

AUP : Assignment - 6 [Process Control Advanced]

Aditi Rajendra Medhane 111803177

11th October 2021

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.

Code

Parent

```
1  #include <sys/types.h>
2  #include <unistd.h>
3  #include <string.h>
4  #include <stdlib.h>
5  #include <stdint.h>
6  #include <stdio.h>
7  #include <errno.h>
8
9  #define ZOMBIE 1
10 #define ORPHAN 2
11
12 #define BUFSIZE 100
13 #define MAXLEN 12
14 #define N 100
15 #define MOD 1000
16
17 static int32_t buf[BUFSIZE];
18 static int32_t arr[BUFSIZE];
19
20 /* merges a[left:mid], a[mid:right], using temp */
21 void merge(int32_t *a, int32_t left, int32_t right, int32_t *buf) {
22     int32_t mid;
23     int32_t size = left;
24     int32_t lp, rp;
25
26     mid = (left + right) / 2;
27
28     lp = left;
29     rp = mid;
30
31     while (lp < mid && rp < right) {
32         if (a[lp] <= a[rp]) {
33             buf[size++] = a[lp++];
34         }
35         else {
36             buf[size++] = a[rp++];
37         }
38     }
39
40     int32_t start, end;
41     if (lp == mid) {
```

```

42         start = rp;
43         end = right;
44     }
45     else {
46         start = lp;
47         end = mid;
48     }
49
50     while (start < end) {
51         buf[size++] = a[start++];
52     }
53
54
55     memcpy(a + left, buf + left, sizeof(int32_t) * (right - left));
56 }
57
58
59 void mergesort_serial(int32_t *a, int32_t left, int32_t right, int32_t *buf) {
60     int32_t mid = (left + right) / 2;
61
62     if ((right - left) <= 1) {
63         /* already sorted */
64         return;
65     }
66
67     mergesort_serial(a, left, mid, buf);
68     mergesort_serial(a, mid, right, buf);
69     merge(a, left, right, buf);
70
71     return;
72 }
73
74 void print_arr(int32_t *arr, int32_t n) {
75     int32_t i;
76     for (i = 0; i < n; i++) {
77         printf("%d ", arr[i]);
78     }
79     printf("\n");
80 }
81
82 void read_array(int32_t *arr, int32_t n) {
83     int32_t i;
84     for (i = 0; i < n; i++) {
85         arr[i] = 0;
86         scanf("%d", &arr[i]);
87     }
88 }
89
90
91
92 int32_t main(int32_t argc, char *argv[]) {
93
94     extern char **environ;
95
96     int32_t n, i;
97
98     scanf("%d", &n);
99
100    read_array(arr, n);
101
102    mergesort_serial(arr, 0, n, buf);
103
104    char **sorted_array = (char **)malloc(sizeof(char *) * (n + 1));
105    if (!sorted_array) {
106        fprintf(stderr, "unable to allocate sufficient memory\n");
107        return 1;
108    }

```

```

109     for (i = 0; i < n; i++) {
110         sorted_array[i] = (char *)malloc(sizeof(char) * MAXLEN);
111         if (!sorted_array[i]) {
112             fprintf(stderr, "malloc failed while allocating string\n");
113         }
114         sprintf(sorted_array[i], "%d", arr[i]);
115     }
116     sorted_array[n] = NULL;
117
118     if (!fork()) {
119         /* first child */
120         execve("./c_1", sorted_array, environ);
121     }
122
123     sleep(5);
124     printf("First child Zombied\n");
125     system("ps -o pid,ppid,stat,comm");
126
127     if (!fork()) {
128         /* second child */
129         execve("./c_2", sorted_array, environ);
130     }
131 }

```

CHILD1

```

1  #include <stdio.h>
2
3  //CHILD 1 : Prints array in ascending order & exits
4  //      Becomes ZOMBIE
5
6  int main(int argc, char *argv[]){
7      int i;
8      printf("\nINSIDE CHILD 1\n");
9      for(i = 0; i < argc; i++){
10         printf("%s ", argv[i]);
11     }
12     return 0;
13 }

```

CHILD2

```

1  #include <stdio.h>
2  #include <stdlib.h>
3
4  #include <unistd.h>
5
6  //CHILD 2: Prints array in descending order
7  //      Exits after parent becoming ORPHAN
8
9  int main(int argc, char *argv[]){
10     int i;
11     printf("\nINSIDE CHILD 2\n");
12
13     for(i = argc -1; i > -1; i--){
14         printf("%s ", argv[i]);
15     }
16
17     fflush(stdout);
18
19     sleep(5);
20
21     system("ps -o pid,ppid,stat,comm");
22
23     printf("Second Child became ORPHAN\n");
24     return 0;
25 }

```

Output

```
hp@aditi:~/Desktop/BTech/AUP/LAB/6$ ./1
1      c_1      c_2

INSIDE CHILD 1
0 First child Zombied
  PID    PPID  STAT  COMMAND
  10105   10097  Ss    bash
  19253   10105  S+    1
  19266   19253  Z+    c_1 <defunct>
  19267   19253  R+    ps
hp@aditi:~/Desktop/BTech/AUP/LAB/6$
INSIDE CHILD 2
0      PID    PPID  STAT  COMMAND
  10105   10097  Ss+   bash
  19268   5853   S     c_2
  19269   19268  R     ps
Second Child became ORPHAN
```

Figure 1: 2 children printing array in ascending and descending order, and becoming ZOMBIE and ORPHAN respectively

Explanation

- For first child to truly become ZOMBIE, the parent should remain in while loop forever to ensure that child exits first.
- However, we also want the second child to become an orphan, which means that the parent needs to exit.
- HENCE, sleep() calls have been inserted appropriately so that the :
 1. first child exists first (thus becoming a zombie for some time)
 2. Then, the parent exits (sleeps for 5)
 3. Finally, the second child exists (sleeps for 10)
- A while loop will not be an appropriate solution here. The parent is not going to reap either of the children

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  #include <sys/types.h>
2  #include <sys/stat.h>
3  #include <fcntl.h>
4  #include <unistd.h>
5  #include <string.h>
6  #include <stdlib.h>
7  #include <stdint.h>
8  #include <stdio.h>
9  #include <errno.h>
10 #include <stdio.h>
11
12 #define OWN_FILE "own_file.txt"
13 #define SCORES_FILE "scores_file.txt"
14
15 #define BUFSIZE 100
16
17 //To generate random values for outcome of dice
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 = real user id */
37
38     owner = geteuid(); /* OWNER = 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

```

hp@aditi:~/Desktop/BTech/AUP/LAB/6$ # INITIAL STATUS of owner & player
hp@aditi:~/Desktop/BTech/AUP/LAB/6$ cat own_file.txt
Iteration: 0    Rolled: 1    Score: 1
Iteration: 1    Rolled: 4    Score: 5
hp@aditi:~/Desktop/BTech/AUP/LAB/6$ cat scores_file.txt
Iteration: 0    Rolled: 1    Score: 1
Iteration: 1    Rolled: 4    Score: 5
hp@aditi:~/Desktop/BTech/AUP/LAB/6$ # Owner = "aditi" and Player = "hp"
hp@aditi:~/Desktop/BTech/AUP/LAB/6$ ls -l 2 scores_file.txt own_file.txt
-rwxrwxr-x 1 aditi hp    17216 Oct 11 06:12 2
-rw-rw-r-- 1 hp    hp      73 Oct 11 06:20 own_file.txt
-rw-rw-r-- 1 hp    aditi   82 Oct 11 06:20 scores_file.txt
hp@aditi:~/Desktop/BTech/AUP/LAB/6$ # PLAY GAME FOR 4 iterations
hp@aditi:~/Desktop/BTech/AUP/LAB/6$ ./2
4
hp@aditi:~/Desktop/BTech/AUP/LAB/6$ cat 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
hp@aditi:~/Desktop/BTech/AUP/LAB/6$ cat 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
hp@aditi:~/Desktop/BTech/AUP/LAB/6$ 

```

Figure 2: File permissions, contents before and after execution of 4 iterations

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  #define _DEFAULT_SOURCE
2
3  #include <sys/types.h>
4  #include <unistd.h>
5  #include <sys/wait.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 //To check if process is in foreground or not
14 int is_ps_fg(void) {
15
16     int p_pgrp, fg_pgrp;
17
18     if ((p_pgrp = getpgrp()) == -1) {
19         perror("parent getpgrp");
20         return errno;
21     }
22
23     if ((fg_pgrp = tcgetpgrp(STDIN_FILENO)) == -1) {
24         perror("parent tcgetpgrp");
25         return errno;
26     }
27
28     if (fg_pgrp == p_pgrp) {
29         return 1;
30     }
31     else {
32         return 0;
33     }
34 }
35
36
37
38
39 int main(void) {
40
41     int cpid;
42     int status;
43
44     if ((cpid = fork()) {
45         /* parent */
46         if (setpgid(cpid, 0) == -1) {
47             perror("setpgid in parent");
48             return errno;
49         }
50
51         if (tcsetpgrp(STDIN_FILENO, cpid) == -1) {
52             perror("tcsetpgrp in parent");
53             return errno;
```



```

54     }
55
56     if (is_ps_fg()) {
57         printf("BEFORE WAIT: Parent is foreground\n\n");
58     }
59     else {
60         printf("BEFORE WAIT: Parent is NOT foreground\n\n");
61     }
62
63     if (wait(&status) == -1) {
64         perror("wait");
65         return errno;
66     }
67
68     if (is_ps_fg()) {
69         printf("AFTER WAIT: Parent is foreground\n\n");
70     }
71     else {
72         printf("AFTER WAIT: Parent is NOT foreground\n\n");
73     }
74 }
75 else {
76     /* child */
77
78
79     if (setpgid(0, 0) == -1) {
80         perror("setpgid in child");
81         return errno;
82     }
83
84
85     if (tcsetpgrp(STDIN_FILENO, getpgid(0)) == -1) {
86         perror("tcsetpgrp in parent");
87         return errno;
88     }
89
90     if (system("ps -o cmd,pid,ppid,pgid,tpgid") == -1) {
91         perror("ps");
92         return errno;
93     }
94
95     if (is_ps_fg()) {
96         printf("\nIN CHILD: Process is foreground\n\n");
97     }
98     else {
99         printf("\nIN CHILD: Process is NOT foreground\n\n");
100    }
101 }
102 }

```

Output

```
hp@aditi:~/Desktop/BTech/AUP/LAB/6$ ./3
BEFORE WAIT: Parent is NOT foreground

CMD          PID    PPID    PGID    TPGID
bash         10105  10097   10105   18479
./3          18478  10105   18478   18479
./3          18479  18478   18479   18479
ps -o cmd,pid,ppid,pgid,tpg 18480  18479   18479   18479

IN CHILD: Process is foreground

AFTER WAIT: Parent is NOT foreground

hp@aditi:~/Desktop/BTech/AUP/LAB/6$
```

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