AUP Assignment 5

111703013 Akshay Rajesh Deodhar

15th September 2020

Q1

> Implement a program in which parent sorts an integer array. Then it creates a child process. The child accepts a number to be searched in the array, performs a binary search in the array and display the result. Appropriately modify the program to create scenarios to demonstrate that zombie and orphan states of the child can be formed.

Answer

1. **For zombie process**:

- The parent is made to sleep for 10 second. Then, the child process accepts an input, searches it, displays the result, and exits.
- When this happens, Ctrl+Z is used to stop the parent process, and *ps -s* is run.
- The output shows that the child process has zombied.

2. **For orphan process**:

- The parent process is made to exit immediately without sleeping. Then the child process **tries to** access an input, searches, displays the result and leaves.
- The child however **fails to execute scanf, and scanf returns -1, failing**. This is because when the parent exited, the child was adopted by *init*. Due to this it relenquishes control of *stdin*, causing scanf to fail.
- The child then calls getppid(), and prints it's and the parent's PID. The ppid is **1**, which means the parent is *init*. This indicates that the child was *orphaned*.

Code

```
1  #include <sys/types.h>
2  #include <unistd.h>
3  #include <string.h>
4  #include <stdlib.h>
5  #include <stdio.h>
```

```c

#define ZOMBIE 1
#define ORPHAN 2

#define BUFSIZE 100
#define N 100
#define MOD 1000
static int buf[BUFSIZE];
static int arr[BUFSIZE];

void random_array(int *arr, int n, int limit) {
        int i;
        for (i = 0; i < n; i++) {
                arr[i] = rand() % limit;
        }
}

int bin_search(int *arr, int n, int x) {
        int left, right, mid;
        int mid_elem;
        left = 0;
        right = n - 1;
        while (left <= right) {
                mid = (left + right) / 2;
                mid_elem = arr[mid];
                if (x < mid_elem) {
                        right = mid - 1;
                }
                else if (x > mid_elem) {
                        left = mid + 1;
                }
                else {
                        return mid;
                }
        }
        return -1;
}


/* merges a[left:mid], a[mid:right], using temp */
void merge(int *a, int left, int right, int *buf) {
        int mid;
        int size = left;
        int lp, rp;

        mid = (left + right) / 2;

        lp = left;
        rp = mid;

```

```
56          while (lp < mid && rp < right) {
57                  if (a[lp] <= a[rp]) {
58                          buf[size++] = a[lp++];
59                  }
60                  else {
61                          buf[size++] = a[rp++];
62                  }
63          }
64
65          int start, end;
66          if (lp == mid) {
67                  start = rp;
68                  end = right;
69          }
70          else {
71                  start = lp;
72                  end = mid;
73          }
74
75          while (start < end) {
76                  buf[size++] = a[start++];
77          }
78
79
80          memcpy(a + left, buf + left, sizeof(int) * (right - left));
81  }
82
83
84  void mergesort_serial(int *a, int left, int right, int *buf) {
85          int mid = (left + right) / 2;
86
87          if ((right - left) <= 1) {
88                  /* already sorted */
89                  return;
90          }
91
92          mergesort_serial(a, left, mid, buf);
93          mergesort_serial(a, mid, right, buf);
94          merge(a, left, right, buf);
95
96          return;
97  }
98
99  void print_arr(int *arr, int n) {
100         int i;
101         for (i = 0; i < n; i++) {
102                 printf("%d ", arr[i]);
103         }
104         printf("\n");
105 }
```

```c


int main(int argc, char *argv[]) {

        int elem, indx;
        int n = N;

        random_array(arr, n, MOD);
        mergesort_serial(arr, 0, n, buf);

        if (fork()) {
                /* parent */
#if SCENARIO == ZOMBIE
                sleep(10);
                exit(0);
#elif SCENARIO == ORPHAN
                exit(0);
#endif
        }
        else {
                /* child */
                print_arr(arr, n);
                printf("\nElement: ");
                fflush(stdin);
                scanf("%d", &elem);
                indx = bin_search(arr, n, elem);
                if (indx >= 0) {
                        printf("Found %d at %d\n", elem, indx);
                }
                else {
                        printf("Element %d not found\n", elem);
                }
#if SCENARIO == ORPHAN
                printf("Child = %d, Parent = %d\n", getpid(), getppid());
#endif
        }

}
```

Output

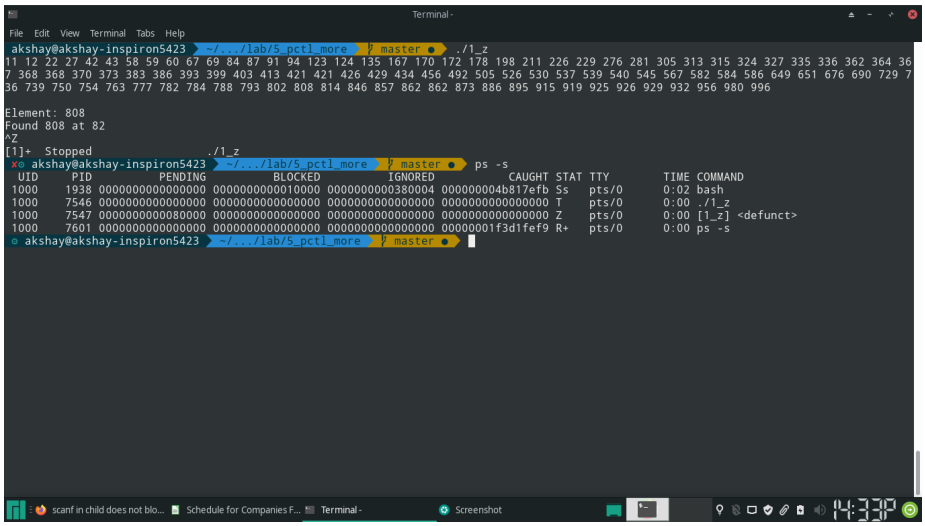

Figure 1: Zombie state formed- process having pid 7547- the second last line of the ps output is the one which zombied

Q2

> The parent starts as many child processes as to the value of its integer command line argument. The child processes simply sleep for the time specified by the argument, then exit. After starting all the children, the parent process does not wait for them immediately, but after a time specified by command line argument, checks the status of all terminated children, print the list of non terminated children and then terminates itself.

```
1
2
3  #include <sys/types.h>
4  #include <sys/wait.h>
5  #include <unistd.h>
6  #include <stdlib.h>
7  #include <stdio.h>
8  #include <errno.h>
9
10 #define SIZE 100
11 static pid_t children[SIZE];
12 static int child_sleep_time[SIZE];
13
14 int main(int argc, char *argv[]) {
15         if (argc < 3) {
16                 fprintf(stderr, "usage: ./2 <no_of_children> <parent_sleep_time>  \
17                         [<child1_sleeptime> ... ]\n");
```

```
18                  return EINVAL;
19          }
20
21          int n_child, n_parent_sleep;
22
23          n_child = atoi(argv[1]);
24
25          if (n_child > SIZE) {
26                  fprintf(stderr, "Maximum number of children is %d\n", SIZE);
27                  return EINVAL;
28          }
29
30          if (argc != (n_child + 3)) {
31                  fprintf(stderr, "Please specify sleep time for each child\n");
32                  return EINVAL;
33          }
34
35          n_parent_sleep = atoi(argv[2]);
36
37
38          int i;
39          int status;
40          int pidret;
41
42          for (i = 0; i < n_child; i++) {
43                  child_sleep_time[i] = atoi(argv[3 + i]);
44          }
45
46
47          for (i = 0; i < n_child; i++) {
48                  if ((children[i] = fork()) == -1) {
49                          perror("fork");
50                          return errno;
51                  }
52                  else if (!children[i]) {
53                          sleep(child_sleep_time[i]);
54                          exit(0);
55                  }
56          }
57
58          sleep(n_parent_sleep);
59
60          for (i = 0; i < n_child; i++) {
61                  printf("Child: %d\t", children[i]);
62                  if ((pidret = waitpid(children[i], &status, WNOHANG)) == 0) {
63                          /* child process has not changed state, is still running
64                           * */
65                          printf("RUNNING\n");
66                  }
67                  else if (pidret != -1) {
```

```
68                              printf("EXITED\n");
69                      }
70                      else {
71                              printf("ERROR\n");
72                      }
73              }
74
75              return 0;
76      }
```

Output

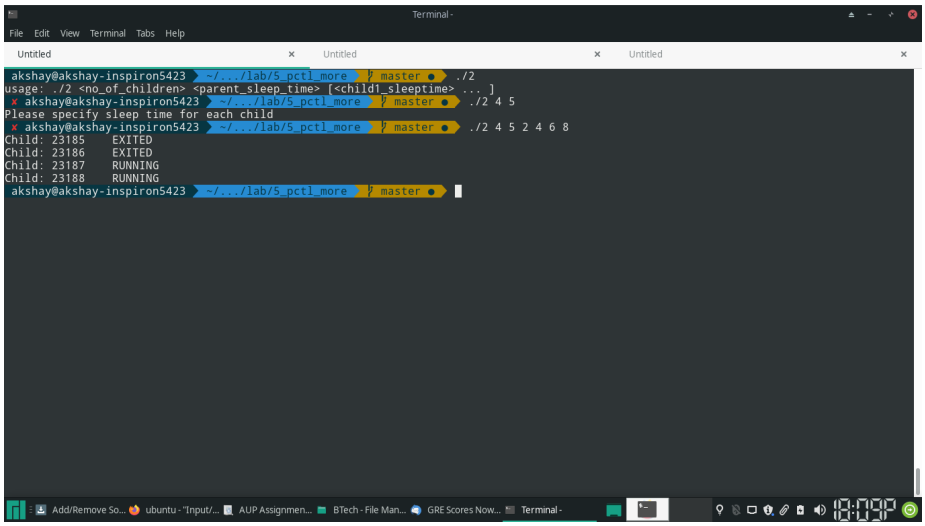

Figure 2: The children whose sleep time is less than the process sleep time (5) are found to have EXITED. Those whose sleep time is more than process sleep time are RUNNING.

Q3

Write a program to create 2 child processes that ultimately become zombie processes. The first child displays some message and immediately terminates. The 2nd child sleeps for 100 and then terminates. Inside the parent program using "system" display the all the process stats and the program exits. Immediately on the command prompt display the all the process stats. What happened to the Zombie processes?

Answer

- It has not been specified that the parent should wait. If the parent does not sleep for more than 100 second, then the child which sleeps **will become orphan, not zombie**.
- The child which exits immediately is zombied. This can be seen by first calling *ps* using system *inside* the program, and immediately calling *ps* after the main program exists on the command prompt.
- The first *ps* (in the program) shows that one of the process is Zombie(Z), and the other is Sleeping(S).
- The second *ps* shows that the other process is Sleeping(S), and it's parent is init, indicated by the ppid printed.

What happens to the zombie process?

The child which exits is zombied. When the parent exits, it effectively becomes like an **orphan** gets attached to *init*. Because it has terminated, it gets **reaped** by init.

On the other hand, the child which is sleeping gets orphaned, as the parent exits before it.

Code

```
1
2
3  #include <sys/types.h>
4  #include <unistd.h>
5  #include <stdlib.h>
6  #include <stdio.h>
7  #include <errno.h>
8
9  void child_1_action(void) {
10         printf("I am the first-born: %d\n", getpid());
11         exit(0);
12 }
13
14 void child_2_action(void) {
15         sleep(100);
16         exit(0);
```

```
17  }
18
19  int main(int argc, char *argv[]) {
20
21          int childpid;
22          void (*child_action[])(void) = {child_1_action, child_2_action};
23
24          for (int i = 0; i < 2; i++) {
25                  if ((childpid = fork()) == -1) {
26                          perror("fork");
27                          return errno;
28                  }
29                  else if (!childpid) {
30                          /* in child */
31                          child_action[i]();
32                          /* will never return here, function has exit() in it */
33                  }
34                  /* is parent */
35          }
36
37          if (system("ps -o command,pid,ppid,state") == -1) {
38                  perror("ps -o command,pid,ppid,state");
39          }
40
41          return 0;
42  }
```

Output

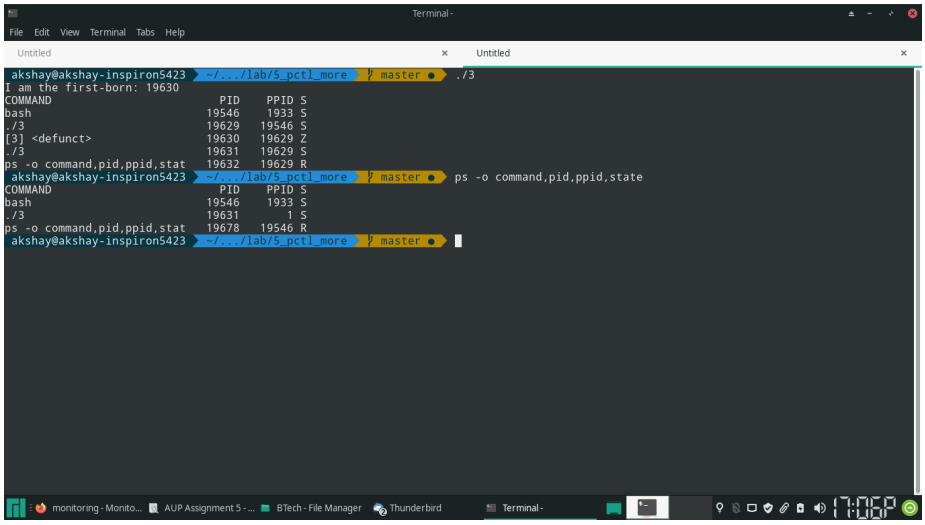

Figure 3: Zombied and Sleeping processes. The process which exits is seen as "Z" in the first output, and the sleeping process is seen to have PPID = 1 (meaning it is orphaned