

AUP Assignment 8

111703013 Akshay Rajesh Deodhar

23rd October 2020

Q1

Write a program to take input from user for number of files to be scanned and word to be searched. Write a multi threaded program to search the files and return pattern if found.

Code

```
1
2  #include <sys/types.h>
3  #include <unistd.h>
4  #include <fcntl.h>
5  #include <pthread.h>
6  #include <errno.h>
7  #include <stdio.h>
8  #include <stdlib.h>
9  #include <string.h>
10 #include <ctype.h>
11
12
13 #define BUF_SIZE 1024
14 typedef struct {
15     int fp;
16     int offset;
17     int chars_read;
18     char buf[BUF_SIZE];
19 }read_buf;
20
21 void get_read_buf(read_buf *b, int fp) {
22     b->fp = fp;
23     b->offset = 0;
24     b->chars_read = -1;
25 }
26
27 void destroy_read_buf(read_buf *b) {
28     close(b->fp);
29     b->offset = b->fp = 0;
30     b->chars_read = -1;
31 }
32
33 int getchar_buffered(read_buf *b) {
34     if (b->chars_read < 0 || b->offset == b->chars_read) {
35         if ((b->chars_read = read(b->fp, b->buf, BUF_SIZE)) == -1) {
36             perror("read");
37             exit(errno);
38         }
39     }
```

```

40         if (!b->chars_read) {
41             /* this is the end of file */
42             return EOF;
43         }
44
45         /* assert: buffer has nonzero number of bytes */
46         b->offset = 0;
47     }
48
49     return b->buf[++(b->offset)];
50 }
51
52 typedef struct {
53     char *filepath;
54     char *word;
55     int count;
56 }search_params;
57
58
59 #define MAX_WORD 512
60 #define IN 101
61 #define OUT 102
62 void *search_word(void *arg) {
63     search_params *sp;
64     sp = (search_params *)arg;
65
66     int c;
67     int fp;
68     read_buf rb;
69
70     int lineno = 1;
71
72     int state = OUT;
73     char word[MAX_WORD + 1];
74     int wordlen = 0;
75
76     if ((fp = open(sp->filepath, O_RDONLY)) == -1) {
77         perror("open");
78         exit(errno);
79     }
80
81     get_read_buf(&rb, fp);
82
83     while ((c = getchar_buffered(&rb)) != EOF) {
84         if (c == '\n') {
85             lineno++;
86         }
87
88         if (state == OUT) {
89             if (isalpha(c) || c == '_') {
90                 state = IN;
91                 wordlen = 0;
92                 word[wordlen++] = c;
93             }
94         }
95         else if (state == IN) {
96             if (!(isalpha(c) || (c == '_'))) {
97                 word[wordlen] = '\0';
98                 state = OUT;

```

```

99         wordlen = 0;
100         /* printf("%s\n", word); */
101         if (strcmp(word, sp->word) == 0) {
102             /* word found */
103             printf("%s: Found word '%s' in line %d\n", sp->filepath, sp->word, lineno);
104             ++sp->count;
105         }
106     }
107     else {
108         word[wordlen++] = c;
109     }
110 }
111 else {
112     fprintf(stderr, "Invalid State\n");
113     exit(1);
114 }
115 }
116
117 destroy_read_buf(&rb);
118
119 /* pthread_exit(NULL); */
120 return &(sp->count);
121 }
122
123 int main(int argc, char *argv[]) {
124
125     /* search word file1 file2 .. */
126     if (argc < 3) {
127         fprintf(stderr, "usage: ./search <word> <file1> [<file2> ... <file-n>]\n");
128         return EINVAL;
129     }
130
131     int i; int n_files = (argc - 2); pthread_t *threads; search_params *parameters;
132
133     if ((threads = (pthread_t *)malloc(sizeof(pthread_t) * n_files)) == NULL) {
134         fprintf(stderr, "malloc failed\n");
135         return 1;
136     }
137
138     if ((parameters = (search_params *)malloc(sizeof(search_params) * n_files)) == NULL) {
139         fprintf(stderr, "malloc failed\n");
140         return 1;
141     }
142
143     for (i = 0; i < n_files; i++) {
144
145         /* add file path to parameters */
146         if ((parameters[i].filepath = (char *)malloc(sizeof(char) * (strlen(argv[2]) + 1))) == NULL) {
147             fprintf(stderr, "malloc failed\n");
148             return 1;
149         }
150         strcpy(parameters[i].filepath, argv[2 + i]);
151
152         /* add search pattern to parameters */
153         if ((parameters[i].word = (char *)malloc(sizeof(char) * (strlen(argv[1]) + 1))) == NULL) {
154             fprintf(stderr, "malloc failed\n");
155             return 1;
156         }
157         strcpy(parameters[i].word, argv[1]);

```

```

158
159     /* initialize count to 0 */
160     parameters[i].count = 0;
161
162     /* dispatch thread for searching file */
163     pthread_create(&threads[i],
164                   NULL,
165                   search_word,
166                   (void *)&parameters[i]);
167 }
168
169 int total_count = 0;
170
171 for (i = 0; i < n_files; i++) {
172     if (pthread_join(threads[i], NULL)) {
173         fprintf(stderr, "Unable to join thread\n");
174     }
175     free(parameters[i].word);
176     free(parameters[i].filepath);
177
178     total_count += parameters[i].count;
179 }
180
181 printf("Found %d occurrences of '%s' during search\n", total_count, argv[1]);
182
183 return 0;
184 }

```

Output

```

akshay@akshay-inspiron5423 ~/.../lab/9_pthreads$ ./1 pthread_t 1.c 2.c 3.c
2.c: Found word 'pthread_t' in line 27
2.c: Found word 'pthread_t' in line 27
2.c: Found word 'pthread_t' in line 27
3.c: Found word 'pthread_t' in line 18
1.c: Found word 'pthread_t' in line 130
1.c: Found word 'pthread_t' in line 132
1.c: Found word 'pthread_t' in line 132
Found 7 occurrences of 'pthread_t' during search
akshay@akshay-inspiron5423 ~/.../lab/9_pthreads$ grep pthread_t 1.c
int i; int n_files = (argc - 2); pthread_t *threads; search_params *parameters;
if ((threads = (pthread_t *)malloc(sizeof(pthread_t) * n_files)) == NULL) {
akshay@akshay-inspiron5423 ~/.../lab/9_pthreads$ grep pthread_t 2.c
pthread_t *threads = (pthread_t *)malloc(sizeof(pthread_t) * n);
akshay@akshay-inspiron5423 ~/.../lab/9_pthreads$ grep pthread_t 3.c
pthread_t threads[N_THREADS];
akshay@akshay-inspiron5423 ~/.../lab/9_pthreads$

```

Figure 1: Search for word pthread_t in C source files

Q2

Write a program to find number of CPUs, create that many threads and attach those threads to CPUs

Code

```
1
2  #define _GNU_SOURCE
3  #include <sched.h>
4  #include <sys/sysinfo.h>
5  #include <unistd.h>
6  #include <pthread.h>
7  #include <stdio.h>
8  #include <errno.h>
9  #include <stdlib.h>
10
11 void *busy_void(void *arg) {
12     int count = 100000;
13     while (count--);
14     return NULL;
15 }
16
17 int main(void) {
18
19     int n;
20
21     if ((n = sysconf(_SC_NPROCESSORS_CONF)) == -1) {
22         perror("sysconf");
23         return errno;
24     }
25
26     printf("Number of CPUs: %d\n", n);
27
28     pthread_t *threads = (pthread_t *)malloc(sizeof(pthread_t) * n);
29     cpu_set_t *cpus = (cpu_set_t *)malloc(sizeof(cpu_set_t) * n);
30
31     int i;
32     for (i = 0; i < n; i++) {
33         CPU_ZERO(&cpus[i]);
34         CPU_SET(i, &cpus[i]);
35     }
36
37     for (i = 0; i < n; i++) {
38         if (pthread_create(&threads[i],
39                           NULL,
40                           busy_void,
41                           NULL) == -1) {
42             fprintf(stderr, "Unable to create thread\n");
43         }
44     }
45
46     /* DANGER: non-POSIX code */
47     for (i = 0; i < n; i++) {
48         if (pthread_setaffinity_np(threads[i],
49                                   sizeof(cpu_set_t),
```

```

50         &cpus[i]) == -1) {
51             fprintf(stderr, "Unable to set affinity\n");
52             return 1;
53         }
54     }
55
56     for (i = 0; i < n; i++) {
57         if (pthread_getaffinity_np(threads[i],
58                                 sizeof(cpu_set_t),
59                                 &cpus[i]) == -1) {
60             fprintf(stderr, "Unable to get affinity\n");
61             return 2;
62         }
63         printf("Thread %d is ", i);
64         if (!CPU_ISSET(i, &cpus[i])) {
65             printf("not ");
66         }
67         printf("attached to CPU %d\n", i);
68
69         if (pthread_join(threads[i], NULL) == -1) {
70             fprintf(stderr, "Unable to join with thread %lu\n", threads[i]);
71         }
72     }
73 }
74
75
76 return 0;
77
78 }

```

```

akshay@akshay-inspiron5423 ~/.../lab/9_pthreads$ ./2
Number of CPUs: 4
Thread 0 is attached to CPU 0
Thread 1 is attached to CPU 1
Thread 2 is attached to CPU 2
Thread 3 is attached to CPU 3
akshay@akshay-inspiron5423 ~/.../lab/9_pthreads$

```

Figure 2: Code verifies that threads are running on CPUs which they are attached to

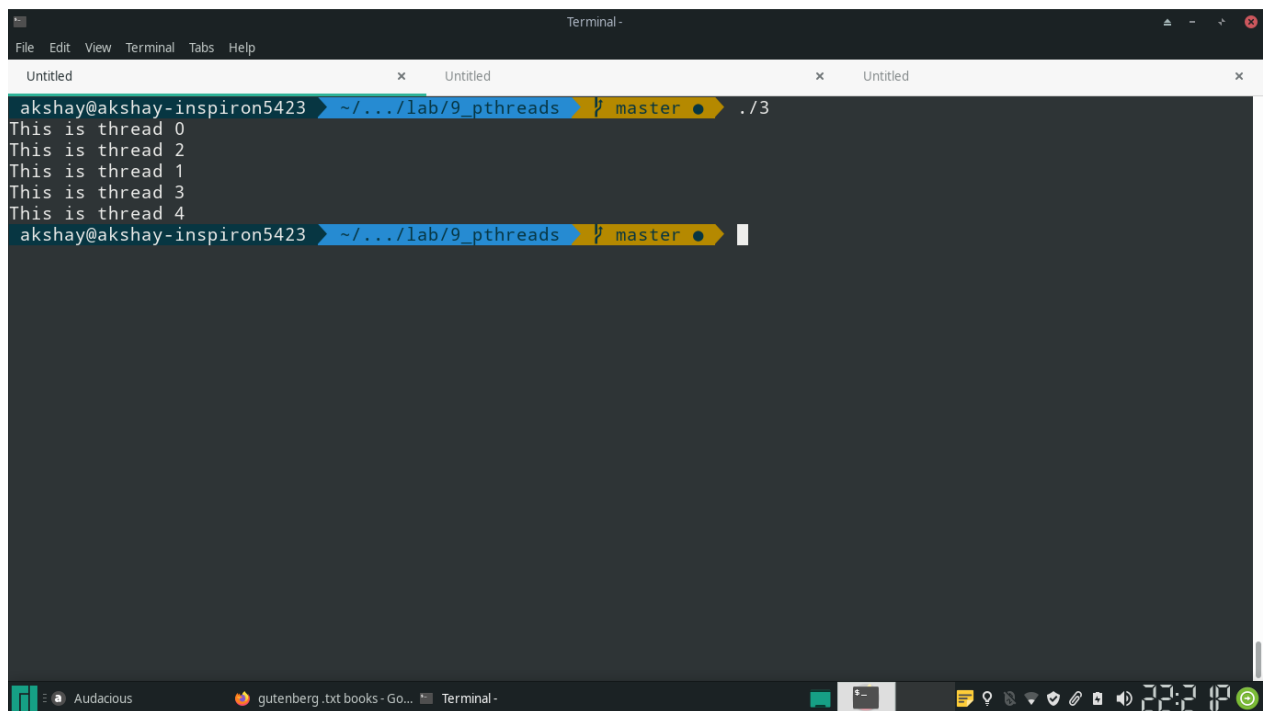
Q3

Write a short program that creates 5 threads which print a thread “id” that is passed to thread function by pointer.

Code

```
1
2
3  #define _GNU_SOURCE
4  #include <sched.h>
5  #include <sys/sysinfo.h>
6  #include <unistd.h>
7  #include <pthread.h>
8  #include <stdio.h>
9  #include <errno.h>
10 #include <stdlib.h>
11
12
13 void *busy_void(void *arg) {
14     printf("This is thread %d\n", *((int *)arg));
15     return NULL;
16 }
17
18 #define N_THREADS 5
19 static int thread_ids[N_THREADS];
20 pthread_t threads[N_THREADS];
21 int main(void) {
22
23     int n, i;
24
25     n = N_THREADS;
26
27     for (i = 0; i < n; i++) {
28         thread_ids[i] = i;
29         if (pthread_create(&threads[i],
30                          NULL,
31                          busy_void,
32                          &thread_ids[i]) == -1) {
33             fprintf(stderr, "Unable to create thread\n");
34         }
35     }
36     for (i = 0; i < n; i++) {
37         if (pthread_join(threads[i], NULL) == -1) {
38             fprintf(stderr, "Unable to join with thread %lu\n", threads[i]);
39         }
40     }
41
42
43     return 0;
44 }
45
```

Output

A terminal window titled "Terminal -" with a menu bar (File, Edit, View, Terminal, Tabs, Help) and two tabs labeled "Untitled". The terminal shows a user prompt "akshay@akshay-inspiron5423" followed by a command ". /3" in a blue prompt. The output consists of five lines: "This is thread 0", "This is thread 2", "This is thread 1", "This is thread 3", and "This is thread 4". Below the output, the prompt "akshay@akshay-inspiron5423" and the command ". /3" are shown again. The terminal is running on a system with a taskbar at the bottom showing icons for Audacious, a web browser, and the terminal itself, along with system status icons on the right.

```
akshay@akshay-inspiron5423 > ~/.../lab/9_pthreads master . /3
This is thread 0
This is thread 2
This is thread 1
This is thread 3
This is thread 4
akshay@akshay-inspiron5423 > ~/.../lab/9_pthreads master . /3
```

Figure 3: Each thread prints it's "thread id". The pointer to the ID was passed to the thread