

AUP Assignment 6

111703013 Akshay Rajesh Deodhar

15th September 2020

Q1

Implement the C program in which main program accepts an integer array. Parent creates two child processes. Parent process sorts the integer array and passes the sorted array to child process through the command line arguments of an exec call. The first child process uses this sorted array to display in ascending order and becomes a zombie process. The second child process uses this sorted array to display in descending order and becomes an orphan process.

Parent

```
1
2  #include <sys/types.h>
3  #include <unistd.h>
4  #include <string.h>
5  #include <stdlib.h>
6  #include <stdint.h>
7  #include <stdio.h>
8  #include <errno.h>
9
10 #define ZOMBIE 1
11 #define ORPHAN 2
12
13 #define BUFSIZE 100
14 #define MAXLEN 12
15 #define N 100
16 #define MOD 1000
17
18 static int32_t buf[BUFSIZE];
19 static int32_t arr[BUFSIZE];
20
21 /* merges a[left:mid], a[mid:right], using temp */
22 void merge(int32_t *a, int32_t left, int32_t right, int32_t *buf) {
23     int32_t mid;
24     int32_t size = left;
25     int32_t lp, rp;
26
27     mid = (left + right) / 2;
28
29     lp = left;
30     rp = mid;
31
32     while (lp < mid && rp < right) {
33         if (a[lp] <= a[rp]) {
34             buf[size++] = a[lp++];
35         }
36         else {
```

```

37         buf[size++] = a[rp++];
38     }
39 }
40
41 int32_t start, end;
42 if (lp == mid) {
43     start = rp;
44     end = right;
45 }
46 else {
47     start = lp;
48     end = mid;
49 }
50
51 while (start < end) {
52     buf[size++] = a[start++];
53 }
54
55
56 memcpy(a + left, buf + left, sizeof(int32_t) * (right - left));
57 }
58
59
60 void mergesort_serial(int32_t *a, int32_t left, int32_t right, int32_t *buf) {
61     int32_t mid = (left + right) / 2;
62
63     if ((right - left) <= 1) {
64         /* already sorted */
65         return;
66     }
67
68     mergesort_serial(a, left, mid, buf);
69     mergesort_serial(a, mid, right, buf);
70     merge(a, left, right, buf);
71
72     return;
73 }
74
75 void print_arr(int32_t *arr, int32_t n) {
76     int32_t i;
77     for (i = 0; i < n; i++) {
78         printf("%d ", arr[i]);
79     }
80     printf("\n");
81 }
82
83 void read_array(int32_t *arr, int32_t n) {
84     int32_t i;
85     for (i = 0; i < n; i++) {
86         arr[i] = 0;
87         scanf("%d", &arr[i]);
88     }
89 }
90
91
92
93 int32_t main(int32_t argc, char *argv[]) {
94
95     extern char **environ;

```

```

96
97     int32_t n, i;
98
99     scanf("%d", &n);
100
101     read_array(arr, n);
102
103     mergesort_serial(arr, 0, n, buf);
104
105     char **sorted_array = (char **)malloc(sizeof(char *) * (n + 1));
106     if (!sorted_array) {
107         fprintf(stderr, "unable to allocate sufficient memory\n");
108         return 1;
109     }
110     for (i = 0; i < n; i++) {
111         sorted_array[i] = (char *)malloc(sizeof(char) * MAXLEN);
112         if (!sorted_array[i]) {
113             fprintf(stderr, "malloc failed while allocating string\n");
114         }
115         sprintf(sorted_array[i], "%d", arr[i]);
116     }
117     sorted_array[n] = NULL;
118
119     /* this trickery works because the size of int32_t is 4 bytes
120      * and the (char *) pointers of argv will either be 4 byte or
121      * 8 byte (on 32 and 64 bit machines respectively, never less */
122
123     if (!fork()) {
124         /* first child */
125         execve("./1_child1", sorted_array, environ);
126     }
127
128     sleep(5);
129     printf("First child Zombied\n");
130     system("ps -o pid,ppid,stat,comm");
131
132     if (!fork()) {
133         /* second child */
134         execve("./1_child2", sorted_array, environ);
135     }
136
137     /* for first child to truly become zombie, the parent should remain in
138      * while loop forever to ensure that child exits first.
139      * However, we also want the second child to become an orphan, which
140      * means that the parent needs to exit.
141      * HENCE, sleep() calls have been inserted appropriately so that the
142      * 1. first child exists first (thus becoming a zombie for some time)
143      * 2. Then, the parent exits (sleeps for 5)
144      * 3. Finally, the second child exits (sleeps for 10)
145      *
146      * A while loop will not be an appropriate solution here
147      * The parent is not going to reap either of the children
148      */
149 }
150

```

First Child

```
1
2  #include <stdio.h>
3
4  /* Child 1: prints array in ascending order and exits, becoming zombie */
5  int main(int argc, char *argv[]) {
6      int i;
7      printf("\nIn First Child\n");
8      for (i = 0; i < argc; i++) {
9          printf("%s ", argv[i]);
10     }
11     return 0;
12 }
```

Second Child

```
1
2
3  #include <unistd.h>
4  #include <stdlib.h>
5  #include <stdio.h>
6
7  /* Child 2: prints array in descending order and exits after parent becoming
8  orphan*/
9  int main(int argc, char *argv[]) {
10     int i;
11
12     printf("\nIn Child 2\n");
13
14     for (i = argc - 1; i > -1; i--) {
15         printf("%s ", argv[i]);
16     }
17
18     fflush(stdout);
19
20     sleep(5);
21
22     system("ps -o pid,ppid,stat,comm");
23
24     printf("Second child became orphan\n");
25 }
```

Output

```

Terminal -
File Edit View Terminal Tabs Help
akshay@akshay-inspiron5423 ~/.../lab/6_pctl_even_more 1 master • make 1
clang 1_child2.c -Wall -o 1_child2
clang 1.c -Wall -o 1
akshay@akshay-inspiron5423 ~/.../lab/6_pctl_even_more 1 master • ./1
1 1_child1 1_child2
akshay@akshay-inspiron5423 ~/.../lab/6_pctl_even_more 1 master • ./1
8
1 4 3 2 6 8 7 5

In First Child
1 2 3 4 5 6 7 8 First child Zombied
PID PPID STAT COMMAND
8813 1628 Ss bash
19971 8813 S+ 1
19972 19971 Z+ 1_child1 <defunct>
19975 19971 R+ ps

In Child 2
8 7 6 5 4 3 2 1 akshay@akshay-inspiron5423 ~/.../lab/6_pctl_even_more 1 master • PID PPID STAT COMMAND
8813 1628 Ss+ bash
19976 1 S 1_child2
20022 19976 R ps
Second child became orphan

```

Figure 1: Two children printing array in ascending and descending order, and becoming zombie and orphan respectively

Q2

Create a game program that switches between the effective user ID and real user ID. The game player may write details (like game iteration number) to a file owned by the game player and manipulates a scores file that should be writable only by the game program owner. Both the game program and scores file are owned by the game program owner. Demonstrate that the game player can switch between the files in turns as own file, scores file, own file and scores file.

```

1
2 #include <sys/types.h>
3 #include <sys/stat.h>
4 #include <fcntl.h>
5 #include <unistd.h>
6 #include <string.h>
7 #include <stdlib.h>
8 #include <stdint.h>
9 #include <stdio.h>
10 #include <errno.h>
11 #include <stdio.h>
12
13 #define OWN_FILE "2_own_file.txt"
14 #define SCORES_FILE "2_scores_file.txt"
15
16 #define BUFSIZE 100
17
18 int dice(int n) {
19     return rand() % n;
20 }
21
22 static char buf[BUFSIZE];

```

```

23
24 int main(void) {
25
26     /* On exec:
27      * RUID  EUID  SSUID
28      * player owner owner */
29
30     int n, i, score, roll, chars_printed;
31     int fp_own, fp_scores;
32     uid_t player, owner;
33
34     /* program knows that it has owner perms due to sticky bit */
35
36     player = getuid(); /* player is real user id */
37
38     owner = geteuid(); /* owner is effective user id */
39
40     scanf("%d", &n);
41
42     score = 0;
43
44     for (i = 0; i < n; i++) {
45
46         roll = dice(6);
47
48         score += roll;
49
50         chars_printed = sprintf(buf, "Iteration: %d\tRolled: %d\tScore: %d\n", i, roll, score);
51
52         /* when entering loop:
53          * RUID  EUID  SSUID
54          * player owner owner */
55
56         if ((fp_scores = open(SCORES_FILE, O_APPEND | O_WRONLY, S_IRUSR | S_IWUSR)) == -1) {
57             perror(SCORES_FILE);
58             return errno;
59         }
60
61         if (write(fp_scores, buf, chars_printed) != chars_printed) {
62             perror("writing score to scores file");
63             return errno;
64         }
65
66         if (setuid(player) == -1) {
67             perror("switching to player");
68             return errno;
69         }
70
71         /* now
72          * RUID  EUID  SSUID
73          * player player owner */
74
75         if ((fp_own = open(OWN_FILE, O_APPEND | O_WRONLY, S_IRUSR | S_IWUSR)) == -1) {
76             perror(OWN_FILE);
77             return errno;
78         }
79
80         if (write(fp_own, buf, chars_printed) != chars_printed) {
81             perror("writings score to player file");

```

```

82         return errno;
83     }
84
85     if (setuid(owner) == -1) {
86         perror("switching to owner");
87         return errno;
88     }
89
90     /* now
91      * RUID      EUID      SSUID
92      * player    owner    player*/
93 }
94 }

```

Output

The terminal window shows the execution of a program in a directory named `~/lab/6_pctl_even_more`. The user `akshay` is running the program as `master`. The output shows the initial status of two files, `2_own_file.txt` and `2_scores_file.txt`, and the results of 5 iterations of a game. The game involves rolling a die and updating scores. The final output shows the scores after 5 iterations.

```

akshay@akshay-inspiron5423 ~/lab/6_pctl_even_more master ● ls -l 2_2_own_file.txt 2_scores_file.txt
-rw-r--r-- 1 aup akshay 17192 Sep 30 16:59 2
-rw-r--r-- 1 akshay akshay 64 Sep 30 17:03 2_own_file.txt
-rw-r--r-- 1 aup akshay 64 Sep 30 17:03 2_scores_file.txt
akshay@akshay-inspiron5423 ~/lab/6_pctl_even_more master ● # aup is "owner", akshay is "player"
akshay@akshay-inspiron5423 ~/lab/6_pctl_even_more master ● cat 2_own_file.txt
Iteration: 0 Rolled: 1 Score: 1
Iteration: 1 Rolled: 4 Score: 5
akshay@akshay-inspiron5423 ~/lab/6_pctl_even_more master ● cat 2_scores_file.txt
Iteration: 0 Rolled: 1 Score: 1
Iteration: 1 Rolled: 4 Score: 5
akshay@akshay-inspiron5423 ~/lab/6_pctl_even_more master ● # initial status of the two files
akshay@akshay-inspiron5423 ~/lab/6_pctl_even_more master ● ./2
5
akshay@akshay-inspiron5423 ~/lab/6_pctl_even_more master ● # game played for 5 iterations
akshay@akshay-inspiron5423 ~/lab/6_pctl_even_more master ● cat 2_own_file.txt
Iteration: 0 Rolled: 1 Score: 1
Iteration: 1 Rolled: 4 Score: 5
Iteration: 0 Rolled: 1 Score: 1
Iteration: 1 Rolled: 4 Score: 5
Iteration: 2 Rolled: 3 Score: 8
Iteration: 3 Rolled: 1 Score: 9
Iteration: 4 Rolled: 5 Score: 14
akshay@akshay-inspiron5423 ~/lab/6_pctl_even_more master ● cat 2_scores_file.txt
Iteration: 0 Rolled: 1 Score: 1
Iteration: 1 Rolled: 4 Score: 5
Iteration: 0 Rolled: 1 Score: 1
Iteration: 1 Rolled: 4 Score: 5
Iteration: 2 Rolled: 3 Score: 8
Iteration: 3 Rolled: 1 Score: 9
Iteration: 4 Rolled: 5 Score: 14

```

Figure 2: File permissions, contents before and after execution of 5 iterations of the game

Q3

Write a program to do the following:

1. Create a child
2. let the child
 1. Create it's own foreground process group.
 2. Call "ps" and verify the above
 3. Verify whether the process has controlling terminal
3. Let the parent
 1. Shift the child to it's own foreground process group.
 2. Check whether the process is in back ground or foreground and has controlling terminal.
 3. Wait for the child to terminate
 4. Check whether the process is in back ground or foreground.

```
1
2
3 #define _DEFAULT_SOURCE
4
5 #include <sys/types.h>
6 #include <unistd.h>
7 #include <sys/wait.h>
8 #include <string.h>
9 #include <stdlib.h>
10 #include <stdint.h>
11 #include <stdio.h>
12 #include <errno.h>
13 #include <stdio.h>
14
15 int is_process_foreground(void) {
16
17     int p_pgrp, fg_pgrp;
18
19     if ((p_pgrp = getpgrp()) == -1) {
20         perror("parent getpgrp");
21         return errno;
22     }
23
24     if ((fg_pgrp = tcgetpgrp(STDIN_FILENO)) == -1) {
25         perror("parent tcgetpgrp");
26         return errno;
27     }
28
29     if (fg_pgrp == p_pgrp) {
30         return 1;
31     }
32     else {
33         return 0;
34     }
35 }
36
37
38
39
40 int main(void) {
41
```



```

42     int cpid;
43     int status;
44
45     if ((cpid = fork())) {
46         /* parent */
47         if (setpgid(cpid, 0) == -1) {
48             perror("setpgid in parent");
49             return errno;
50         }
51
52         if (tcsetpgrp(STDIN_FILENO, cpid) == -1) {
53             perror("tcsetpgrp in parent");
54             return errno;
55         }
56
57         if (is_process_foreground()) {
58             printf("Before Wait: Parent is foreground\n");
59         }
60         else {
61             printf("Before Wait: Parent is NOT foreground\n");
62         }
63
64         if (wait(&status) == -1) {
65             perror("wait");
66             return errno;
67         }
68
69         if (is_process_foreground()) {
70             printf("After Wait: Parent is foreground\n");
71         }
72         else {
73             printf("After Wait: Parent is NOT foreground\n");
74         }
75     }
76     else {
77         /* child */
78
79
80         if (setpgid(0, 0) == -1) {
81             perror("setpgid in child");
82             return errno;
83         }
84
85
86         if (tcsetpgrp(STDIN_FILENO, getpgid(0)) == -1) {
87             perror("tcsetpgrp in parent");
88             return errno;
89         }
90
91         if (system("ps -o cmd,pid,ppid,pgid,tpgid") == -1) {
92             perror("ps");
93             return errno;
94         }
95

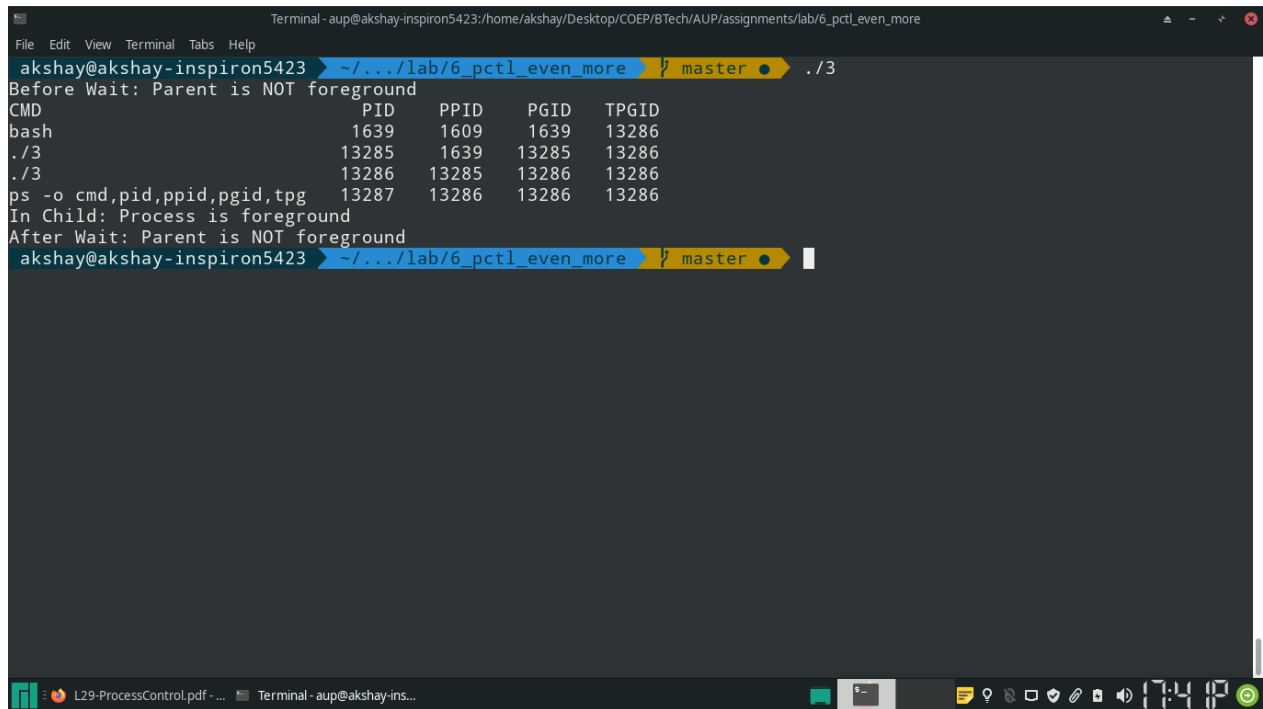
```

```

96         if (is_process_foreground()) {
97             printf("In Child: Process is foreground\n");
98         }
99         else {
100             printf("In Child: Process is NOT foreground\n");
101         }
102     }
103 }

```

Output



```

Terminal - aup@akshay-inspiron5423:/home/akshay/Desktop/COEP/BTech/AUP/assignments/lab/6_pctl_even_more
File Edit View Terminal Tabs Help
akshay@akshay-inspiron5423 ~/.../lab/6_pctl_even_more master . /3
Before Wait: Parent is NOT foreground
CMD      PID    PPID   PGID   TPGID
bash     1639   1609   1639   13286
./3      13285  1639   13285  13286
./3      13286  13285  13286  13286
ps -o cmd,pid,ppid,pgid,tpg 13287 13286 13286 13286
In Child: Process is foreground
After Wait: Parent is NOT foreground
akshay@akshay-inspiron5423 ~/.../lab/6_pctl_even_more master .

```

Figure 3: Output of ps, and check for which process has the controlling terminal