
825.
826. extern __useconds_t ualarm (__useconds_t __value, __useconds_t __interval)
827.      __attribute__ ((__nothrow__ , __leaf__));
828.
829.
830.
831.
832.
833.
834. extern int usleep (__useconds_t __useconds);
835. # 469 "/usr/include/unistd.h" 3 4
836. extern int pause (void);
```

```
837.
838.
839.
840. extern int chown (const char *__file, __uid_t __owner, __gid_t __group)
841.      __attribute__ ((__nothrow__ , __leaf__)) __attribute__ ((__nonnull__ (1))
     ) ;
842.
843.
844.
845. extern int fchown (int __fd, __uid_t __owner, __gid_t __group) __attribute__
     ((__nothrow__ , __leaf__)) ;
846.
847.
848.
849.
850. extern int lchown (const char *__file, __uid_t __owner, __gid_t __group)
851.      __attribute__ ((__nothrow__ , __leaf__)) __attribute__ ((__nonnull__ (1))
     ) ;
852.
853.
854.
855.
856.
857.
858. extern int fchownat (int __fd, const char *__file, __uid_t __owner,
859.        __gid_t __group, int __flag)
860.      __attribute__ ((__nothrow__ , __leaf__)) __attribute__ ((__nonnull__ (2))
     ) ;
861.
862.
863.
864. extern int chdir (const char *__path) __attribute__ ((__nothrow__ , __leaf__))
     __attribute__ ((__nonnull__ (1))) ;
865.
866.
867.
868. extern int fchdir (int __fd) __attribute__ ((__nothrow__ , __leaf__)) ;
869. # 511 "/usr/include/unistd.h" 3 4
870. extern char *getcwd (char *__buf, size_t __size) __attribute__ ((__nothrow__ ,
     __leaf__)) ;
871. # 525 "/usr/include/unistd.h" 3 4
872. extern char *getwd (char *__buf)
873.      __attribute__ ((__nothrow__ , __leaf__)) __attribute__ ((__nonnull__ (1))
     ) __attribute__ ((__deprecated__)) ;
874.
875.
876.
877.
878. extern int dup (int __fd) __attribute__ ((__nothrow__ , __leaf__)) ;
```

```
879.
880.
881. extern int dup2 (int __fd, int __fd2) __attribute__ ((__nothrow__ , __leaf__))
     ;
882. # 543 "/usr/include/unistd.h" 3 4
883. extern char **__environ;
884.
885.
886.
887.
888.
889.
890.
891. extern int execve (const char *__path, char *const __argv[],
892.       char *const __envp[]) __attribute__ ((__nothrow__ , __leaf__)) __attribu
     te__ ((__nonnull__ (1, 2)));
893.
894.
895.
896.
897. extern int fexecve (int __fd, char *const __argv[], char *const __envp[])
898.      __attribute__ ((__nothrow__ , __leaf__)) __attribute__ ((__nonnull__ (2))
     );
899.
900.
901.
902.
903. extern int execv (const char *__path, char *const __argv[])
904.       __attribute__ ((__nothrow__ , __leaf__)) __attribute__ ((__nonnull__ (1,
     2)));
905.
906.
907.
908. extern int execle (const char *__path, const char *__arg, ...)
909.       __attribute__ ((__nothrow__ , __leaf__)) __attribute__ ((__nonnull__ (1,
     2)));
910.
911.
912.
913. extern int execl (const char *__path, const char *__arg, ...)
914.       __attribute__ ((__nothrow__ , __leaf__)) __attribute__ ((__nonnull__ (1,
     2)));
915.
916.
917.
918. extern int execvp (const char *__file, char *const __argv[])
919.       __attribute__ ((__nothrow__ , __leaf__)) __attribute__ ((__nonnull__ (1,
     2)));
920.
```

```
921.
922.
923.
924. extern int execlp (const char *__file, const char *__arg, ...)
925.       __attribute__ ((__nothrow__ , __leaf__)) __attribute__ ((__nonnull__ (1,
     2)));
926. # 598 "/usr/include/unistd.h" 3 4
927. extern int nice (int __inc) __attribute__ ((__nothrow__ , __leaf__)) ;
928.
929.
930.
931.
932. extern void _exit (int __status) __attribute__ ((__noreturn__));
933.
934.
935.
936.
937.
938. # 1 "/usr/include/x86_64-linux-gnu/bits/confname.h" 1 3 4
939. # 24 "/usr/include/x86_64-linux-gnu/bits/confname.h" 3 4
940. enum
941.   {
942.     _PC_LINK_MAX,
943.
944.     _PC_MAX_CANON,
945.
946.     _PC_MAX_INPUT,
947.
948.     _PC_NAME_MAX,
949.
950.     _PC_PATH_MAX,
951.
952.     _PC_PIPE_BUF,
953.
954.     _PC_CHOWN_RESTRICTED,
955.
956.     _PC_NO_TRUNC,
957.
958.     _PC_VDISABLE,
959.
960.     _PC_SYNC_IO,
961.
962.     _PC_ASYNC_IO,
963.
964.     _PC_PRIO_IO,
965.
966.     _PC_SOCK_MAXBUF,
967.
```

```
968.     _PC_FILESIZEBITS,
969.
970.     _PC_REC_INCR_XFER_SIZE,
971.
972.     _PC_REC_MAX_XFER_SIZE,
973.
974.     _PC_REC_MIN_XFER_SIZE,
975.
976.     _PC_REC_XFER_ALIGN,
977.
978.     _PC_ALLOC_SIZE_MIN,
979.
980.     _PC_SYMLINK_MAX,
981.
982.     _PC_2_SYMLINKS
983.
984.   };
985.
986.
987. enum
988.   {
989.     _SC_ARG_MAX,
990.
991.     _SC_CHILD_MAX,
992.
993.     _SC_CLK_TCK,
994.
995.     _SC_NGROUPS_MAX,
996.
997.     _SC_OPEN_MAX,
998.
999.     _SC_STREAM_MAX,
1000.
1001.         _SC_TZNAME_MAX,
1002.
1003.         _SC_JOB_CONTROL,
1004.
1005.         _SC_SAVED_IDS,
1006.
1007.         _SC_REALTIME_SIGNALS,
1008.
1009.         _SC_PRIORITY_SCHEDULING,
1010.
1011.         _SC_TIMERS,
1012.
1013.         _SC_ASYNCHRONOUS_IO,
1014.
1015.         _SC_PRIORITIZED_IO,
```

```
1016.
1017.       _SC_SYNCHRONIZED_IO,
1018.
1019.       _SC_FSYNC,
1020.
1021.       _SC_MAPPED_FILES,
1022.
1023.       _SC_MEMLOCK,
1024.
1025.       _SC_MEMLOCK_RANGE,
1026.
1027.       _SC_MEMORY_PROTECTION,
1028.
1029.       _SC_MESSAGE_PASSING,
1030.
1031.       _SC_SEMAPHORES,
1032.
1033.       _SC_SHARED_MEMORY_OBJECTS,
1034.
1035.       _SC_AIO_LISTIO_MAX,
1036.
1037.       _SC_AIO_MAX,
1038.
1039.       _SC_AIO_PRIO_DELTA_MAX,
1040.
1041.       _SC_DELAYTIMER_MAX,
1042.
1043.       _SC_MQ_OPEN_MAX,
1044.
1045.       _SC_MQ_PRIO_MAX,
1046.
1047.       _SC_VERSION,
1048.
1049.       _SC_PAGESIZE,
1050.
1051.
1052.       _SC_RTSIG_MAX,
1053.
1054.       _SC_SEM_NSEMS_MAX,
1055.
1056.       _SC_SEM_VALUE_MAX,
1057.
1058.       _SC_SIGQUEUE_MAX,
1059.
1060.       _SC_TIMER_MAX,
1061.
1062.
1063.
```

```
1064.
1065.         _SC_BC_BASE_MAX,
1066.
1067.         _SC_BC_DIM_MAX,
1068.
1069.         _SC_BC_SCALE_MAX,
1070.
1071.         _SC_BC_STRING_MAX,
1072.
1073.         _SC_COLL_WEIGHTS_MAX,
1074.
1075.         _SC_EQUIV_CLASS_MAX,
1076.
1077.         _SC_EXPR_NEST_MAX,
1078.
1079.         _SC_LINE_MAX,
1080.
1081.         _SC_RE_DUP_MAX,
1082.
1083.         _SC_CHARCLASS_NAME_MAX,
1084.
1085.
1086.         _SC_2_VERSION,
1087.
1088.         _SC_2_C_BIND,
1089.
1090.         _SC_2_C_DEV,
1091.
1092.         _SC_2_FORT_DEV,
1093.
1094.         _SC_2_FORT_RUN,
1095.
1096.         _SC_2_SW_DEV,
1097.
1098.         _SC_2_LOCALEDEF,
1099.
1100.
1101.         _SC_PII,
1102.
1103.         _SC_PII_XTI,
1104.
1105.         _SC_PII_SOCKET,
1106.
1107.         _SC_PII_INTERNET,
1108.
1109.         _SC_PII_OSI,
1110.
1111.     _SC_POLL,
```

```
1112.
1113.    _SC_SELECT,
1114.
1115.    _SC_UIO_MAXIOV,
1116.
1117.    _SC_IOV_MAX = _SC_UIO_MAXIOV,
1118.
1119.    _SC_PII_INTERNET_STREAM,
1120.
1121.        _SC_PII_INTERNET_DGRAM,
1122.
1123.        _SC_PII_OSI_COTS,
1124.
1125.        _SC_PII_OSI_CLTS,
1126.
1127.        _SC_PII_OSI_M,
1128.
1129.        _SC_T_IOV_MAX,
1130.
1131.
1132.
1133.        _SC_THREADS,
1134.
1135.        _SC_THREAD_SAFE_FUNCTIONS,
1136.
1137.        _SC_GETGR_R_SIZE_MAX,
1138.
1139.        _SC_GETPW_R_SIZE_MAX,
1140.
1141.        _SC_LOGIN_NAME_MAX,
1142.
1143.        _SC_TTY_NAME_MAX,
1144.
1145.        _SC_THREAD_DESTRUCTOR_ITERATIONS,
1146.
1147.        _SC_THREAD_KEYS_MAX,
1148.
1149.        _SC_THREAD_STACK_MIN,
1150.
1151.        _SC_THREAD_THREADS_MAX,
1152.
1153.        _SC_THREAD_ATTR_STACKADDR,
1154.
1155.        _SC_THREAD_ATTR_STACKSIZE,
1156.
1157.        _SC_THREAD_PRIORITY_SCHEDULING,
1158.
1159.        _SC_THREAD_PRIO_INHERIT,
```

```
1160.
1161.         _SC_THREAD_PRIO_PROTECT,
1162.
1163.         _SC_THREAD_PROCESS_SHARED,
1164.
1165.
1166.         _SC_NPROCESSORS_CONF,
1167.
1168.         _SC_NPROCESSORS_ONLN,
1169.
1170.         _SC_PHYS_PAGES,
1171.
1172.         _SC_AVPHYS_PAGES,
1173.
1174.         _SC_ATEXIT_MAX,
1175.
1176.         _SC_PASS_MAX,
1177.
1178.
1179.         _SC_XOPEN_VERSION,
1180.
1181.         _SC_XOPEN_XCU_VERSION,
1182.
1183.         _SC_XOPEN_UNIX,
1184.
1185.         _SC_XOPEN_CRYPT,
1186.
1187.         _SC_XOPEN_ENH_I18N,
1188.
1189.         _SC_XOPEN_SHM,
1190.
1191.
1192.         _SC_2_CHAR_TERM,
1193.
1194.         _SC_2_C_VERSION,
1195.
1196.         _SC_2_UPE,
1197.
1198.
1199.         _SC_XOPEN_XPG2,
1200.
1201.         _SC_XOPEN_XPG3,
1202.
1203.         _SC_XOPEN_XPG4,
1204.
1205.
1206.         _SC_CHAR_BIT,
1207.
```

```
1208.         _SC_CHAR_MAX,
1209.
1210.         _SC_CHAR_MIN,
1211.
1212.         _SC_INT_MAX,
1213.
1214.         _SC_INT_MIN,
1215.
1216.         _SC_LONG_BIT,
1217.
1218.         _SC_WORD_BIT,
1219.
1220.         _SC_MB_LEN_MAX,
1221.
1222.         _SC_NZERO,
1223.
1224.         _SC_SSIZE_MAX,
1225.
1226.         _SC_SCHAR_MAX,
1227.
1228.         _SC_SCHAR_MIN,
1229.
1230.         _SC_SHRT_MAX,
1231.
1232.         _SC_SHRT_MIN,
1233.
1234.         _SC_UCHAR_MAX,
1235.
1236.         _SC_UINT_MAX,
1237.
1238.         _SC_ULONG_MAX,
1239.
1240.         _SC_USHRT_MAX,
1241.
1242.
1243.         _SC_NL_ARGMAX,
1244.
1245.         _SC_NL_LANGMAX,
1246.
1247.         _SC_NL_MSGMAX,
1248.
1249.         _SC_NL_NMAX,
1250.
1251.         _SC_NL_SETMAX,
1252.
1253.         _SC_NL_TEXTMAX,
1254.
1255.
```

```
1256.         _SC_XBS5_ILP32_OFF32,
1257.
1258.         _SC_XBS5_ILP32_OFFBIG,
1259.
1260.         _SC_XBS5_LP64_OFF64,
1261.
1262.         _SC_XBS5_LPBIG_OFFBIG,
1263.
1264.
1265.         _SC_XOPEN_LEGACY,
1266.
1267.         _SC_XOPEN_REALTIME,
1268.
1269.         _SC_XOPEN_REALTIME_THREADS,
1270.
1271.
1272.         _SC_ADVISORY_INFO,
1273.
1274.         _SC_BARRIERS,
1275.
1276.         _SC_BASE,
1277.
1278.         _SC_C_LANG_SUPPORT,
1279.
1280.         _SC_C_LANG_SUPPORT_R,
1281.
1282.         _SC_CLOCK_SELECTION,
1283.
1284.         _SC_CPUTIME,
1285.
1286.         _SC_THREAD_CPUTIME,
1287.
1288.         _SC_DEVICE_IO,
1289.
1290.         _SC_DEVICE_SPECIFIC,
1291.
1292.         _SC_DEVICE_SPECIFIC_R,
1293.
1294.         _SC_FD_MGMT,
1295.
1296.         _SC_FIFO,
1297.
1298.         _SC_PIPE,
1299.
1300.         _SC_FILE_ATTRIBUTES,
1301.
1302.         _SC_FILE_LOCKING,
1303.
```

```
1304.         _SC_FILE_SYSTEM,
1305.
1306.         _SC_MONOTONIC_CLOCK,
1307.
1308.         _SC_MULTI_PROCESS,
1309.
1310.         _SC_SINGLE_PROCESS,
1311.
1312.         _SC_NETWORKING,
1313.
1314.         _SC_READER_WRITER_LOCKS,
1315.
1316.         _SC_SPIN_LOCKS,
1317.
1318.         _SC_REGEXP,
1319.
1320.         _SC_REGEX_VERSION,
1321.
1322.         _SC_SHELL,
1323.
1324.         _SC_SIGNALS,
1325.
1326.         _SC_SPAWN,
1327.
1328.         _SC_SPORADIC_SERVER,
1329.
1330.         _SC_THREAD_SPORADIC_SERVER,
1331.
1332.         _SC_SYSTEM_DATABASE,
1333.
1334.         _SC_SYSTEM_DATABASE_R,
1335.
1336.         _SC_TIMEOUTS,
1337.
1338.         _SC_TYPED_MEMORY_OBJECTS,
1339.
1340.         _SC_USER_GROUPS,
1341.
1342.         _SC_USER_GROUPS_R,
1343.
1344.         _SC_2_PBS,
1345.
1346.         _SC_2_PBS_ACCOUNTING,
1347.
1348.         _SC_2_PBS_LOCATE,
1349.
1350.         _SC_2_PBS_MESSAGE,
1351.
```

```
1352.         _SC_2_PBS_TRACK,
1353.
1354.         _SC_SYMLOOP_MAX,
1355.
1356.         _SC_STREAMS,
1357.
1358.         _SC_2_PBS_CHECKPOINT,
1359.
1360.
1361.         _SC_V6_ILP32_OFF32,
1362.
1363.         _SC_V6_ILP32_OFFBIG,
1364.
1365.         _SC_V6_LP64_OFF64,
1366.
1367.         _SC_V6_LPBIG_OFFBIG,
1368.
1369.
1370.         _SC_HOST_NAME_MAX,
1371.
1372.         _SC_TRACE,
1373.
1374.         _SC_TRACE_EVENT_FILTER,
1375.
1376.         _SC_TRACE_INHERIT,
1377.
1378.         _SC_TRACE_LOG,
1379.
1380.
1381.         _SC_LEVEL1_ICACHE_SIZE,
1382.
1383.         _SC_LEVEL1_ICACHE_ASSOC,
1384.
1385.         _SC_LEVEL1_ICACHE_LINESIZE,
1386.
1387.         _SC_LEVEL1_DCACHE_SIZE,
1388.
1389.         _SC_LEVEL1_DCACHE_ASSOC,
1390.
1391.         _SC_LEVEL1_DCACHE_LINESIZE,
1392.
1393.         _SC_LEVEL2_CACHE_SIZE,
1394.
1395.         _SC_LEVEL2_CACHE_ASSOC,
1396.
1397.         _SC_LEVEL2_CACHE_LINESIZE,
1398.
1399.         _SC_LEVEL3_CACHE_SIZE,
```

```
1400.
1401.         _SC_LEVEL3_CACHE_ASSOC,
1402.
1403.         _SC_LEVEL3_CACHE_LINESIZE,
1404.
1405.         _SC_LEVEL4_CACHE_SIZE,
1406.
1407.         _SC_LEVEL4_CACHE_ASSOC,
1408.
1409.         _SC_LEVEL4_CACHE_LINESIZE,
1410.
1411.
1412.
1413.         _SC_IPV6 = _SC_LEVEL1_ICACHE_SIZE + 50,
1414.
1415.         _SC_RAW_SOCKETS,
1416.
1417.
1418.         _SC_V7_ILP32_OFF32,
1419.
1420.         _SC_V7_ILP32_OFFBIG,
1421.
1422.         _SC_V7_LP64_OFF64,
1423.
1424.         _SC_V7_LPBIG_OFFBIG,
1425.
1426.
1427.         _SC_SS_REPL_MAX,
1428.
1429.
1430.         _SC_TRACE_EVENT_NAME_MAX,
1431.
1432.         _SC_TRACE_NAME_MAX,
1433.
1434.         _SC_TRACE_SYS_MAX,
1435.
1436.         _SC_TRACE_USER_EVENT_MAX,
1437.
1438.
1439.         _SC_XOPEN_STREAMS,
1440.
1441.
1442.         _SC_THREAD_ROBUST_PRIO_INHERIT,
1443.
1444.         _SC_THREAD_ROBUST_PRIO_PROTECT
1445.
1446.       };
1447.
```

```
1448.
1449.   enum
1450.     {
1451.       _CS_PATH,
1452.
1453.
1454.       _CS_V6_WIDTH_RESTRICTED_ENVS,
1455.
1456.
1457.
1458.       _CS_GNU_LIBC_VERSION,
1459.
1460.       _CS_GNU_LIBPTHREAD_VERSION,
1461.
1462.
1463.       _CS_V5_WIDTH_RESTRICTED_ENVS,
1464.
1465.
1466.
1467.       _CS_V7_WIDTH_RESTRICTED_ENVS,
1468.
1469.
1470.
1471.       _CS_LFS_CFLAGS = 1000,
1472.
1473.       _CS_LFS_LDFLAGS,
1474.
1475.       _CS_LFS_LIBS,
1476.
1477.       _CS_LFS_LINTFLAGS,
1478.
1479.       _CS_LFS64_CFLAGS,
1480.
1481.       _CS_LFS64_LDFLAGS,
1482.
1483.       _CS_LFS64_LIBS,
1484.
1485.       _CS_LFS64_LINTFLAGS,
1486.
1487.
1488.       _CS_XBS5_ILP32_OFF32_CFLAGS = 1100,
1489.
1490.       _CS_XBS5_ILP32_OFF32_LDFLAGS,
1491.
1492.       _CS_XBS5_ILP32_OFF32_LIBS,
1493.
1494.       _CS_XBS5_ILP32_OFF32_LINTFLAGS,
1495.
```

```
1496.         _CS_XBS5_ILP32_OFFBIG_CFLAGS,
1497.
1498.         _CS_XBS5_ILP32_OFFBIG_LDFLAGS,
1499.
1500.         _CS_XBS5_ILP32_OFFBIG_LIBS,
1501.
1502.         _CS_XBS5_ILP32_OFFBIG_LINTFLAGS,
1503.
1504.         _CS_XBS5_LP64_OFF64_CFLAGS,
1505.
1506.         _CS_XBS5_LP64_OFF64_LDFLAGS,
1507.
1508.         _CS_XBS5_LP64_OFF64_LIBS,
1509.
1510.         _CS_XBS5_LP64_OFF64_LINTFLAGS,
1511.
1512.         _CS_XBS5_LPBIG_OFFBIG_CFLAGS,
1513.
1514.         _CS_XBS5_LPBIG_OFFBIG_LDFLAGS,
1515.
1516.         _CS_XBS5_LPBIG_OFFBIG_LIBS,
1517.
1518.         _CS_XBS5_LPBIG_OFFBIG_LINTFLAGS,
1519.
1520.
1521.         _CS_POSIX_V6_ILP32_OFF32_CFLAGS,
1522.
1523.         _CS_POSIX_V6_ILP32_OFF32_LDFLAGS,
1524.
1525.         _CS_POSIX_V6_ILP32_OFF32_LIBS,
1526.
1527.         _CS_POSIX_V6_ILP32_OFF32_LINTFLAGS,
1528.
1529.         _CS_POSIX_V6_ILP32_OFFBIG_CFLAGS,
1530.
1531.         _CS_POSIX_V6_ILP32_OFFBIG_LDFLAGS,
1532.
1533.         _CS_POSIX_V6_ILP32_OFFBIG_LIBS,
1534.
1535.         _CS_POSIX_V6_ILP32_OFFBIG_LINTFLAGS,
1536.
1537.         _CS_POSIX_V6_LP64_OFF64_CFLAGS,
1538.
1539.         _CS_POSIX_V6_LP64_OFF64_LDFLAGS,
1540.
1541.         _CS_POSIX_V6_LP64_OFF64_LIBS,
1542.
1543.         _CS_POSIX_V6_LP64_OFF64_LINTFLAGS,
```

```
1544.
1545.         _CS_POSIX_V6_LPBIG_OFFBIG_CFLAGS,
1546.
1547.         _CS_POSIX_V6_LPBIG_OFFBIG_LDFLAGS,
1548.
1549.         _CS_POSIX_V6_LPBIG_OFFBIG_LIBS,
1550.
1551.         _CS_POSIX_V6_LPBIG_OFFBIG_LINTFLAGS,
1552.
1553.
1554.         _CS_POSIX_V7_ILP32_OFF32_CFLAGS,
1555.
1556.         _CS_POSIX_V7_ILP32_OFF32_LDFLAGS,
1557.
1558.         _CS_POSIX_V7_ILP32_OFF32_LIBS,
1559.
1560.         _CS_POSIX_V7_ILP32_OFF32_LINTFLAGS,
1561.
1562.         _CS_POSIX_V7_ILP32_OFFBIG_CFLAGS,
1563.
1564.         _CS_POSIX_V7_ILP32_OFFBIG_LDFLAGS,
1565.
1566.         _CS_POSIX_V7_ILP32_OFFBIG_LIBS,
1567.
1568.         _CS_POSIX_V7_ILP32_OFFBIG_LINTFLAGS,
1569.
1570.         _CS_POSIX_V7_LP64_OFF64_CFLAGS,
1571.
1572.         _CS_POSIX_V7_LP64_OFF64_LDFLAGS,
1573.
1574.         _CS_POSIX_V7_LP64_OFF64_LIBS,
1575.
1576.         _CS_POSIX_V7_LP64_OFF64_LINTFLAGS,
1577.
1578.         _CS_POSIX_V7_LPBIG_OFFBIG_CFLAGS,
1579.
1580.         _CS_POSIX_V7_LPBIG_OFFBIG_LDFLAGS,
1581.
1582.         _CS_POSIX_V7_LPBIG_OFFBIG_LIBS,
1583.
1584.         _CS_POSIX_V7_LPBIG_OFFBIG_LINTFLAGS,
1585.
1586.
1587.         _CS_V6_ENV,
1588.
1589.         _CS_V7_ENV
1590.
1591.     };
```

```
1592.     # 610 "/usr/include/unistd.h" 2 3 4
1593.
1594.
1595.     extern long int pathconf (const char *__path, int __name)
1596.          __attribute__ ((__nothrow__ , __leaf__)) __attribute__ ((__nonnull__
     (1)));
1597.
1598.
1599.     extern long int fpathconf (int __fd, int __name) __attribute__ ((__nothro
    w__ , __leaf__));
1600.
1601.
1602.     extern long int sysconf (int __name) __attribute__ ((__nothrow__ , __leaf
    __));
1603.
1604.
1605.
1606.     extern size_t confstr (int __name, char *__buf, size_t __len) __attribute
    __ ((__nothrow__ , __leaf__));
1607.
1608.
1609.
1610.
1611.     extern __pid_t getpid (void) __attribute__ ((__nothrow__ , __leaf__));
1612.
1613.
1614.     extern __pid_t getppid (void) __attribute__ ((__nothrow__ , __leaf__));
1615.
1616.
1617.     extern __pid_t getpgrp (void) __attribute__ ((__nothrow__ , __leaf__));
1618.
1619.
1620.     extern __pid_t __getpgid (__pid_t __pid) __attribute__ ((__nothrow__ , __
    leaf__));
1621.
1622.     extern __pid_t getpgid (__pid_t __pid) __attribute__ ((__nothrow__ , __le
    af__));
1623.
1624.
1625.
1626.
1627.
1628.
1629.     extern int setpgid (__pid_t __pid, __pid_t __pgid) __attribute__ ((__noth
    row__ , __leaf__));
1630.     # 660 "/usr/include/unistd.h" 3 4
1631.     extern int setpgrp (void) __attribute__ ((__nothrow__ , __leaf__));
1632.
1633.
```

```
1634.
1635.
1636.
1637.
1638.    extern __pid_t setsid (void) __attribute__ ((__nothrow__ , __leaf__));
1639.
1640.
1641.
1642.    extern __pid_t getsid (__pid_t __pid) __attribute__ ((__nothrow__ , __lea
   f__));
1643.
1644.
1645.
1646.    extern __uid_t getuid (void) __attribute__ ((__nothrow__ , __leaf__));
1647.
1648.
1649.    extern __uid_t geteuid (void) __attribute__ ((__nothrow__ , __leaf__));
1650.
1651.
1652.    extern __gid_t getgid (void) __attribute__ ((__nothrow__ , __leaf__));
1653.
1654.
1655.    extern __gid_t getegid (void) __attribute__ ((__nothrow__ , __leaf__));
1656.
1657.
1658.
1659.
1660.    extern int getgroups (int __size, __gid_t __list[]) __attribute__ ((__not
   hrow__ , __leaf__)) ;
1661.    # 700 "/usr/include/unistd.h" 3 4
1662.    extern int setuid (__uid_t __uid) __attribute__ ((__nothrow__ , __leaf__))
    ;
1663.
1664.
1665.
1666.
1667.    extern int setreuid (__uid_t __ruid, __uid_t __euid) __attribute__ ((__no
   throw__ , __leaf__)) ;
1668.
1669.
1670.
1671.
1672.    extern int seteuid (__uid_t __uid) __attribute__ ((__nothrow__ , __leaf__)
   ) ;
1673.
1674.
1675.
1676.
1677.
```

```
1678.
1679.    extern int setgid (__gid_t __gid) __attribute__ ((__nothrow__ , __leaf__))
     ;
1680.
1681.
1682.
1683.
1684.    extern int setregid (__gid_t __rgid, __gid_t __egid) __attribute__ ((__no
    throw__ , __leaf__)) ;
1685.
1686.
1687.
1688.
1689.    extern int setegid (__gid_t __gid) __attribute__ ((__nothrow__ , __leaf__)
    ) ;
1690.    # 756 "/usr/include/unistd.h" 3 4
1691.    extern __pid_t fork (void) __attribute__ ((__nothrow__));
1692.
1693.
1694.
1695.
1696.
1697.
1698.
1699.    extern __pid_t vfork (void) __attribute__ ((__nothrow__ , __leaf__));
1700.
1701.
1702.
1703.
1704.
1705.    extern char *ttyname (int __fd) __attribute__ ((__nothrow__ , __leaf__));

1706.
1707.
1708.
1709.    extern int ttyname_r (int __fd, char *__buf, size_t __buflen)
1710.         __attribute__ ((__nothrow__ , __leaf__)) __attribute__ ((__nonnull__
     (2))) ;
1711.
1712.
1713.
1714.    extern int isatty (int __fd) __attribute__ ((__nothrow__ , __leaf__));
1715.
1716.
1717.
1718.
1719.    extern int ttyslot (void) __attribute__ ((__nothrow__ , __leaf__));
1720.
1721.
```

```
1722.
1723.
1724.    extern int link (const char *__from, const char *__to)
1725.         __attribute__ ((__nothrow__ , __leaf__)) __attribute__ ((__nonnull__
     (1, 2))) ;
1726.
1727.
1728.
1729.
1730.    extern int linkat (int __fromfd, const char *__from, int __tofd,
1731.         const char *__to, int __flags)
1732.         __attribute__ ((__nothrow__ , __leaf__)) __attribute__ ((__nonnull__
     (2, 4))) ;
1733.
1734.
1735.
1736.
1737.    extern int symlink (const char *__from, const char *__to)
1738.         __attribute__ ((__nothrow__ , __leaf__)) __attribute__ ((__nonnull__
     (1, 2))) ;
1739.
1740.
1741.
1742.
1743.    extern ssize_t readlink (const char *__restrict __path,
1744.        char *__restrict __buf, size_t __len)
1745.         __attribute__ ((__nothrow__ , __leaf__)) __attribute__ ((__nonnull__
     (1, 2))) ;
1746.
1747.
1748.
1749.
1750.    extern int symlinkat (const char *__from, int __tofd,
1751.              const char *__to) __attribute__ ((__nothrow__ , __leaf__)) __attr
    ibute__ ((__nonnull__ (1, 3))) ;
1752.
1753.
1754.    extern ssize_t readlinkat (int __fd, const char *__restrict __path,
1755.         char *__restrict __buf, size_t __len)
1756.         __attribute__ ((__nothrow__ , __leaf__)) __attribute__ ((__nonnull__
     (2, 3))) ;
1757.
1758.
1759.
1760.    extern int unlink (const char *__name) __attribute__ ((__nothrow__ , __le
    af__)) __attribute__ ((__nonnull__ (1)));
1761.
1762.
1763.
```

```
1764.     extern int unlinkat (int __fd, const char *__name, int __flag)
1765.          __attribute__ ((__nothrow__ , __leaf__)) __attribute__ ((__nonnull__
     (2)));
1766.
1767.
1768.
1769.     extern int rmdir (const char *__path) __attribute__ ((__nothrow__ , __lea
    f__)) __attribute__ ((__nonnull__ (1)));
1770.
1771.
1772.
1773.     extern __pid_t tcgetpgrp (int __fd) __attribute__ ((__nothrow__ , __leaf_
    _));
1774.
1775.
1776.     extern int tcsetpgrp (int __fd, __pid_t __pgrp_id) __attribute__ ((__noth
    row__ , __leaf__));
1777.
1778.
1779.
1780.
1781.
1782.
1783.     extern char *getlogin (void);
1784.
1785.
1786.
1787.
1788.
1789.
1790.
1791.     extern int getlogin_r (char *__name, size_t __name_len) __attribute__ ((_
    _nonnull__ (1)));
1792.
1793.
1794.
1795.
1796.     extern int setlogin (const char *__name) __attribute__ ((__nothrow__ , __
    leaf__)) __attribute__ ((__nonnull__ (1)));
1797.
1798.
1799.
1800.
1801.
1802.
1803.
1804.     # 1 "/usr/include/x86_64-linux-gnu/bits/getopt_posix.h" 1 3 4
1805.     # 27 "/usr/include/x86_64-linux-gnu/bits/getopt_posix.h" 3 4
1806.     # 1 "/usr/include/x86_64-linux-gnu/bits/getopt_core.h" 1 3 4
```

```
1807.     # 28 "/usr/include/x86_64-linux-gnu/bits/getopt_core.h" 3 4
1808.
1809.
1810.
1811.
1812.
1813.
1814.
1815.
1816.     extern char *optarg;
1817.     # 50 "/usr/include/x86_64-linux-gnu/bits/getopt_core.h" 3 4
1818.     extern int optind;
1819.
1820.
1821.
1822.
1823.     extern int opterr;
1824.
1825.
1826.
1827.     extern int optopt;
1828.     # 91 "/usr/include/x86_64-linux-gnu/bits/getopt_core.h" 3 4
1829.     extern int getopt (int ___argc, char *const *___argv, const char *__short
    opts)
1830.            __attribute__ ((__nothrow__ , __leaf__)) __attribute__ ((__nonnull
    __ (2, 3)));
1831.
1832.
1833.     # 28 "/usr/include/x86_64-linux-gnu/bits/getopt_posix.h" 2 3 4
1834.
1835.
1836.     # 49 "/usr/include/x86_64-linux-gnu/bits/getopt_posix.h" 3 4
1837.
1838.     # 870 "/usr/include/unistd.h" 2 3 4
1839.
1840.
1841.
1842.
1843.
1844.
1845.
1846.     extern int gethostname (char *__name, size_t __len) __attribute__ ((__not
    hrow__ , __leaf__)) __attribute__ ((__nonnull__ (1)));
1847.
1848.
1849.
1850.
1851.
1852.
```

```
1853.    extern int sethostname (const char *__name, size_t __len)
1854.         __attribute__ ((__nothrow__ , __leaf__)) __attribute__ ((__nonnull__
    (1))) ;
1855.
1856.
1857.
1858.    extern int sethostid (long int __id) __attribute__ ((__nothrow__ , __leaf
   __)) ;
1859.
1860.
1861.
1862.
1863.
1864.    extern int getdomainname (char *__name, size_t __len)
1865.         __attribute__ ((__nothrow__ , __leaf__)) __attribute__ ((__nonnull__
    (1))) ;
1866.    extern int setdomainname (const char *__name, size_t __len)
1867.         __attribute__ ((__nothrow__ , __leaf__)) __attribute__ ((__nonnull__
    (1))) ;
1868.
1869.
1870.
1871.
1872.
1873.    extern int vhangup (void) __attribute__ ((__nothrow__ , __leaf__));
1874.
1875.
1876.    extern int revoke (const char *__file) __attribute__ ((__nothrow__ , __le
   af__)) __attribute__ ((__nonnull__ (1))) ;
1877.
1878.
1879.
1880.
1881.
1882.
1883.
1884.    extern int profil (unsigned short int *__sample_buffer, size_t __size,
1885.         size_t __offset, unsigned int __scale)
1886.         __attribute__ ((__nothrow__ , __leaf__)) __attribute__ ((__nonnull__
    (1)));
1887.
1888.
1889.
1890.
1891.
1892.    extern int acct (const char *__name) __attribute__ ((__nothrow__ , __leaf
   __));
1893.
1894.
```

```
1895.
1896.    extern char *getusershell (void) __attribute__ ((__nothrow__ , __leaf__));
1897.    extern void endusershell (void) __attribute__ ((__nothrow__ , __leaf__));
1898.    extern void setusershell (void) __attribute__ ((__nothrow__ , __leaf__));
1899.
1900.
1901.
1902.
1903.
1904.    extern int daemon (int __nochdir, int __noclose) __attribute__ ((__nothro
   w__ , __leaf__)) ;
1905.
1906.
1907.
1908.
1909.
1910.
1911.    extern int chroot (const char *__path) __attribute__ ((__nothrow__ , __le
   af__)) __attribute__ ((__nonnull__ (1))) ;
1912.
1913.
1914.
1915.    extern char *getpass (const char *__prompt) __attribute__ ((__nonnull__ (
   1)));
1916.
1917.
1918.
1919.
1920.
1921.
1922.
1923.    extern int fsync (int __fd);
1924.    # 967 "/usr/include/unistd.h" 3 4
1925.    extern long int gethostid (void);
1926.
1927.
1928.    extern void sync (void) __attribute__ ((__nothrow__ , __leaf__));
1929.
1930.
1931.
1932.
1933.
1934.    extern int getpagesize (void) __attribute__ ((__nothrow__ , __leaf__)) __
   attribute__ ((__const__));
1935.
1936.
```

```
1937.
1938.
1939.    extern int getdtablesize (void) __attribute__ ((__nothrow__ , __leaf__));
1940.    # 991 "/usr/include/unistd.h" 3 4
1941.    extern int truncate (const char *__file, __off_t __length)
1942.         __attribute__ ((__nothrow__ , __leaf__)) __attribute__ ((__nonnull__
     (1))) ;
1943.    # 1014 "/usr/include/unistd.h" 3 4
1944.    extern int ftruncate (int __fd, __off_t __length) __attribute__ ((__nothr
    ow__ , __leaf__)) ;
1945.    # 1035 "/usr/include/unistd.h" 3 4
1946.    extern int brk (void *__addr) __attribute__ ((__nothrow__ , __leaf__)) ;
1947.
1948.
1949.
1950.
1951.
1952.    extern void *sbrk (intptr_t __delta) __attribute__ ((__nothrow__ , __leaf
    __));
1953.    # 1056 "/usr/include/unistd.h" 3 4
1954.    extern long int syscall (long int __sysno, ...) __attribute__ ((__nothrow
    __ , __leaf__));
1955.    # 1079 "/usr/include/unistd.h" 3 4
1956.    extern int lockf (int __fd, int __cmd, __off_t __len) ;
1957.    # 1115 "/usr/include/unistd.h" 3 4
1958.    extern int fdatasync (int __fildes);
1959.    # 1124 "/usr/include/unistd.h" 3 4
1960.    extern char *crypt (const char *__key, const char *__salt)
1961.         __attribute__ ((__nothrow__ , __leaf__)) __attribute__ ((__nonnull__
     (1, 2)));
1962.    # 1161 "/usr/include/unistd.h" 3 4
1963.    int getentropy (void *__buffer, size_t __length) ;
1964.
1965.
1966.
1967.
1968.
1969.
1970.
1971.
1972.    # 8 "hello.c" 2
1973.    # 1 "/usr/include/stdlib.h" 1 3 4
1974.    # 25 "/usr/include/stdlib.h" 3 4
1975.    # 1 "/usr/include/x86_64-linux-gnu/bits/libc-header-start.h" 1 3 4
1976.    # 26 "/usr/include/stdlib.h" 2 3 4
1977.
1978.
```

```
1979.
1980.
1981.
1982.   # 1 "/usr/lib/gcc/x86_64-linux-gnu/8/include/stddef.h" 1 3 4
1983.   # 328 "/usr/lib/gcc/x86_64-linux-gnu/8/include/stddef.h" 3 4
1984.   typedef int wchar_t;
1985.   # 32 "/usr/include/stdlib.h" 2 3 4
1986.
1987.
1988.
1989.
1990.
1991.
1992.
1993.   # 1 "/usr/include/x86_64-linux-gnu/bits/waitflags.h" 1 3 4
1994.   # 52 "/usr/include/x86_64-linux-gnu/bits/waitflags.h" 3 4
1995.   typedef enum
1996.   {
1997.     P_ALL,
1998.     P_PID,
1999.     P_PGID
2000.   } idtype_t;
2001.   # 40 "/usr/include/stdlib.h" 2 3 4
2002.   # 1 "/usr/include/x86_64-linux-gnu/bits/waitstatus.h" 1 3 4
2003.   # 41 "/usr/include/stdlib.h" 2 3 4
2004.   # 55 "/usr/include/stdlib.h" 3 4
2005.   # 1 "/usr/include/x86_64-linux-gnu/bits/floatn.h" 1 3 4
2006.   # 120 "/usr/include/x86_64-linux-gnu/bits/floatn.h" 3 4
2007.   # 1 "/usr/include/x86_64-linux-gnu/bits/floatn-common.h" 1 3 4
2008.   # 24 "/usr/include/x86_64-linux-gnu/bits/floatn-common.h" 3 4
2009.   # 1 "/usr/include/x86_64-linux-gnu/bits/long-double.h" 1 3 4
2010.   # 25 "/usr/include/x86_64-linux-gnu/bits/floatn-common.h" 2 3 4
2011.   # 121 "/usr/include/x86_64-linux-gnu/bits/floatn.h" 2 3 4
2012.   # 56 "/usr/include/stdlib.h" 2 3 4
2013.
2014.
2015.   typedef struct
2016.     {
2017.       int quot;
2018.       int rem;
2019.     } div_t;
2020.
2021.
2022.
2023.   typedef struct
2024.     {
2025.       long int quot;
2026.       long int rem;
```

```
2027.       } ldiv_t;
2028.
2029.
2030.
2031.
2032.
2033.     __extension__ typedef struct
2034.       {
2035.         long long int quot;
2036.         long long int rem;
2037.       } lldiv_t;
2038.     # 97 "/usr/include/stdlib.h" 3 4
2039.     extern size_t __ctype_get_mb_cur_max (void) __attribute__ ((__nothrow__ ,
      __leaf__)) ;
2040.
2041.
2042.
2043.     extern double atof (const char *__nptr)
2044.           __attribute__ ((__nothrow__ , __leaf__)) __attribute__ ((__pure__))
     __attribute__ ((__nonnull__ (1))) ;
2045.
2046.     extern int atoi (const char *__nptr)
2047.           __attribute__ ((__nothrow__ , __leaf__)) __attribute__ ((__pure__))
     __attribute__ ((__nonnull__ (1))) ;
2048.
2049.     extern long int atol (const char *__nptr)
2050.           __attribute__ ((__nothrow__ , __leaf__)) __attribute__ ((__pure__))
     __attribute__ ((__nonnull__ (1))) ;
2051.
2052.
2053.
2054.     __extension__ extern long long int atoll (const char *__nptr)
2055.           __attribute__ ((__nothrow__ , __leaf__)) __attribute__ ((__pure__))
     __attribute__ ((__nonnull__ (1))) ;
2056.
2057.
2058.
2059.     extern double strtod (const char *__restrict __nptr,
2060.             char **__restrict __endptr)
2061.           __attribute__ ((__nothrow__ , __leaf__)) __attribute__ ((__nonnull__
      (1)));
2062.
2063.
2064.
2065.     extern float strtof (const char *__restrict __nptr,
2066.             char **__restrict __endptr) __attribute__ ((__nothrow__ , __leaf__)
     ) __attribute__ ((__nonnull__ (1)));
2067.
2068.     extern long double strtold (const char *__restrict __nptr,
```

```
2069.           char **__restrict __endptr)
2070.          __attribute__ ((__nothrow__ , __leaf__)) __attribute__ ((__nonnull__
    (1)));
2071.    # 176 "/usr/include/stdlib.h" 3 4
2072.    extern long int strtol (const char *__restrict __nptr,
2073.       char **__restrict __endptr, int __base)
2074.          __attribute__ ((__nothrow__ , __leaf__)) __attribute__ ((__nonnull__
    (1)));
2075.
2076.    extern unsigned long int strtoul (const char *__restrict __nptr,
2077.           char **__restrict __endptr, int __base)
2078.          __attribute__ ((__nothrow__ , __leaf__)) __attribute__ ((__nonnull__
    (1)));
2079.
2080.
2081.
2082.    __extension__
2083.    extern long long int strtoq (const char *__restrict __nptr,
2084.            char **__restrict __endptr, int __base)
2085.          __attribute__ ((__nothrow__ , __leaf__)) __attribute__ ((__nonnull__
    (1)));
2086.
2087.    __extension__
2088.    extern unsigned long long int strtouq (const char *__restrict __nptr,
2089.                char **__restrict __endptr, int __base)
2090.          __attribute__ ((__nothrow__ , __leaf__)) __attribute__ ((__nonnull__
    (1)));
2091.
2092.
2093.
2094.
2095.    __extension__
2096.    extern long long int strtoll (const char *__restrict __nptr,
2097.             char **__restrict __endptr, int __base)
2098.          __attribute__ ((__nothrow__ , __leaf__)) __attribute__ ((__nonnull__
    (1)));
2099.
2100.    __extension__
2101.    extern unsigned long long int strtoull (const char *__restrict __nptr,
2102.         char **__restrict __endptr, int __base)
2103.          __attribute__ ((__nothrow__ , __leaf__)) __attribute__ ((__nonnull__
    (1)));
2104.    # 385 "/usr/include/stdlib.h" 3 4
2105.    extern char *l64a (long int __n) __attribute__ ((__nothrow__ , __leaf__))
    ;
2106.
2107.
2108.    extern long int a64l (const char *__s)
2109.          __attribute__ ((__nothrow__ , __leaf__)) __attribute__ ((__pure__))
   __attribute__ ((__nonnull__ (1))) ;
```

```
2110.
2111.
2112.
2113.
2114.    # 1 "/usr/include/x86_64-linux-gnu/sys/types.h" 1 3 4
2115.    # 27 "/usr/include/x86_64-linux-gnu/sys/types.h" 3 4
2116.
2117.
2118.
2119.
2120.
2121.
2122.    typedef __u_char u_char;
2123.    typedef __u_short u_short;
2124.    typedef __u_int u_int;
2125.    typedef __u_long u_long;
2126.    typedef __quad_t quad_t;
2127.    typedef __u_quad_t u_quad_t;
2128.    typedef __fsid_t fsid_t;
2129.
2130.
2131.    typedef __loff_t loff_t;
2132.
2133.
2134.
2135.
2136.    typedef __ino_t ino_t;
2137.    # 59 "/usr/include/x86_64-linux-gnu/sys/types.h" 3 4
2138.    typedef __dev_t dev_t;
2139.    # 69 "/usr/include/x86_64-linux-gnu/sys/types.h" 3 4
2140.    typedef __mode_t mode_t;
2141.
2142.
2143.
2144.
2145.    typedef __nlink_t nlink_t;
2146.    # 103 "/usr/include/x86_64-linux-gnu/sys/types.h" 3 4
2147.    typedef __id_t id_t;
2148.    # 114 "/usr/include/x86_64-linux-gnu/sys/types.h" 3 4
2149.    typedef __daddr_t daddr_t;
2150.    typedef __caddr_t caddr_t;
2151.
2152.
2153.
2154.
2155.
2156.    typedef __key_t key_t;
2157.
```

```
2158.
2159.
2160.
2161.   # 1 "/usr/include/x86_64-linux-gnu/bits/types/clock_t.h" 1 3 4
2162.
2163.
2164.
2165.
2166.
2167.
2168.   typedef __clock_t clock_t;
2169.   # 127 "/usr/include/x86_64-linux-gnu/sys/types.h" 2 3 4
2170.
2171.   # 1 "/usr/include/x86_64-linux-gnu/bits/types/clockid_t.h" 1 3 4
2172.
2173.
2174.
2175.
2176.
2177.
2178.   typedef __clockid_t clockid_t;
2179.   # 129 "/usr/include/x86_64-linux-gnu/sys/types.h" 2 3 4
2180.   # 1 "/usr/include/x86_64-linux-gnu/bits/types/time_t.h" 1 3 4
2181.
2182.
2183.
2184.
2185.
2186.
2187.   typedef __time_t time_t;
2188.   # 130 "/usr/include/x86_64-linux-gnu/sys/types.h" 2 3 4
2189.   # 1 "/usr/include/x86_64-linux-gnu/bits/types/timer_t.h" 1 3 4
2190.
2191.
2192.
2193.
2194.
2195.
2196.   typedef __timer_t timer_t;
2197.   # 131 "/usr/include/x86_64-linux-gnu/sys/types.h" 2 3 4
2198.   # 144 "/usr/include/x86_64-linux-gnu/sys/types.h" 3 4
2199.   # 1 "/usr/lib/gcc/x86_64-linux-gnu/8/include/stddef.h" 1 3 4
2200.   # 145 "/usr/include/x86_64-linux-gnu/sys/types.h" 2 3 4
2201.
2202.
2203.
2204.   typedef unsigned long int ulong;
2205.   typedef unsigned short int ushort;
```

```
2206.   typedef unsigned int uint;
2207.
2208.
2209.
2210.
2211.   # 1 "/usr/include/x86_64-linux-gnu/bits/stdint-intn.h" 1 3 4
2212.   # 24 "/usr/include/x86_64-linux-gnu/bits/stdint-intn.h" 3 4
2213.   typedef __int8_t int8_t;
2214.   typedef __int16_t int16_t;
2215.   typedef __int32_t int32_t;
2216.   typedef __int64_t int64_t;
2217.   # 156 "/usr/include/x86_64-linux-gnu/sys/types.h" 2 3 4
2218.   # 177 "/usr/include/x86_64-linux-gnu/sys/types.h" 3 4
2219.   typedef unsigned int u_int8_t __attribute__ ((__mode__ (__QI__)));
2220.   typedef unsigned int u_int16_t __attribute__ ((__mode__ (__HI__)));
2221.   typedef unsigned int u_int32_t __attribute__ ((__mode__ (__SI__)));
2222.   typedef unsigned int u_int64_t __attribute__ ((__mode__ (__DI__)));
2223.
2224.   typedef int register_t __attribute__ ((__mode__ (__word__)));
2225.   # 193 "/usr/include/x86_64-linux-gnu/sys/types.h" 3 4
2226.   # 1 "/usr/include/endian.h" 1 3 4
2227.   # 36 "/usr/include/endian.h" 3 4
2228.   # 1 "/usr/include/x86_64-linux-gnu/bits/endian.h" 1 3 4
2229.   # 37 "/usr/include/endian.h" 2 3 4
2230.   # 60 "/usr/include/endian.h" 3 4
2231.   # 1 "/usr/include/x86_64-linux-gnu/bits/byteswap.h" 1 3 4
2232.   # 33 "/usr/include/x86_64-linux-gnu/bits/byteswap.h" 3 4
2233.   static __inline __uint16_t
2234.   __bswap_16 (__uint16_t __bsx)
2235.   {
2236.
2237.     return __builtin_bswap16 (__bsx);
2238.
2239.
2240.
2241.   }
2242.
2243.
2244.
2245.
2246.
2247.
2248.   static __inline __uint32_t
2249.   __bswap_32 (__uint32_t __bsx)
2250.   {
2251.
2252.     return __builtin_bswap32 (__bsx);
2253.
```

```
2254.
2255.
2256.   }
2257.   # 69 "/usr/include/x86_64-linux-gnu/bits/byteswap.h" 3 4
2258.   __extension__ static __inline __uint64_t
2259.   __bswap_64 (__uint64_t __bsx)
2260.   {
2261.
2262.     return __builtin_bswap64 (__bsx);
2263.
2264.
2265.
2266.   }
2267.   # 61 "/usr/include/endian.h" 2 3 4
2268.   # 1 "/usr/include/x86_64-linux-gnu/bits/uintn-identity.h" 1 3 4
2269.   # 32 "/usr/include/x86_64-linux-gnu/bits/uintn-identity.h" 3 4
2270.   static __inline __uint16_t
2271.   __uint16_identity (__uint16_t __x)
2272.   {
2273.     return __x;
2274.   }
2275.
2276.   static __inline __uint32_t
2277.   __uint32_identity (__uint32_t __x)
2278.   {
2279.     return __x;
2280.   }
2281.
2282.   static __inline __uint64_t
2283.   __uint64_identity (__uint64_t __x)
2284.   {
2285.     return __x;
2286.   }
2287.   # 62 "/usr/include/endian.h" 2 3 4
2288.   # 194 "/usr/include/x86_64-linux-gnu/sys/types.h" 2 3 4
2289.
2290.
2291.   # 1 "/usr/include/x86_64-linux-gnu/sys/select.h" 1 3 4
2292.   # 30 "/usr/include/x86_64-linux-gnu/sys/select.h" 3 4
2293.   # 1 "/usr/include/x86_64-linux-gnu/bits/select.h" 1 3 4
2294.   # 22 "/usr/include/x86_64-linux-gnu/bits/select.h" 3 4
2295.   # 1 "/usr/include/x86_64-linux-gnu/bits/wordsize.h" 1 3 4
2296.   # 23 "/usr/include/x86_64-linux-gnu/bits/select.h" 2 3 4
2297.   # 31 "/usr/include/x86_64-linux-gnu/sys/select.h" 2 3 4
2298.
2299.
2300.   # 1 "/usr/include/x86_64-linux-gnu/bits/types/sigset_t.h" 1 3 4
2301.
```

```
2302.
2303.
2304.   # 1 "/usr/include/x86_64-linux-gnu/bits/types/__sigset_t.h" 1 3 4
2305.
2306.
2307.
2308.
2309.   typedef struct
2310.   {
2311.     unsigned long int __val[(1024 / (8 * sizeof (unsigned long int)))];
2312.   } __sigset_t;
2313.   # 5 "/usr/include/x86_64-linux-gnu/bits/types/sigset_t.h" 2 3 4
2314.
2315.
2316.   typedef __sigset_t sigset_t;
2317.   # 34 "/usr/include/x86_64-linux-gnu/sys/select.h" 2 3 4
2318.
2319.
2320.
2321.   # 1 "/usr/include/x86_64-linux-gnu/bits/types/struct_timeval.h" 1 3 4
2322.
2323.
2324.
2325.
2326.
2327.
2328.
2329.   struct timeval
2330.   {
2331.     __time_t tv_sec;
2332.     __suseconds_t tv_usec;
2333.   };
2334.   # 38 "/usr/include/x86_64-linux-gnu/sys/select.h" 2 3 4
2335.
2336.   # 1 "/usr/include/x86_64-linux-gnu/bits/types/struct_timespec.h" 1 3 4
2337.   # 9 "/usr/include/x86_64-linux-gnu/bits/types/struct_timespec.h" 3 4
2338.   struct timespec
2339.   {
2340.     __time_t tv_sec;
2341.     __syscall_slong_t tv_nsec;
2342.   };
2343.   # 40 "/usr/include/x86_64-linux-gnu/sys/select.h" 2 3 4
2344.
2345.
2346.
2347.   typedef __suseconds_t suseconds_t;
2348.
2349.
```

```
2350.
2351.
2352.
2353.   typedef long int __fd_mask;
2354.   # 59 "/usr/include/x86_64-linux-gnu/sys/select.h" 3 4
2355.   typedef struct
2356.     {
2357.
2358.
2359.
2360.
2361.
2362.
2363.       __fd_mask __fds_bits[1024 / (8 * (int) sizeof (__fd_mask))];
2364.
2365.
2366.     } fd_set;
2367.
2368.
2369.
2370.
2371.
2372.
2373.   typedef __fd_mask fd_mask;
2374.   # 91 "/usr/include/x86_64-linux-gnu/sys/select.h" 3 4
2375.
2376.   # 101 "/usr/include/x86_64-linux-gnu/sys/select.h" 3 4
2377.   extern int select (int __nfds, fd_set *__restrict __readfds,
2378.        fd_set *__restrict __writefds,
2379.        fd_set *__restrict __exceptfds,
2380.        struct timeval *__restrict __timeout);
2381.   # 113 "/usr/include/x86_64-linux-gnu/sys/select.h" 3 4
2382.   extern int pselect (int __nfds, fd_set *__restrict __readfds,
2383.         fd_set *__restrict __writefds,
2384.         fd_set *__restrict __exceptfds,
2385.         const struct timespec *__restrict __timeout,
2386.         const __sigset_t *__restrict __sigmask);
2387.   # 126 "/usr/include/x86_64-linux-gnu/sys/select.h" 3 4
2388.
2389.   # 197 "/usr/include/x86_64-linux-gnu/sys/types.h" 2 3 4
2390.
2391.
2392.
2393.
2394.
2395.   typedef __blksize_t blksize_t;
2396.
2397.
```

```
2398.
2399.
2400.
2401.
2402.   typedef __blkcnt_t blkcnt_t;
2403.
2404.
2405.
2406.   typedef __fsblkcnt_t fsblkcnt_t;
2407.
2408.
2409.
2410.   typedef __fsfilcnt_t fsfilcnt_t;
2411.   # 244 "/usr/include/x86_64-linux-gnu/sys/types.h" 3 4
2412.   # 1 "/usr/include/x86_64-linux-gnu/bits/pthreadtypes.h" 1 3 4
2413.   # 23 "/usr/include/x86_64-linux-gnu/bits/pthreadtypes.h" 3 4
2414.   # 1 "/usr/include/x86_64-linux-gnu/bits/thread-shared-types.h" 1 3 4
2415.   # 77 "/usr/include/x86_64-linux-gnu/bits/thread-shared-types.h" 3 4
2416.   # 1 "/usr/include/x86_64-linux-gnu/bits/pthreadtypes-arch.h" 1 3 4
2417.   # 21 "/usr/include/x86_64-linux-gnu/bits/pthreadtypes-arch.h" 3 4
2418.   # 1 "/usr/include/x86_64-linux-gnu/bits/wordsize.h" 1 3 4
2419.   # 22 "/usr/include/x86_64-linux-gnu/bits/pthreadtypes-arch.h" 2 3 4
2420.   # 65 "/usr/include/x86_64-linux-gnu/bits/pthreadtypes-arch.h" 3 4
2421.   struct __pthread_rwlock_arch_t
2422.   {
2423.     unsigned int __readers;
2424.     unsigned int __writers;
2425.     unsigned int __wrphase_futex;
2426.     unsigned int __writers_futex;
2427.     unsigned int __pad3;
2428.     unsigned int __pad4;
2429.
2430.     int __cur_writer;
2431.     int __shared;
2432.     signed char __rwelision;
2433.
2434.
2435.
2436.
2437.     unsigned char __pad1[7];
2438.
2439.
2440.     unsigned long int __pad2;
2441.
2442.
2443.     unsigned int __flags;
2444.   # 99 "/usr/include/x86_64-linux-gnu/bits/pthreadtypes-arch.h" 3 4
2445.   };
```

```
2446.   # 78 "/usr/include/x86_64-linux-gnu/bits/thread-shared-types.h" 2 3 4
2447.
2448.
2449.
2450.
2451.   typedef struct __pthread_internal_list
2452.   {
2453.     struct __pthread_internal_list *__prev;
2454.     struct __pthread_internal_list *__next;
2455.   } __pthread_list_t;
2456.   # 118 "/usr/include/x86_64-linux-gnu/bits/thread-shared-types.h" 3 4
2457.   struct __pthread_mutex_s
2458.   {
2459.     int __lock ;
2460.     unsigned int __count;
2461.     int __owner;
2462.
2463.     unsigned int __nusers;
2464.   # 148 "/usr/include/x86_64-linux-gnu/bits/thread-shared-types.h" 3 4
2465.     int __kind;
2466.
2467.
2468.
2469.
2470.
2471.     short __spins; short __elision;
2472.     __pthread_list_t __list;
2473.   # 165 "/usr/include/x86_64-linux-gnu/bits/thread-shared-types.h" 3 4
2474.
2475.   };
2476.
2477.
2478.
2479.
2480.   struct __pthread_cond_s
2481.   {
2482.     __extension__ union
2483.     {
2484.       __extension__ unsigned long long int __wseq;
2485.       struct
2486.       {
2487.         unsigned int __low;
2488.         unsigned int __high;
2489.       } __wseq32;
2490.     };
2491.     __extension__ union
2492.     {
2493.       __extension__ unsigned long long int __g1_start;
```

```
2494.       struct
2495.       {
2496.         unsigned int __low;
2497.         unsigned int __high;
2498.       } __g1_start32;
2499.     };
2500.     unsigned int __g_refs[2] ;
2501.     unsigned int __g_size[2];
2502.     unsigned int __g1_orig_size;
2503.     unsigned int __wrefs;
2504.     unsigned int __g_signals[2];
2505.   };
2506.   # 24 "/usr/include/x86_64-linux-gnu/bits/pthreadtypes.h" 2 3 4
2507.
2508.
2509.
2510.   typedef unsigned long int pthread_t;
2511.
2512.
2513.
2514.
2515.   typedef union
2516.   {
2517.     char __size[4];
2518.     int __align;
2519.   } pthread_mutexattr_t;
2520.
2521.
2522.
2523.
2524.   typedef union
2525.   {
2526.     char __size[4];
2527.     int __align;
2528.   } pthread_condattr_t;
2529.
2530.
2531.
2532.   typedef unsigned int pthread_key_t;
2533.
2534.
2535.
2536.   typedef int pthread_once_t;
2537.
2538.
2539.   union pthread_attr_t
2540.   {
2541.     char __size[56];
```

```
2542.     long int __align;
2543.   };
2544.
2545.   typedef union pthread_attr_t pthread_attr_t;
2546.
2547.
2548.
2549.
2550.   typedef union
2551.   {
2552.     struct __pthread_mutex_s __data;
2553.     char __size[40];
2554.     long int __align;
2555.   } pthread_mutex_t;
2556.
2557.
2558.   typedef union
2559.   {
2560.     struct __pthread_cond_s __data;
2561.     char __size[48];
2562.     __extension__ long long int __align;
2563.   } pthread_cond_t;
2564.
2565.
2566.
2567.
2568.
2569.   typedef union
2570.   {
2571.     struct __pthread_rwlock_arch_t __data;
2572.     char __size[56];
2573.     long int __align;
2574.   } pthread_rwlock_t;
2575.
2576.   typedef union
2577.   {
2578.     char __size[8];
2579.     long int __align;
2580.   } pthread_rwlockattr_t;
2581.
2582.
2583.
2584.
2585.
2586.   typedef volatile int pthread_spinlock_t;
2587.
2588.
2589.
```

```
2590.
2591.    typedef union
2592.    {
2593.      char __size[32];
2594.      long int __align;
2595.    } pthread_barrier_t;
2596.
2597.    typedef union
2598.    {
2599.      char __size[4];
2600.      int __align;
2601.    } pthread_barrierattr_t;
2602.    # 245 "/usr/include/x86_64-linux-gnu/sys/types.h" 2 3 4
2603.
2604.
2605.
2606.    # 395 "/usr/include/stdlib.h" 2 3 4
2607.
2608.
2609.
2610.
2611.
2612.
2613.    extern long int random (void) __attribute__ ((__nothrow__ , __leaf__));
2614.
2615.
2616.    extern void srandom (unsigned int __seed) __attribute__ ((__nothrow__ , _
     _leaf__));
2617.
2618.
2619.
2620.
2621.
2622.    extern char *initstate (unsigned int __seed, char *__statebuf,
2623.        size_t __statelen) __attribute__ ((__nothrow__ , __leaf__)) __attribut
     e__ ((__nonnull__ (2)));
2624.
2625.
2626.
2627.    extern char *setstate (char *__statebuf) __attribute__ ((__nothrow__ , __
     leaf__)) __attribute__ ((__nonnull__ (1)));
2628.
2629.
2630.
2631.
2632.
2633.
2634.
2635.    struct random_data
```

```
2636.       {
2637.         int32_t *fptr;
2638.         int32_t *rptr;
2639.         int32_t *state;
2640.         int rand_type;
2641.         int rand_deg;
2642.         int rand_sep;
2643.         int32_t *end_ptr;
2644.       };
2645.
2646.     extern int random_r (struct random_data *__restrict __buf,
2647.                          int32_t *__restrict __result) __attribute__ ((__nothrow__ , __leaf
    __)) __attribute__ ((__nonnull__ (1, 2)));
2648.
2649.     extern int srandom_r (unsigned int __seed, struct random_data *__buf)
2650.          __attribute__ ((__nothrow__ , __leaf__)) __attribute__ ((__nonnull__
     (2)));
2651.
2652.     extern int initstate_r (unsigned int __seed, char *__restrict __statebuf,

2653.        size_t __statelen,
2654.        struct random_data *__restrict __buf)
2655.          __attribute__ ((__nothrow__ , __leaf__)) __attribute__ ((__nonnull__
     (2, 4)));
2656.
2657.     extern int setstate_r (char *__restrict __statebuf,
2658.              struct random_data *__restrict __buf)
2659.          __attribute__ ((__nothrow__ , __leaf__)) __attribute__ ((__nonnull__
     (1, 2)));
2660.
2661.
2662.
2663.
2664.
2665.     extern int rand (void) __attribute__ ((__nothrow__ , __leaf__));
2666.
2667.     extern void srand (unsigned int __seed) __attribute__ ((__nothrow__ , __l
    eaf__));
2668.
2669.
2670.
2671.     extern int rand_r (unsigned int *__seed) __attribute__ ((__nothrow__ , __
    leaf__));
2672.
2673.
2674.
2675.
2676.
2677.
```

```
2678.
2679.    extern double drand48 (void) __attribute__ ((__nothrow__ , __leaf__));
2680.    extern double erand48 (unsigned short int __xsubi[3]) __attribute__ ((__n
    othrow__ , __leaf__)) __attribute__ ((__nonnull__ (1)));
2681.
2682.
2683.    extern long int lrand48 (void) __attribute__ ((__nothrow__ , __leaf__));

2684.    extern long int nrand48 (unsigned short int __xsubi[3])
2685.          __attribute__ ((__nothrow__ , __leaf__)) __attribute__ ((__nonnull__
     (1)));
2686.
2687.
2688.    extern long int mrand48 (void) __attribute__ ((__nothrow__ , __leaf__));

2689.    extern long int jrand48 (unsigned short int __xsubi[3])
2690.          __attribute__ ((__nothrow__ , __leaf__)) __attribute__ ((__nonnull__
     (1)));
2691.
2692.
2693.    extern void srand48 (long int __seedval) __attribute__ ((__nothrow__ , __
    leaf__));
2694.    extern unsigned short int *seed48 (unsigned short int __seed16v[3])
2695.          __attribute__ ((__nothrow__ , __leaf__)) __attribute__ ((__nonnull__
     (1)));
2696.    extern void lcong48 (unsigned short int __param[7]) __attribute__ ((__not
    hrow__ , __leaf__)) __attribute__ ((__nonnull__ (1)));
2697.
2698.
2699.
2700.
2701.
2702.    struct drand48_data
2703.      {
2704.        unsigned short int __x[3];
2705.        unsigned short int __old_x[3];
2706.        unsigned short int __c;
2707.        unsigned short int __init;
2708.        __extension__ unsigned long long int __a;
2709.
2710.      };
2711.
2712.
2713.    extern int drand48_r (struct drand48_data *__restrict __buffer,
2714.             double *__restrict __result) __attribute__ ((__nothrow__ , __leaf
    __)) __attribute__ ((__nonnull__ (1, 2)));
2715.    extern int erand48_r (unsigned short int __xsubi[3],
2716.            struct drand48_data *__restrict __buffer,
2717.             double *__restrict __result) __attribute__ ((__nothrow__ , __leaf
    __)) __attribute__ ((__nonnull__ (1, 2)));
```

```
2718.
2719.
2720.    extern int lrand48_r (struct drand48_data *__restrict __buffer,
2721.            long int *__restrict __result)
2722.          __attribute__ ((__nothrow__ , __leaf__)) __attribute__ ((__nonnull__
     (1, 2)));
2723.    extern int nrand48_r (unsigned short int __xsubi[3],
2724.            struct drand48_data *__restrict __buffer,
2725.            long int *__restrict __result)
2726.          __attribute__ ((__nothrow__ , __leaf__)) __attribute__ ((__nonnull__
     (1, 2)));
2727.
2728.
2729.    extern int mrand48_r (struct drand48_data *__restrict __buffer,
2730.            long int *__restrict __result)
2731.          __attribute__ ((__nothrow__ , __leaf__)) __attribute__ ((__nonnull__
     (1, 2)));
2732.    extern int jrand48_r (unsigned short int __xsubi[3],
2733.            struct drand48_data *__restrict __buffer,
2734.            long int *__restrict __result)
2735.          __attribute__ ((__nothrow__ , __leaf__)) __attribute__ ((__nonnull__
     (1, 2)));
2736.
2737.
2738.    extern int srand48_r (long int __seedval, struct drand48_data *__buffer)

2739.          __attribute__ ((__nothrow__ , __leaf__)) __attribute__ ((__nonnull__
     (2)));
2740.
2741.    extern int seed48_r (unsigned short int __seed16v[3],
2742.            struct drand48_data *__buffer) __attribute__ ((__nothrow__ , __lea
    f__)) __attribute__ ((__nonnull__ (1, 2)));
2743.
2744.    extern int lcong48_r (unsigned short int __param[7],
2745.            struct drand48_data *__buffer)
2746.          __attribute__ ((__nothrow__ , __leaf__)) __attribute__ ((__nonnull__
     (1, 2)));
2747.
2748.
2749.
2750.
2751.    extern void *malloc (size_t __size) __attribute__ ((__nothrow__ , __leaf_
    _)) __attribute__ ((__malloc__)) ;
2752.
2753.    extern void *calloc (size_t __nmemb, size_t __size)
2754.          __attribute__ ((__nothrow__ , __leaf__)) __attribute__ ((__malloc__))
     ;
2755.
2756.
2757.
```

```
2758.
2759.
2760.
2761.    extern void *realloc (void *__ptr, size_t __size)
2762.          __attribute__ ((__nothrow__ , __leaf__)) __attribute__ ((__warn_unus
    ed_result__));
2763.
2764.
2765.
2766.
2767.
2768.
2769.
2770.    extern void *reallocarray (void *__ptr, size_t __nmemb, size_t __size)
2771.          __attribute__ ((__nothrow__ , __leaf__)) __attribute__ ((__warn_unus
    ed_result__));
2772.
2773.
2774.
2775.    extern void free (void *__ptr) __attribute__ ((__nothrow__ , __leaf__));

2776.
2777.
2778.    # 1 "/usr/include/alloca.h" 1 3 4
2779.    # 24 "/usr/include/alloca.h" 3 4
2780.    # 1 "/usr/lib/gcc/x86_64-linux-gnu/8/include/stddef.h" 1 3 4
2781.    # 25 "/usr/include/alloca.h" 2 3 4
2782.
2783.
2784.
2785.
2786.
2787.
2788.
2789.    extern void *alloca (size_t __size) __attribute__ ((__nothrow__ , __leaf_
    _));
2790.
2791.
2792.
2793.
2794.
2795.
2796.    # 567 "/usr/include/stdlib.h" 2 3 4
2797.
2798.
2799.
2800.
2801.
```

```
2802.    extern void *valloc (size_t __size) __attribute__ ((__nothrow__ , __leaf_
    _)) __attribute__ ((__malloc__)) ;
2803.
2804.
2805.
2806.
2807.    extern int posix_memalign (void **__memptr, size_t __alignment, size_t __
    size)
2808.          __attribute__ ((__nothrow__ , __leaf__)) __attribute__ ((__nonnull__
     (1))) ;
2809.
2810.
2811.
2812.
2813.    extern void *aligned_alloc (size_t __alignment, size_t __size)
2814.          __attribute__ ((__nothrow__ , __leaf__)) __attribute__ ((__malloc__))
     __attribute__ ((__alloc_size__ (2))) ;
2815.
2816.
2817.
2818.    extern void abort (void) __attribute__ ((__nothrow__ , __leaf__)) __attri
    bute__ ((__noreturn__));
2819.
2820.
2821.
2822.    extern int atexit (void (*__func) (void)) __attribute__ ((__nothrow__ , _
    _leaf__)) __attribute__ ((__nonnull__ (1)));
2823.
2824.
2825.
2826.
2827.
2828.
2829.
2830.    extern int at_quick_exit (void (*__func) (void)) __attribute__ ((__nothro
    w__ , __leaf__)) __attribute__ ((__nonnull__ (1)));
2831.
2832.
2833.
2834.
2835.
2836.
2837.    extern int on_exit (void (*__func) (int __status, void *__arg), void *__a
    rg)
2838.          __attribute__ ((__nothrow__ , __leaf__)) __attribute__ ((__nonnull__
     (1)));
2839.
2840.
2841.
2842.
```

```
2843.
2844.    extern void exit (int __status) __attribute__ ((__nothrow__ , __leaf__))
    __attribute__ ((__noreturn__));
2845.
2846.
2847.
2848.
2849.
2850.    extern void quick_exit (int __status) __attribute__ ((__nothrow__ , __lea
    f__)) __attribute__ ((__noreturn__));
2851.
2852.
2853.
2854.
2855.
2856.    extern void _Exit (int __status) __attribute__ ((__nothrow__ , __leaf__))
     __attribute__ ((__noreturn__));
2857.
2858.
2859.
2860.
2861.    extern char *getenv (const char *__name) __attribute__ ((__nothrow__ , __
    leaf__)) __attribute__ ((__nonnull__ (1))) ;
2862.    # 644 "/usr/include/stdlib.h" 3 4
2863.    extern int putenv (char *__string) __attribute__ ((__nothrow__ , __leaf__)
    ) __attribute__ ((__nonnull__ (1)));
2864.
2865.
2866.
2867.
2868.
2869.    extern int setenv (const char *__name, const char *__value, int __replace)

2870.            __attribute__ ((__nothrow__ , __leaf__)) __attribute__ ((__nonnull__
     (2)));
2871.
2872.
2873.    extern int unsetenv (const char *__name) __attribute__ ((__nothrow__ , __
    leaf__)) __attribute__ ((__nonnull__ (1)));
2874.
2875.
2876.
2877.
2878.
2879.
2880.    extern int clearenv (void) __attribute__ ((__nothrow__ , __leaf__));
2881.    # 672 "/usr/include/stdlib.h" 3 4
2882.    extern char *mktemp (char *__template) __attribute__ ((__nothrow__ , __le
    af__)) __attribute__ ((__nonnull__ (1)));
2883.    # 685 "/usr/include/stdlib.h" 3 4
```

```
2884.     extern int mkstemp (char *__template) __attribute__ ((__nonnull__ (1))) ;
2885.     # 707 "/usr/include/stdlib.h" 3 4
2886.     extern int mkstemps (char *__template, int __suffixlen) __attribute__ ((_
    _nonnull__ (1))) ;
2887.     # 728 "/usr/include/stdlib.h" 3 4
2888.     extern char *mkdtemp (char *__template) __attribute__ ((__nothrow__ , __l
    eaf__)) __attribute__ ((__nonnull__ (1))) ;
2889.     # 781 "/usr/include/stdlib.h" 3 4
2890.     extern int system (const char *__command) ;
2891.     # 797 "/usr/include/stdlib.h" 3 4
2892.     extern char *realpath (const char *__restrict __name,
2893.               char *__restrict __resolved) __attribute__ ((__nothrow__ , __lea
    f__)) ;
2894.
2895.
2896.
2897.
2898.
2899.
2900.     typedef int (*__compar_fn_t) (const void *, const void *);
2901.     # 817 "/usr/include/stdlib.h" 3 4
2902.     extern void *bsearch (const void *__key, const void *__base,
2903.             size_t __nmemb, size_t __size, __compar_fn_t __compar)
2904.          __attribute__ ((__nonnull__ (1, 2, 5))) ;
2905.
2906.
2907.
2908.
2909.
2910.
2911.
2912.     extern void qsort (void *__base, size_t __nmemb, size_t __size,
2913.          __compar_fn_t __compar) __attribute__ ((__nonnull__ (1, 4)));
2914.     # 837 "/usr/include/stdlib.h" 3 4
2915.     extern int abs (int __x) __attribute__ ((__nothrow__ , __leaf__)) __attri
    bute__ ((__const__)) ;
2916.     extern long int labs (long int __x) __attribute__ ((__nothrow__ , __leaf_
    _)) __attribute__ ((__const__)) ;
2917.
2918.
2919.     __extension__ extern long long int llabs (long long int __x)
2920.          __attribute__ ((__nothrow__ , __leaf__)) __attribute__ ((__const__))
     ;
2921.
2922.
2923.
2924.
2925.
```

```
2926.
2927.    extern div_t div (int __numer, int __denom)
2928.          __attribute__ ((__nothrow__ , __leaf__)) __attribute__ ((__const__))
     ;
2929.    extern ldiv_t ldiv (long int __numer, long int __denom)
2930.          __attribute__ ((__nothrow__ , __leaf__)) __attribute__ ((__const__))
     ;
2931.
2932.
2933.    __extension__ extern lldiv_t lldiv (long long int __numer,
2934.            long long int __denom)
2935.          __attribute__ ((__nothrow__ , __leaf__)) __attribute__ ((__const__))
     ;
2936.    # 869 "/usr/include/stdlib.h" 3 4
2937.    extern char *ecvt (double __value, int __ndigit, int *__restrict __decpt,

2938.          int *__restrict __sign) __attribute__ ((__nothrow__ , __leaf__)) __a
    ttribute__ ((__nonnull__ (3, 4))) ;
2939.
2940.
2941.
2942.
2943.    extern char *fcvt (double __value, int __ndigit, int *__restrict __decpt,

2944.          int *__restrict __sign) __attribute__ ((__nothrow__ , __leaf__)) __a
    ttribute__ ((__nonnull__ (3, 4))) ;
2945.
2946.
2947.
2948.
2949.    extern char *gcvt (double __value, int __ndigit, char *__buf)
2950.          __attribute__ ((__nothrow__ , __leaf__)) __attribute__ ((__nonnull__
     (3))) ;
2951.
2952.
2953.
2954.
2955.    extern char *qecvt (long double __value, int __ndigit,
2956.           int *__restrict __decpt, int *__restrict __sign)
2957.          __attribute__ ((__nothrow__ , __leaf__)) __attribute__ ((__nonnull__
     (3, 4))) ;
2958.    extern char *qfcvt (long double __value, int __ndigit,
2959.           int *__restrict __decpt, int *__restrict __sign)
2960.          __attribute__ ((__nothrow__ , __leaf__)) __attribute__ ((__nonnull__
     (3, 4))) ;
2961.    extern char *qgcvt (long double __value, int __ndigit, char *__buf)
2962.          __attribute__ ((__nothrow__ , __leaf__)) __attribute__ ((__nonnull__
     (3))) ;
2963.
2964.
```

```
2965.
2966.
2967.    extern int ecvt_r (double __value, int __ndigit, int *__restrict __decpt,
2968.           int *__restrict __sign, char *__restrict __buf,
2969.           size_t __len) __attribute__ ((__nothrow__ , __leaf__)) __attribute__
     ((__nonnull__ (3, 4, 5)));
2970.    extern int fcvt_r (double __value, int __ndigit, int *__restrict __decpt,
2971.           int *__restrict __sign, char *__restrict __buf,
2972.           size_t __len) __attribute__ ((__nothrow__ , __leaf__)) __attribute__
     ((__nonnull__ (3, 4, 5)));
2973.
2974.    extern int qecvt_r (long double __value, int __ndigit,
2975.           int *__restrict __decpt, int *__restrict __sign,
2976.           char *__restrict __buf, size_t __len)
2977.           __attribute__ ((__nothrow__ , __leaf__)) __attribute__ ((__nonnull__
     (3, 4, 5)));
2978.    extern int qfcvt_r (long double __value, int __ndigit,
2979.           int *__restrict __decpt, int *__restrict __sign,
2980.           char *__restrict __buf, size_t __len)
2981.           __attribute__ ((__nothrow__ , __leaf__)) __attribute__ ((__nonnull__
     (3, 4, 5)));
2982.
2983.
2984.
2985.
2986.
2987.    extern int mblen (const char *__s, size_t __n) __attribute__ ((__nothrow_
     _ , __leaf__));
2988.
2989.
2990.    extern int mbtowc (wchar_t *__restrict __pwc,
2991.           const char *__restrict __s, size_t __n) __attribute__ ((__nothrow__ ,
     __leaf__));
2992.
2993.
2994.    extern int wctomb (char *__s, wchar_t __wchar) __attribute__ ((__nothrow_
     _ , __leaf__));
2995.
2996.
2997.
2998.    extern size_t mbstowcs (wchar_t *__restrict __pwcs,
2999.           const char *__restrict __s, size_t __n) __attribute__ ((__nothrow__ ,
     __leaf__));
3000.
3001.    extern size_t wcstombs (char *__restrict __s,
3002.           const wchar_t *__restrict __pwcs, size_t __n)
3003.           __attribute__ ((__nothrow__ , __leaf__));
3004.
```

```
3005.
3006.
3007.
3008.
3009.
3010.
3011.    extern int rpmatch (const char *__response) __attribute__ ((__nothrow__ ,
     __leaf__)) __attribute__ ((__nonnull__ (1))) ;
3012.    # 954 "/usr/include/stdlib.h" 3 4
3013.    extern int getsubopt (char **__restrict __optionp,
3014.             char *const *__restrict __tokens,
3015.             char **__restrict __valuep)
3016.          __attribute__ ((__nothrow__ , __leaf__)) __attribute__ ((__nonnull__
     (1, 2, 3))) ;
3017.    # 1000 "/usr/include/stdlib.h" 3 4
3018.    extern int getloadavg (double __loadavg[], int __nelem)
3019.          __attribute__ ((__nothrow__ , __leaf__)) __attribute__ ((__nonnull__
     (1)));
3020.    # 1010 "/usr/include/stdlib.h" 3 4
3021.    # 1 "/usr/include/x86_64-linux-gnu/bits/stdlib-float.h" 1 3 4
3022.    # 1011 "/usr/include/stdlib.h" 2 3 4
3023.    # 1020 "/usr/include/stdlib.h" 3 4
3024.
3025.    # 9 "hello.c" 2
3026.
3027.
3028.    # 10 "hello.c"
3029.    int main(int argc,char *argv[]){
3030.     int i;
3031.
3032.     if(argc!=4){
3033.      printf("用法: Hello 学号 姓名 秒数! \n");
3034.      exit(1);
3035.     }
3036.     for(i=0;i<8;i++){
3037.      printf("Hello %s %s\n",argv[1],argv[2]);
3038.      sleep(atoi(argv[3]));
3039.     }
3040.     getchar();
3041.     return 0;
3042.    }
```

2.3 Hello 的预处理结果解析

预处理将 hello.c 中的宏和头文件全部展开，注释删除，得到一个 3000+行的

完整的源代码。

2.4 本章小结

本章节简单介绍了 c 语言文件在编译之前的预处理过程，即.c 文件生成.i 文本文件的过程，对预处理的含义、具体执行过程和预处理结果进行了解析。

（第 2 章 0.5 分）

第 3 章 编译

3.1 编译的概念与作用

概念：编译是指利用编译程序从源语言编写的源程序产生目标程序的过程，这里是将预处理文本文件 hello.i 翻译成汇编语言程序 hello.s 的过程。

作用：编译过程是指对预处理生存的.i 文件进行一系列词法分析、语法分析、语义分析，最后优化后生成相应的汇编代码文件。具体步骤如下：

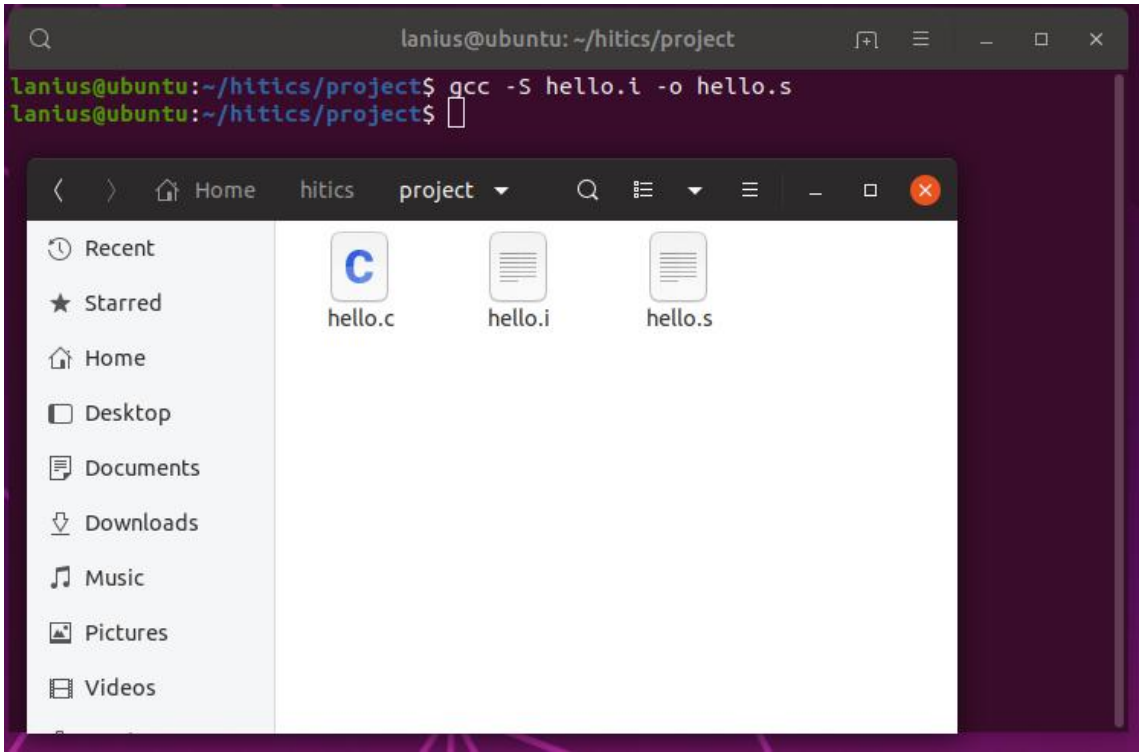
（1）词法分析，将字符串转化为内部表示结构；

（2）语法分析，将上一步分析得到的标记流转化生成为一棵语法树；

（3）生成汇编代码，将语法树转化为汇编目标代码。

3.2 在 Ubuntu 下编译的命令

gcc -S hello.i -o hello.s




```
1.      .file   "hello.c"
2.      .text
3.      .section    .rodata
4.      .align 8
5.  .LC0:
6.      .string "\347\224\250\346\263\225: Hello \345\255\246\345\217\267 \345\247
    \223\345\220\215 \347\247\222\346\225\260\357\274\201"
7.  .LC1:
8.      .string "Hello %s %s\n"
9.      .text
10.     .globl  main
11.     .type   main, @function
12. main:
13. .LFB6:
14.     .cfi_startproc
15.     pushq   %rbp
16.     .cfi_def_cfa_offset 16
17.     .cfi_offset 6, -16
18.     movq    %rsp, %rbp
19.     .cfi_def_cfa_register 6
20.     subq    $32, %rsp
21.     movl    %edi, -20(%rbp)
22.     movq    %rsi, -32(%rbp)
23.     cmpl    $4, -20(%rbp)
24.     je  .L2
25.     leaq    .LC0(%rip), %rdi
```

```
    call    puts@PLT
    movl    $1, %edi
    call    exit@PLT
.L2:
    movl    $0, -4(%rbp)
    jmp .L3
.L4:
    movq    -32(%rbp), %rax
    addq    $16, %rax
    movq    (%rax), %rdx
    movq    -32(%rbp), %rax
    addq    $8, %rax
    movq    (%rax), %rax
    movq    %rax, %rsi
    leaq    .LC1(%rip), %rdi
    movl    $0, %eax
    call    printf@PLT
    movq    -32(%rbp), %rax
    addq    $24, %rax
    movq    (%rax), %rax
    movq    %rax, %rdi
    call    atoi@PLT
    movl    %eax, %edi
    call    sleep@PLT
    addl    $1, -4(%rbp)
.L3:
    cmpl    $7, -4(%rbp)
    jle .L4
    call    getchar@PLT
    movl    $0, %eax
    leave
    .cfi_def_cfa 7, 8
    ret
    .cfi_endproc
.LFE6:
    .size   main, .-main
    .ident  "GCC: (Ubuntu 8.3.0-6ubuntu1) 8.3.0"
    .section    .note.GNU-stack,"",@progbits
```

3.3 Hello 的编译结果解析

3.3.1 数据

3.3.2 赋值

（1）赋值语句 i=0，编译器将其编译后变成如下汇编代码:

```
movl    $0, -4(%rbp)
```

利用栈底指针 rbp 加上偏移量-4 对第一个局部变量 i 进行赋值。

3.3.3 类型转换

变量 argv[3]是 char*类型的，但 sleep()的参数要求 int 型，因此在这里进行了类型转换，调用了 atoi()函数，汇编代码如下:

3.3.4 算数操作

计数变量 i++的语句，编译器将其编译后变成如下汇编代码：

```
movl    %eax, %edi
call    sleep@PLT
```

3.3.5 关系操作

判断语句往往编译成 cmp 系列语句和 j 系列条件跳转语句的组合，具体参考 CSAPP 课本中相关部分的介绍。

（1）判断语句 argc!=4 的，编译器将其编译后变成如下汇编代码：

```
cmpl    $4, -20(%rbp)
je      .L2
```

（2）计数变量 i<8 的判断语句，编译器将其编译后变成如下汇编代码：

```
cmpl    $7, -4(%rbp)
jle     .L4
```

3.3.6 数组/指针/结构操作

取数组的第 i 位一般是按照取数组头指针加上第 i 位的偏移量来操作的；指针跟数组类似，如果 x 表示一个指针，rax 表示其存储的寄存器，你要访问*x，那么就是(%rax)；结构也是类似的，通过结构在结构体中的偏移量来访问。

3.3.7 控制转移

if,for 循环等控制转移语句都是由比较语句和条件跳转来实现的，比较语句被编译器编译成 cmp 语句，条件跳转则由 je、jle 等实现。

（1）if (argc!=4) 编译器将其编译后变成如下汇编代码：

```
cmpl    $4, -20(%rbp)
```

（2）for 循环语句编译器将其编译后变成如下汇编代码：

```
.L4:
    movq    -32(%rbp), %rax
```

```
    addq    $16, %rax
    movq    (%rax), %rdx
    movq    -32(%rbp), %rax
    addq    $8, %rax
    movq    (%rax), %rax
    movq    %rax, %rsi
    leaq.LC1(%rip), %rdi
    movl    $0, %eax
    call printf@PLT
    movq    -32(%rbp), %rax
    addq    $24, %rax
    movq    (%rax), %rax
    movq    %rax, %rdi
    call atoi@PLT
    movl    %eax, %edi
    call sleep@PLT
    addl$1, -4(%rbp)
.L3:
    cmpl    $7, -4(%rbp)
    jle  .L4
    call getchar@PLT
    movl    $0, %eax
    leave
    .cfi_def_cfa 7, 8
    ret
    .cfi_endproc
```

3.3.8 函数操作

一个函数的返回值一般存在寄存器 eax 中，如果要设定返回值的话，那就先将返回值传入 eax，然后再用 ret 语句返回。

（1）输出语句 printf("用法: Hello 学号 姓名 秒数！\n");首先编译器将输出的字符串进行编译，转换为如下代码：

```
.LC0:
    .string "\347\224\250\346\263\225: Hello \345\255\246\345\217\267
\345\247\223\345\220\215 \347\247\222\346\225\260\357\274\201"
```

然后将输出语句编译后变成如下汇编代码：

```
leaq    .LC0(%rip), %rdi
```

```
Call    puts@PLT
```

（2）输出语句 printf("Hello %s %s\n",argv[1],argv[2])，首先编译器将输出的字符串进行编译，转换为如下代码：

```
.LC1:
    .string   "Hello %s %s\n"
    .text
    .globl    main
    .type     main, @function
```

然后将输出语句编译后变成如下汇编代码：

```
callprintf@PLT
```

（2）系统函数 sleep(atoi(argv[3]));编译器将其编译后变成如下汇编代码：

```
 movq   %rax, %rdi
call atoi@PLT
movl    %eax, %edi
call sleep@PLT
```

3.4 本章小结

本章简单介绍了编译的概念和作用，并以 hello.c 的编译过程为实例详细剖析了一个 c 程序是如何被编译器翻译成汇编语言文件 hello.s 的，并分类展示出不同功能的 c 语句对应的汇编语句，并简单给出了对应的编译原理。

本章通过实例分析，有助于学生将所学的理论知识与实际样例相结合，更加立体地学习和感受编译原理，有助于加深对计算机执行程序的过程的理解。

（第 3 章 2 分）

第 4 章 汇编

4.1 汇编的概念与作用

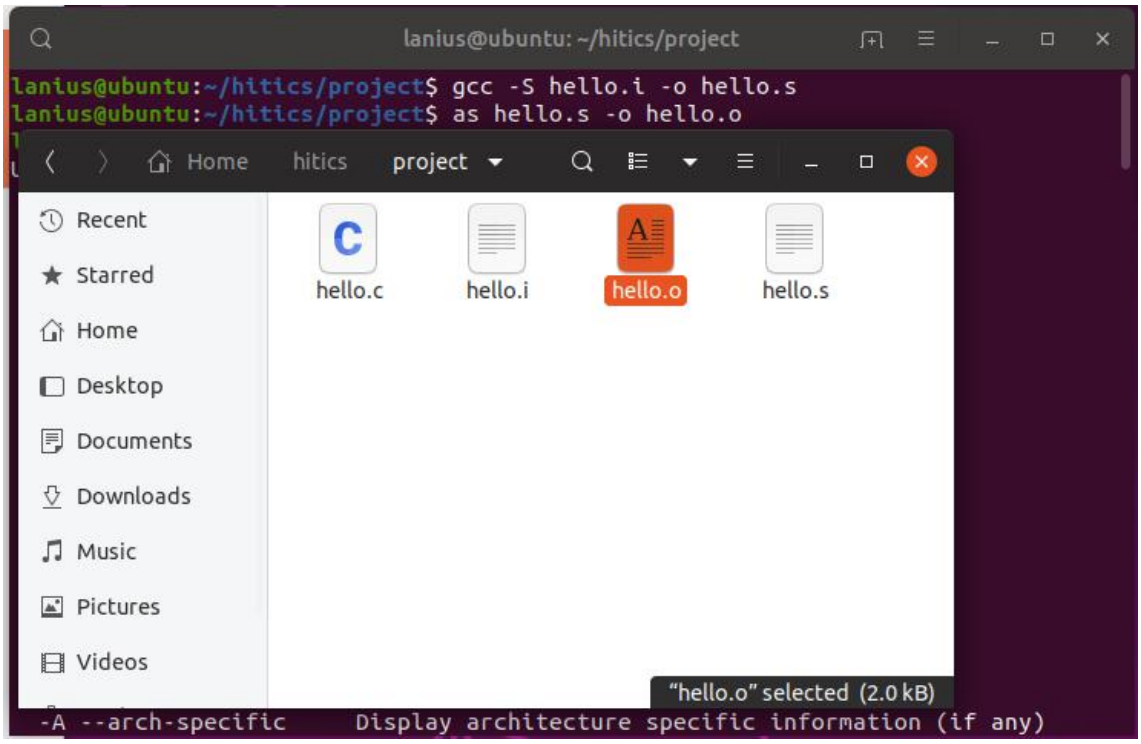
概念：汇编是指汇编器（as）将汇编文件 hello.s 翻译成机器语言指令，并把这些指令打包成一种叫做可重定位目标文件的格式，并将结果保存在目标文件 hello.o 中。

作用：汇编器是将汇编代码转变成机器可以执行的命令，每一个汇编语句几乎都对应一条机器指令。汇编相对于编译过程比较简单，根据汇编指令和机器指令的对照表一一翻译即可。

4.3 可重定位目标 elf 格式

as hello.s -o hello.o

之后就生成了二进制文件 hello.o 如下：



4.4 Hello.o 的结果解析

用 readelf 打开 hello.o 文件，命令行为 readelf -a hello.o，如下面各图所示。

得到的 elf 格式文件组成如下：（1）ELF 头，用于总的描述 ELF 文件各个信息的段；

（2）节头，描述了.o 文件中出现的各个节的类型、位置、所占空间大小等信息；

```
ELF Header:
  Magic:   7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00
  Class:                             ELF64
  Data:                              2's complement, little endian
  Version:                           1 (current)
  OS/ABI:                            UNIX - System V
  ABI Version:                       0
  Type:                              REL (Relocatable file)
  Machine:                           Advanced Micro Devices X86-64
  Version:                           0x1
  Entry point address:               0x0
  Start of program headers:          0 (bytes into file)
  Start of section headers:          1152 (bytes into file)
  Flags:                             0x0
  Size of this header:               64 (bytes)
  Size of program headers:           0 (bytes)
  Number of program headers:         0
  Size of section headers:           64 (bytes)
  Number of section headers:         13
  Section header string table index: 12
```

（3）重定位节.rela.text，这个节包含了.text（具体指令）节中需要进行重定位的信息。这些信息描述的位置，在由.o 文件生成可执行文件的时候需要被修改（重定位）。在这个 hello.o 里面需要被重定位的有 printf , puts , exit , sleepsecs , getchar , sleep ,rodata 里面的两个元素（.L0 和.L1 字符串）。.rela.eh_frame 是 eh_frame 节的重定位信息。

```
Relocation section '.rela.text' at offset 0x340 contains 8 entries:
  Offset          Info           Type           Sym. Value    Sym. Name + Addend
000000000018  000500000002 R_X86_64_PC32     0000000000000000 .rodata - 4
00000000001d  000b00000004 R_X86_64_PLT32    0000000000000000 puts - 4
000000000027  000c00000004 R_X86_64_PLT32    0000000000000000 exit - 4
000000000050  000500000002 R_X86_64_PC32     0000000000000000 .rodata + 22
00000000005a  000d00000004 R_X86_64_PLT32    0000000000000000 printf - 4
```

```
00000000006d  000e00000004 R_X86_64_PLT32    0000000000000000 atoi - 4
000000000074  000f00000004 R_X86_64_PLT32    0000000000000000 sleep - 4
000000000083  001000000004 R_X86_64_PLT32    0000000000000000 getchar - 4

Relocation section '.rela.eh_frame' at offset 0x400 contains 1 entry:
  Offset          Info           Type           Sym. Value    Sym. Name + Addend
000000000020  000200000002 R_X86_64_PC32     0000000000000000 .text + 0

The decoding of unwind sections for machine type Advanced Micro Devices X86-64 is not currently supported.
```

4.5 本章小结

本章简单介绍了汇编这一过程，即汇编器（as）将汇编文件 hello.s 翻译成机器语言指令并存储到 hello.o 文件中的过程，同样以 hello 程序为实例，将理论知识与实际操作的过程穿插编排。

本章通过实例分析与理论知识讲解相结合，有助于学生将课本上课学到的知识与实际操作结合，融会贯通，更加深入地学习和感受汇编的过程，有助于加深对汇编语言和汇编过程的理解。

（第 4 章 1 分）

第 5 章 链接

5.1 链接的概念与作用

概念：链接是将各种代码和数据片段收集并组合成一个单一文件的过程，这个文件可被加载到内存并执行。链接可以执行于编译时，也就是在源代码被编译成机器代码时；也可以执行于加载时，也就是在程序被加载器加载到内存并执行时；甚至于运行时，也就是由应用程序来执行。

作用：链接是由叫做链接器的程序执行的。链接器使得分离编译成为可能。链接的存在降低了模块化编程的难度。

5.2 在 Ubuntu 下链接的命令

ld -o hello -dynamic-linker /lib64/ld-linux-x86-64.so.2 /usr/lib/x86_64-linux-gnu/crt1.o /usr/lib/x86_64-linux-gnu/crti.o hello.o /usr/lib/x86_64-linux-gnu/libc.so /usr/lib/x86_64-linux-gnu/crtn.o

5.3 可执行目标文件 hello 的格式

键入 readelf -a hello >hello.elf 命令生成 hello 程序的 ELF 格式文件 hello.elf。打开 hello.elf，首先是 ELF 头部分如下：

节头信息如下：

```
ELF Header:
  Magic:   7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00
  Class:                             ELF64
  Data:                              2's complement, little endian
  Version:                           1 (current)
  OS/ABI:                            UNIX - System V
  ABI Version:                       0
  Type:                              REL (Relocatable file)
  Machine:                           Advanced Micro Devices X86-64
  Version:                           0x1
  Entry point address:               0x0
  Start of program headers:          0 (bytes into file)
  Start of section headers:          1152 (bytes into file)
  Flags:                             0x0
  Size of this header:               64 (bytes)
```

```
Size of program headers:           0 (bytes)
Number of program headers:         0
Size of section headers:           64 (bytes)
Number of section headers:         13
Section header string table index: 12
```

节头对 hello 中所有的节信息进行了声明，其中包括大小 Size 以及在程序中的偏移量 Offset，因此根据节头中的信息我们就可以用 HexEdit 定位各个节所占的区间（起始位置，大小）。其中 Address 是程序被载入到虚拟地址的起始地址。

5.4 hello 的虚拟地址空间

PHDR：程序头表
INTERP：程序执行前需要调用的解释器
LOAD：程序目标代码和常量信息
DYNAMIC：动态链接器所使用的信息
NOTE:：辅助信息
GNU_EH_FRAME：保存异常信息
GNU_STACK：使用系统栈所需要的权限信息
GNU_RELRO：保存在重定位之后只读信息的位置
其余的从.dynamic 到.strtab 节的内容是存放在 0x00400fff 后面

5.5 链接的重定位过程分析

对比 5.3 的节头信息和 4.3 的节头信息，hello 相对于 hello.o 有如下不同：
1. hello.o 中的相对偏移地址到了 hello 中变成了虚拟内存地址
2. hello 中相对 hello.o 增加了许多的外部链接来的函数。
3. hello 相对 hello.o 多了很多的节类似于.init,.plt 等
4. hello.o 中跳转以及函数调用的地址在 hello 中都被更换成了虚拟内存地址
分析可得比 hello.o 多出来如下节表头：
.interp：保存 ld.so 的路径
.note.ABI-tag
.note.gnu.build-i：编译信息表
.gnu.hash：gnu 的扩展符号 hash 表
.dynsym：动态符号表
.dynstr：动态符号表中的符号名称
.gnu.version：符号版本
.gnu.version_r：符号引用版本
.rela.dyn：动态重定位表
.rela.plt：.plt 节的重定位条目
.init：程序初始化
.plt：动态链接表
.fini：程序终止时需要的执行的指令

.eh_frame：程序执行错误时的指令
.dynamic：存放被 ld.so 使用的动态链接信息
.got：存放程序中变量全局偏移量
.got.plt：存放程序中函数的全局偏移量
.data：初始化过的全局变量或者声明过的函数

接下来分析一下汇编程序。
键入 objdump -d -r hello 得到将 hello 的反汇编，将 hello.o 的反汇编之后的结果存入 hello_o.s
我们可以发现，在 hello_o.s 中调用函数都是使用 call+<main+偏移量的做法>，而在 hello_.s 是直接使用 call+<函数名> 的方法来直接调用的。

5.6 hello 的执行流程

1. 先是加载程序_init (argc=1, argv=0x7fffffffde38, envp=0x7fffffffde48)
2. 0x00000000004004d0 in _start ()
3. 0x0000000000400480 in __libc_start_main@plt ()
4. 0x0000000000400670 in __libc_csu_init ()
5. 0x0000000000400430 in _init ()
6. 0x00000000004005b0 in frame_dummy ()
7. 0x0000000000400540 in register_tm_clones ()
8. 0x00000000004005f2 in main ()
9. 0x0000000000400460 in puts@plt ()
10. 0x00000000004004a0 in exit@plt ()
11. 0x0000000000400580 in __do_global_dtors_aux ()
12. 0x0000000000400500 in deregister_tm_clones ()
13. 0x00000000004006e4 in _fini ()

5.7 Hello 的动态链接分析

对于动态共享链接库中 PIC 函数，编译器没有办法预测函数的运行时地址，所以需要添加重定位记录，等待动态链接器处理，为避免运行时修改调用模块的代码段，链接器采用延迟绑定的策略。动态链接器使用过程链接表 PLT+全局偏移量表 GOT 实现函数的动态链接，GOT 中存放函数目标地址，PLT 使用 GOT 中地址跳转到目标函数。在 dl_init 调用之前，对于每一条 PIC 函数调用，调用的目标地址都实际指向 PLT 中的代码逻辑，GOT 存放的是 PLT 中函数调用指令的下一条指令地址。
在 dl_init 调用之后，0x601008 和 0x601010 处的两个 8B 数据分别发生改变为 0x7fd9 d3925170 和 0x7fd9 d3713680，其中 GOT[1]指向重定位表（依次为.plt 节需要重定位的函数的运行时地址）用来确定调用的函数地址，GOT[2]指向动态链接器 ld-linux.so 运行时地址。

在之后的函数调用时，首先跳转到 PLT 执行.plt 中逻辑，第一次访问跳转时 GOT

地址为下一条指令，将函数序号压栈，然后跳转到 PLT[0]，在 PLT[0]中将重定位表地址压栈，然后访问动态链接器，在动态链接器中使用函数序号和重定位表确定函数运行时地址，重写 GOT，再将控制传递给目标函数。之后如果对同样的函数调用，第一次访问跳转直接跳转到目标函数。
因为在 PLT 中使用的 jmp，所以执行完目标函数之后的返回地址为最近 call 指令下一条指令地址，即在 main 中的调用完成地址。

5.8 本章小结

本章主要介绍了链接的概念与作用，对 hello 的 ELF 格式文件包含的信息进行解析，分析了 hello 的虚拟地址空间、重定位过程、执行流程、动态链接过程。

本章对链接的步骤和过程进行了详细的分解和解析，这样有利于学生在以后处理相关的问题的时候沉着应对。

（第 5 章 1 分）

第 6 章 hello 进程管理

6.1 进程的概念与作用

概念：一个执行中程序的实例。

作用：每次用户通过向 shell 输入一个可执行目标文件的名字，运行程序时， shell 就会创建一个新的进程，然后在这个新进程的上下文中运行这个可执行目标文件。应用程序也能够创建新进程，并且在这个新进程的上下文中运行它们自己的代码或其他应用程

6.2 简述壳 Shell-bash 的作用与处理流程

作用：：Shell 是一个用 C 语言编写的程序应用程序，他在操作系统中提供了一个用户与系统内核进行交互的界面。
处理流程一般是这样的：
1. 读取用户的输入
2. 分析输入内容，获得输入参数
3. 如果是内核命令则直接执行，否则调用相应的程序执行命令
4. 在程序运行期间，shell 需要监视键盘的输入内容，并且做出相应的反应

6.3 Hello 的 fork 进程创建过程

Shell 通过调用 fork 函数创建一个新的运行的子进程。也就是 Hello 程序，Hello 进程几乎但不完全与 Shell 相同。Hello 进程得到与 Shell 用户级虚拟地址空间相同的（但是独立的）一份副本，包括代码和数据段、堆、共享库以及用户栈。Hello 进程还获得与 Shell 任何打开文件描述符相同的副本，这就意味着当 Shell 调用 fork 时，Hello 可以读写 Shell 中打开的任何文件。Sehll 和 Hello 进程之间最大的区别在于它们有不同的 PID。

6.4 Hello 的 execve 过程

每一个进程都有一段唯一属于自己的内存地址段，在 execve 运行时，开始先是从 0x00400000(对于 32 位系统来说是 0x8048000)开始程序的执行。先是从可执行文件中加载的内容，然后是运行时的堆栈和共享库的存储器映射区域。

当 fork 之后，子进程调用 execve 函数（传入命令行参数）在当前进程的上下文中加载并运行一个新程序即 hello 程序。execve 函数加载并运行可执行目标文件 hello, 且带参数列表 argv 和环境变量列表 envp 。只有当出现错误时，例如找不到 hello,

execve 才会返回到调用程序。所以，与 fork 一次调用返回两次不同， execve 调用一次并从不返回。

6.5 Hello 的进程执行

Linux 系统中的每个程序都运行在一个进程上下文中，有自己的虚拟地址空间。当 shell 运行一个程序时，父 shell 进程生成一个子进程，它是父进程的一个复制。子进程通过 execve 系统调用启动加载器。加载器删除子进程现有的虚拟内存段，并创建一组新的代码、数据、堆和栈段。新的栈和堆段被初始化为零。通过将虚拟地址空间中的页映射到可执行文件的页大小的片(chunk), 新的代码和数据段被初始化为可执行文件的内容。最后，加载器跳转到_start 地址，它最终会调用应用程序的 main 函数。在内核和前端之前切换的动作被称为上下文切换。

6.6 hello 的异常与信号处理

6.6.1 异常种类

hello 执行过程中会出现的异常种类有：

（1）中断：SIGSTP：挂起程序

（2）终止：SIGINT:终止程序

6.6.2 各种命令的执行

1. ctrl-z：这个操作向进程发送了一个 sigtstp 信号，让进程暂时挂起。

（1）输入 ps 命令符可以发现 hello 进程还没有被关闭：

（2）jobs 命令可以查看当前的关键命令（ctrl+Z/ctrl+C 这类）内容，比如这时候就会返回 ctrl+Z 表示暂停命令：

（3）pstree 是用进程树的方法把各个进程用树状图的方式连接起来：

（4）fg：发送 SIGCONT 信号继续执行停止的进程：

（5）5.kill -9 pid：发送 SIGKILL 信号给指定的 pid 杀死进程：

2.ctrl-c 操作：这个操作向进程发送了一个 sigint 信号，让进程直接结束，输入 ps 命令可以发现当前 hello 进程已经被终止了：

3.运行过程中脸滚键盘并不影响进程：

6.7 本章小结

本章简要阐述了进程的定义和作用，详细介绍了 shell 的一般处理流程，描述了 shell 如何在用户和系统内核之间建起一个交互的桥梁，以 hello 程序为例介绍了 shell 的基本操作以及各种内核信号和命令，还总结了 shell 是如何 fork 新建子进程、execve 如何执行进程、hello 进程如何在内核和前端中反复跳跃运行的。

（第 6 章 1 分）

第 7 章 hello 的存储管理

7.1 hello 的存储器地址空间

逻辑地址：又称相对地址，是程序运行由 CPU 产生的与段相关的偏移地址部分。他是描述一个程序运行段的地址。

物理地址：程序运行时加载到内存地址寄存器中的地址，内存单元的真正地址。他是在前端总线上传输的而且是唯一的。在 hello 程序中，他就表示了这个程序运行时的一条确切的指令在内存地址上的具体哪一块进行执行。

线性地址：这个和虚拟地址是同一个东西，是经过段机制转化之后用于描述程序分页信息的地址。他是对程序运行区块的一个抽象映射，即一个程序应该在内存的哪些块上运行。

7.2 Intel 逻辑地址到线性地址的变换-段式管理

一个逻辑地址由两部份组成，段标识符和段内偏移量。段标识符是由一个 16 位长的字段组成，称为段选择符。其中前 13 位是一个索引号。后面 3 位包含一些硬件细节，如图：

索引号是“段描述符”的索引，很多个段描述符，就组了一个数组，叫“段描述符表”，这样，可以通过段标识符的前 13 位，直接在段描述符表中找到一个具体的段描述符，这个描述符就描述了一个段，每一个段描述符由 8 个字节组成，如下图：

其中 Base 字段，它描述了一个段的开始位置的线性地址，一些全局的段描述符，就放在“全局段描述符表(GDT)”中，一些局部的，例如每个进程自己的，就放在所谓的“局部段描述符表(LDT)”中，由段选择符中的 T1 字段表示选择使用哪个，=0，表示用 GDT，=1 表示用 LDT。GDT 在内存中的地址和大小存放在 CPU 的 gdtr 控制寄存器中，而 LDT 则在 ldtr 寄存器中。如下图：

下面是转换的具体步骤：
1. 给定一个完整的逻辑地址[段选择符：段内偏移地址]。
2. 看段选择符的 T1=0 还是 1，知道当前要转换是 GDT 中的段，还是 LDT 中的段，再根据相应寄存器，得到其地址和大小。可以得到一个数组。
3. 取出段选择符中前 13 位，在数组中查找到对应的段描述符，得到 Base，也就是基地址。
4. 线性地址 = Base + offset。

7.3 Hello 的线性地址到物理地址的变换-页式管理

线性地址（书里的虚拟地址 VA）到物理地址（PA）之间的转换通过分页机制完成。而分页机制是对虚拟地址内存空间进行分页。线性地址被分为以固定长度为单位的组，称为页(page)。为了节省空间，引入了一个二级管理模式的机器来组织分页单元。

1.分页单元中，页目录是唯一的，它的地址放在 CPU 的 cr3 寄存器中，是进行地址转换的开始点；

2.每一个活动的进程，因为都有其独立的对应的虚似内存（页目录也是唯一的），那么它也对应了一个独立的页目录地址。——运行一个进程，需要将它的页目录地址放到 cr3 寄存器中；

3.每一个 32 位的线性地址被划分为三部份，面目录索引(10 位)：页表索引(10 位)：偏移(12 位)。

依据以下步骤进行转换：

1.从 cr3 中取出进程的页目录地址（操作系统负责在调度进程的时候，把这个地址装入对应寄存器）；

2.根据线性地址前十位，在数组中，找到对应的索引项，因为引入了二级管理模式，页目录中的项，不再是页的地址，而是一个页表的地址。（又引入了一个数组），页的地址被放到页表中去了；

3.根据线性地址的中间十位，在页表（也是数组）中找到页的起始地址；

4.将页的起始地址与线性地址中最后 12 位相加，得到最终我们想要的物理地址。

7.4 TLB 与四级页表支持下的 VA 到 PA 的变换

下图给出了 Core i7 MMU 如何使用四级的页表来将虚拟地址翻译成物理地址。36 位 VPN 被划分成四个 9 位的片，每个片被用作到一个页表的偏移量。CR3 寄存器包含 Ll 页表的物理地址。VPN 1 提供到一个 Ll PET 的偏移量，这个 PTE 包含 L2 页表的基地址。VPN 2 提供到一个 L2 PTE 的偏移量，以此类推。

7.5 三级 Cache 支持下的物理内存访问

得到物理地址之后，先将物理地址拆分成 CT（标记）+CI（索引）+CO（偏移量），然后在一级 cache 内部找，如果未能寻找到标记位为有效的字节（miss）的话就去二级和三级 cache 中寻找对应的字节，找到之后返回结果。

7.6 hello 进程 fork 时的内存映射

mm_struct（内存描述符）：描述了一个进程的整个虚拟内存空间

vm_area_struct（区域结构描述符）：描述了进程的虚拟内存空间的一个区间

在用 fork 创建虚拟内存的时候，要经历以下步骤：
1. 创建当前进程的 mm_struct,vm_area_struct 和页表的原样副本
2. 两个进程的每个页面都标记为只读页面
3. 两个进程的每个 vm_area_struct 都标记为私有，这样就只能在写入时复制。

7.7 hello 进程 execve 时的内存映射

execve 函数调用驻留在内核区域的启动加载器代码，在当前进程中加载并运行包含在可执行目标文件 hello 中的程序，用 hello 程序有效地替代了当前程序。加载并运行 hello 需要以下几个步骤：
1） 删除已存在的用户区域，删除当前进程虚拟地址的用户部分中的已存在的区域结构；
2） 映射私有区域，为新程序的代码、数据、bss 和栈区域创建新的区域结构，所有这些新的区域都是私有的、写时复制的。代码和数据区域被映射为 hello 文件中的.text 和.data 区，bss 区域是请求二进制零的，映射到匿名文件，其大小包含在 hello 中，栈和堆地址也是请求二进制零的，初始长度为零；
3） 映射共享区域， hello 程序与共享对象 libc.so 链接，libc.so 是动态链接到这个程序中的，然后再映射到用户虚拟地址空间中的共享区域内；
4） 设置程序计数器（PC），execve 做的最后一件事情就是设置当前进程上下文的程序计数器，使之指向代码区域的入口点。

7.8 缺页故障与缺页中断处理

在虚拟内存的习惯说法中， DRAM 缓存不命中称为缺页(page fault)
情况 1：段错误：首先，先判断这个缺页的虚拟地址是否合法，那么遍历所有的合法区域结构，如果这个虚拟地址对所有的区域结构都无法匹配，那么就返回一个段错误（segment fault）
情况 2：非法访问：接着查看这个地址的权限，判断一下进程是否有读写改这个地址的权限。
情况 3：如果不是上面两种情况那就是正常缺页，那就选择一个页面牺牲然后换入新的页面并更新到页表。

7.9 动态存储分配管理

printf 函数会调用 malloc，下面简述动态内存管理的基本方法与策略：
动态内存分配器维护着一个进程的虚拟内存区域，称为堆。分配器将堆视为一组不同大小的块的集合来维护。每个块就是一个连续的虚拟内存片，要么是已分配的，要么是空闲的。已分配的块显式地保留为供应用程序使用。空闲块可用来分配。空闲块保持空闲，直到它显式地被应用所分配。一个已分配的块保持已分配状态，直到它被释放，这种释放要么是应用程序显式执行的，要么是内存分配器自身隐式执行的。分配器分为两种基本风格：显式分配器、隐式分配器。显式分

配器：要求应用显式地释放任何已分配的块。隐式分配器：要求分配器检测一个已分配块何时不再使用，那么就释放这个块，自动释放未使用的已经分配的块的过程叫做垃圾收集。

一、 带边界标签的隐式空闲链表

1）堆及堆中内存块的组织结构：

在内存块中增加 4B 的 Header 和 4B 的 Footer，其中 Header 用于寻找下一个 blcok，Footer 用于寻找上一个 block。Footer 的设计是专门为了合并空闲块方便的。因为 Header 和 Footer 大小已知，所以我们利用 Header 和 Footer 中存放的块大小就可以寻找上下 block。

2）隐式链表所谓隐式空闲链表，对比于显式空闲链表，代表并不直接对空闲块进行链接，而是将对内存空间中的所有块组织成一个大链表，其中 Header 和 Footer 中的 block 大小间接起到了前驱、后继指针的作用。

3）空闲块合并

因为有了 Footer，所以我们可以方便的对前面的空闲块进行合并。合并的情况一共分为四种：前空后不空，前不空后空，前后都空，前后都不空。对于四种情况分别进行空闲块合并，我们只需要通过改变 Header 和 Footer 中的值就可以完成这一操作。

二、 显示空间链表基本原理

将空闲块组织成链表形式的数据结构。堆可以组织成一个双向空闲链表，在每个空闲块中，都包含一个 pred（前驱）和 succ（后继）指针，如下图：

使用双向链表而不是隐式空闲链表，使首次适配的分配时间从块总数的线性时间减少到了空闲块数量的线性时间。

维护链表的顺序有：后进先出（LIFO），将新释放的块放置在链表的开始处，使用 LIFO 的顺序和首次适配的放置策略，分配器会最先检查最近使用过的块，在这种情况下，释放一个块可以在线性的时间内完成，如果使用了边界标记，那么合并也可以在常数时间内完成。按照地址顺序来维护链表，其中链表中的每个块的地址都小于它的后继的地址，在这种情况下，释放一个块需要线性时间的搜索来定位合适的前驱。平衡点在于，按照地址排序首次适配比 LIFO 排序的首次适配有着更高的内存利用率，接近最佳适配的利用率。

7.10 本章小结

本章介绍了储存器的地址空间，讲述了虚拟地址、物理地址、线性地址、逻辑地址的概念，还有进程 fork 和 execve 时的内存映射的内容。描述了系统如何应对那些缺页异常，最后描述了 malloc 的内存分配管理机制。

（第 7 章 2 分）

第 8 章 hello 的 I0 管理

8.1 Linux 的 I0 设备管理方法

设备的模型化：文件
文件的类型：
1. 普通文件（regular file）：包含任意数据的文件。
2. 目录（directory）：包含一组链接的文件，每个链接都将一个文件名映射到一个文件（他还有另一个名字叫做“文件夹”）。
3. 套接字（socket）：用来与另一个进程进行跨网络通信的文件
4. 命名通道
5. 符号链接
6. 字符和块设备

设备管理：unix io 接口
1. 打开和关闭文件
2. 读取和写入文件
3. 改变当前文件的位置

8.2 简述 Unix I0 接口及其函数

Unix I/O 接口：
1.打开文件。一个应用程序通过要求内核打开相应的文件，来宣告它想要访问一个 I/O 设备。内核返回一个小的非负整数，叫做描述符，它在后续对此文件的所有操作中标识这个文件。内核记录有关这个打开文件的所有信息。应用程序只需记住这个描述符。
2.Linux shell 创建的每个进程开始时都有三个打开的文件：标准输入（描述符为 0）、标准输出（描述符为 1）和标准错误（描述符为 2）。头文件< unistd.h> 定义了常量 STDIN_FILENO 、STOOUT_FILENO 和 STDERR_FILENO, 它们可用来代替显式的描述符值。
3.改变当前的文件位置。对于每个打开的文件，内核保持着一个文件位置 k, 初始为 0。这个文件位置是从文件开头起始的字节偏移量。应用程序能够通过执行 seek 操作，显式地设置文件的当前位置为 K 。
4.读写文件。一个读操作就是从文件复制 n>0 个字节到内存，从当前文件位置 k 开始，然后将 k 增加到 k+n 。给定一个大小为 m 字节的文件，当 k~m 时执行读操作会触发一个称为 end-of-file(EOF) 的条件，应用程序能检测到这个条件。在文件结尾处并没有明确的“EOF 符号” 。类似地，写操作就是从内存复制 n>0 个字节到一个文件，从当前文件位置 k 开始，然后更新 k 。
5.关闭文件。当应用完成了对文件的访问之后，它就通知内核关闭这个文件。作为

响应，内核释放文件打开时创建的数据结构，并将这个描述符恢复到可用的描述符池中。无论一个进程因为何种原因终止时，内核都会关闭所有打开的文件并释放它们的内存资源。

Unix I/O 函数：
1.进程是通过调用 open 函数来打开一个已存在的文件或者创建一个新文件的：int open(char *filename, int flags, mode_t mode);
open 函数将 filename 转换为一个文件描述符，并且返回描述符数字。返回的描述符总是在进程中当前没有打开的最小描述符。flags 参数指明了进程打算如何访问这个文件，mode 参数指定了新文件的访问权限位。
返回：若成功则为新文件描述符，若出错为-1。
2.进程通过调用 close 函数关闭一个打开的文件。
int close(int fd);
返回：若成功则为 0, 若出错则为-1。
3.应用程序是通过分别调用 read 和 write 函数来执行输入和输出的。
（1）ssize_t read(int fd, void *buf, size_t n);
read 函数从描述符为 fd 的当前文件位置复制最多 n 个字节到内存位置 buf 。返回值-1 表示一个错误，而返回值 0 表示 EOF。否则，返回值表示的是实际传送的字节数量。
返回：若成功则为读的字节数，若 EOF 则为 0, 若出错为-1。
（2）ssize_t write(int fd, const void *buf, size_t n);
write 函数从内存位置 buf 复制至多 n 个字节到描述符 fd 的当前文件位置。图 10-3 展示了一个程序使用 read 和 write 调用一次一个字节地从标准输入复制到标准输出。
返回：若成功则为写的字节数，若出错则为-1。

8.3 printf 的实现分析

8.4 getchar 的实现分析

异步异常-键盘中断的处理：键盘中断处理子程序。接受按键扫描码转成 ascii 码，保存到系统的键盘缓冲区。
getchar 等调用 read 系统函数，通过系统调用读取按键 ascii 码，直到接受到回车键才返回。
1.查看 getchar 函数：

（1）异步异常-键盘中断的处理：当用户按键时，键盘接口会得到一个代表该按键的键盘扫描码，同时产生一个中断请求，中断请求抢占当前进程运行键盘中断子程序，键盘中断子程序先从键盘接口取得该按键的扫描码，然后将该按键扫描码转换成 ASCII 码，保存到系统的键盘缓冲区之中。
（2）可以看出，这里面调用了一个 read 函数，通过系统调用读取按键 ascii 码，直到接受到回车键才返回。这个 read 函数是将整个缓冲区都读到了 buf 里面，返

回值是缓冲区的长度。我们可以发现，如果 buf 长度为 0，getchar 才会调用 read 函数，否则是直接将保存的 buf 中的最前面的元素返回。

8.5 本章小结

本章主要介绍了 Linux 的 IO 设备管理方法、Unix IO 接口及其函数，分析了 printf 函数和 getchar 函数。这有助于我们以后在写函数的时候在标准 I/O 库没有的情况下编写自己的 I/O 函数。

（第 8 章 1 分）

结论

hello 程序的一生：

（1）编写，在键盘鼠标等 I/O 设备下，hello.c 被编写出来并以文件的方式储存在主存里面；

（2）预处理，将 hello.c 调用的所有外部的库展开合并，预处理成为文本文件 hello.i；

（3）编译，将 hello.i 编译成为汇编文件 hello.s；

（4）汇编，将 hello.s 汇编成为可重定位目标文件 hello.o；

（5）链接，链接器将外部文件和 hello.o 链接在一起成为可执行二进制文件 hello；

（6）运行：在 shell 中输入./hello 1173710111 张淑慧 运行该程序；

（7）创建子进程：shell 进程调用 fork 为其创建子进程，shell 调用 execve，execve 调用启动加载器，加映射虚拟内存，进入程序入口后程序开始载入物理内存，然后进入 main 函数；

（8）执行指令：CPU 为其分配时间片，在一个时间片中，hello 享有 CPU 资源，顺序执行自己的控制逻辑流。hello 在执行的过程中可能会遇到异常和信号以及命令，执行异常信号的处理流程；

（9）内存申请和访问：MMU 将程序中使用的虚拟内存地址通过页表映射成物理地址，printf 会调用 malloc 向动态内存分配器申请堆中的内存；

（10）结束运行：shell 父进程回收子进程，内核删除为这个进程创建的所有数据结构。

-1 分，根据内容酌情加分）

附件

列出所有的中间产物的文件名，并予以说明起作用。

（附件 0 分，缺失 -1 分）

参考文献

为完成本次大作业你翻阅的书籍与网站等

[1] 林来兴. 空间控制技术[M]. 北京：中国宇航出版社，1992：25-42.

[2] 辛希孟. 信息技术与信息服务国际研讨会论文集：A 集[C]. 北京：中国科学出版社，1999.

[3] 赵耀东. 新时代的工业工程师[M/OL]. 台北：天下文化出版社，1998 [1998-09-26]. http://www.ie.nthu.edu.tw/info/ie.newie.htm（Big5）.

[4] 谌颖. 空间交会控制理论与方法研究[D]. 哈尔滨：哈尔滨工业大学，1992：8-13.

[5] KANAMORI H. Shaking Without Quaking[J]. Science，1998，279（5359）：2063-2064.

[6] CHRISTINE M. Plant Physiology: Plant Biology in the Genome Era[J/OL]. Science，1998，281：331-332[1998-09-23]. http://www.sciencemag.org/cgi/collection/anatmorp.

（参考文献 0 分，缺失 -1 分）