Assignment 2

2a - Orphan, Zombie Process

Code

```
/*
Problem Statement - Implement the C program in which main program accepts the
integers to be sorted. Main program uses the FORK system call to create a new
process called a child process. Parent process sorts the integers using sorting
algorithm and waits for child process using WAIT system call to sort the integers
using any sorting algorithm. Also demonstrate zombie and orphan states
*/

#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>

// using merge and quick sort, but any sorting algorithm will be fine
void merge(int a[], int l, int m, int r) {
  int i, j, k;
  int n1 = m - l + 1;
  int n2 = r - m;

  int L[n1], R[n2];

  for (i = 0; i < n1; i++)
    L[i] = a[l + i];
  for (j = 0; j < n2; j++)
    R[j] = a[m + 1 + j];
```

```
  i = 0;
  j = 0;
  k = l;

  while (i < n1 && j < n2) {
    if (L[i] <= R[j]) {
      a[k] = L[i];
      i++;
    } else {
      a[k] = R[j];
      j++;
    }
    k++;
  }

  while (i < n1) {
    a[k] = L[i];
    i++;
    k++;
  }

  while (j < n2) {
    a[k] = R[j];
    j++;
    k++;
  }
}

void mergeSort(int a[], int l, int r) {
  if (l < r) {
    int m = l + (r - l) / 2;

    mergeSort(a, l, m);
    mergeSort(a, m + 1, r);

    merge(a, l, m, r);
  }
}

void printArray(int A[], int n) { int i; }

void swap(int *a, int *b) {
  int t = *a;
  *a = *b;
  *b = t;
}

int partition(int a[], int low, int high) {
  int pivot = a[high];
```

```
  int i = (low - 1);

  for (int j = low; j <= high - 1; j++) {
    if (a[j] < pivot) {
      i++;
      swap(&a[i], &a[j]);
    }
  }
  swap(&a[i + 1], &a[high]);
  return (i + 1);
}

void quickSort(int a[], int low, int high) {
  if (low < high) {
    int pi = partition(a, low, high);

    quickSort(a, low, pi - 1);
    quickSort(a, pi + 1, high);
  }
}

// implementing first part of the statement
void normal() {
  int n;
  printf("array size\n");
  scanf("%d", &n);

  int a[n];

  printf("\nEnter array\n");
  for (int i = 0; i < n; i++) {
    scanf("%d", &a[i]);
  }

  printf("\nGiven array is \n");
  for (int i = 0; i < n; i++)
    printf("%d ", a[i]);
  printf("\n");

  pid_t pid = fork();

  if (pid == -1) {
    printf("Error: error while executing fork()");
    return;
  }

  // child quick sort
  if (pid == 0) {
    printf("Child is: %d\n", (int)getpid());
    printf("Parent is: %d\n", (int)getppid());
```

```
    quickSort(a, 0, n - 1);
    printf("Sorted array \n");
    for (int i = 0; i < n; i++)
      printf("%d ", a[i]);
    printf("\n");
    exit(0); //child exiting and informing the parent that child has been
terminated
  }
  // parent merge sort
  else {
    pid_t cpid = wait(NULL); //parent waiting for all child to finish and start
after receiving the response from child
(https://stackoverflow.com/a/42426884/17222693)

    printf("Process ID is: %d\n", (int)getpid());

    mergeSort(a, 0, n - 1);

    printf("Sorted array is \n");
    for (int i = 0; i < n; i++)
      printf("%d ", a[i]);
    printf("\n");
  }
}

//child becomes orphan beacause parent exits without waiting for the child process
to finish
void orphan() {
  pid_t pid = fork();

  if (pid == 0) {
    printf("\nChild %d is going to sleep for 5 sec...\n", (int)getpid());
    sleep(10); //explicitly running child process for desired long enough time,
parentid will be different before and after sleep()
    printf("\nAfter child wakes up...\nParent of child is %d\n", (int)getppid());
//prints id of init process (after re-parenting) as original parent has been
exited
    system("ps -elf | grep fork");  //system command to display all the processes
with current program('fork') in full format

    exit(0); //child exits immediately
  } else {
    sleep(5);
    printf("\nParent Process %d, child is still executing\nParent exits.\n",
(int)getpid()); //parent exits without waiting for the child to finish
  }
  return;
}

//child becomes zombie as it exits without informing it's parent, so parent
```

```
doesn't actually know that the child has finished
void zombie() {
  pid_t pid = fork();

  if (pid > 0) {
    system("ps -efl | grep defunct"); //parent start executing without checking
child's status
    sleep(2); //explicitly running parent's block for desired long enough time so
that in meantime child will exit
    system("ps -efl | grep defunct");
  } else {
    printf("Inside child process, Parent process ID: %d\nChild exit.\n",
getppid());

    exit(0);
  }
  return;
}

int main() {
  int opt = 1;

  while (opt) {
    printf("1 Normal\n");
    printf("2 Orphan\n");
    printf("3 Zombie\n");
    printf("Enter an option: ");
    scanf("%d", &opt);

    switch (opt) {
    case 1:
      normal();
      break;
    case 2:
      orphan();
      break;
    case 3:
      zombie();
      break;
    default:
      break;
    }
  }

  return 0;
}

/*
gcc 'Assignment 2a.c' -o 'Assignment 2a.out'
*/
```

Output

```
abhishek-jadhav@abhishek-jadhav-ubuntu:~/Codes/OS Assignments/33232$ ./a.out
1 Normal
2 Orphan
3 Zombie
Enter an option: 1
array size
4

Enter array
1 2 3 4

Given array is
1 2 3 4
Child is: 4660
Parent is: 4659
Sorted array
1 2 3 4
Process ID is: 4659
Sorted array is
1 2 3 4
1 Normal
2 Orphan
3 Zombie
Enter an option: 2

Child 4661 is going to sleep for 5 sec...

Parent Process 4659, child is still executing
Parent exits.
1 Normal
2 Orphan
3 Zombie
Enter an option:
After child wakes up...
Parent of child is 4659
4 S message+     825       1  0  80   0 -  2721 -      13:17 ?        00:00:03
@dbus-daemon --system --address=systemd: --nofork --nopidfile --systemd-activation
--syslog-only
0 S abhishe+    1750    1733  0  80   0 -  3632 ep_pol 13:19 ?        00:00:04
/usr/bin/dbus-daemon --session --address=systemd: --nofork --nopidfile --systemd-
activation --syslog-only
0 S abhishe+    1865    1857  0  80   0 -  2058 ep_pol 13:19 ?        00:00:00
/usr/bin/dbus-daemon --config-file=/usr/share/defaults/at-spi2/accessibility.conf
--nofork --print-address 3
```

```
0 S abhishe+    4662    4661  0  80   0 -   718 do_wai 14:44 pts/0    00:00:00 sh
-c ps -elf | grep fork
0 S abhishe+    4664    4662  0  80   0 -  4448 pipe_r 14:44 pts/0    00:00:00
grep fork
^C
abhishek-jadhav@abhishek-jadhav-ubuntu:~/Codes/OS Assignments/33232$ ./a.out
1 Normal
2 Orphan
3 Zombie
Enter an option: 3
Inside child process, Parent process ID: 4665
Child exit.
1 Z abhishe+    4666    4665  0  80   0 -     0 -      14:46 pts/0    00:00:00
[a.out] <defunct>
0 S abhishe+    4667    4665  0  80   0 -   718 do_wai 14:46 pts/0    00:00:00 sh
-c ps -efl | grep defunct
0 S abhishe+    4669    4667  0  80   0 -  4448 pipe_r 14:46 pts/0    00:00:00
grep defunct
1 Z abhishe+    4666    4665  0  80   0 -     0 -      14:46 pts/0    00:00:00
[a.out] <defunct>
0 S abhishe+    4670    4665  0  80   0 -   718 do_wai 14:46 pts/0    00:00:00 sh
-c ps -efl | grep defunct
0 S abhishe+    4672    4670  0  80   0 -  4448 pipe_r 14:46 pts/0    00:00:00
grep defunct
```

2b - Reverse array using Exec

Code

2b.c

```c
/*
Problem Statement - Implement the C program in which main program accepts an
integer array. Main program uses the FORK system call to create a new process
called a child process. Parent process sorts an integer array and passes the
sorted array to child process through the command line arguments of EXECVE system
call. The child process uses EXECVE system call to load new program which display
array in reverse order
*/

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
```

```
// using merge sort, but any sorting algorithm will be fine
void merge(int a[], int l, int m, int r) {
  int i, j, k;
  int n1 = m - l + 1;
  int n2 = r - m;

  int L[n1], R[n2];

  for (i = 0; i < n1; i++) {
    L[i] = a[l + i];
  }

  for (j = 0; j < n2; j++) {
    R[j] = a[m + 1 + j];
  }

  i = 0;
  j = 0;
  k = l;

  while (i < n1 && j < n2) {
    if (L[i] <= R[j]) {
      a[k] = L[i];
      i++;
    } else {
      a[k] = R[j];
      j++;
    }
    k++;
  }

  while (i < n1) {
    a[k] = L[i];
    i++;
    k++;
  }

  while (j < n2) {
    a[k] = R[j];
    j++;
    k++;
  }
}

void sort(int a[], int l, int r) {
  if (l < r) {
    int m = l + (r - l) / 2;
    sort(a, l, m);
    sort(a, m + 1, r);
    merge(a, l, m, r);
```

```
  }
}

int main() {
  //-------------- Taking array input and sorting it --------------
    int n;
    printf("\nNo of elements\n");
    scanf("%d", &n);

    int a[n];
    printf("\nElements\n");
    for (int i = 0; i < n; i++) {
        scanf("%d", &a[i]);
    }

    sort(a, 0, n - 1);

    printf("\nSorted array\n");
    for (int i = 0; i < n; i++) {
        printf("%d ", a[i]);
    }
    printf("\n");
  //---------------------------------------------------------------

  char strArr[n][1000];  //for storing the array elements as strings
  for (int i = 0; i < n; i++) {
    sprintf(strArr[i], "%d", a[i]);  //stores prepared string to the buffer
strArr[i]
  }
  char *args[n + 2]; //filename + no. of array elements + NULL item = n + 2
  args[0] = "./reverse"; //first argument to the exec() is the filename of the
executable

  for (int i = 0; i < n; i++) {
    args[i + 1] = strArr[i]; //storing array element in the argument array
  }

  args[n + 1] = NULL; //last item of argument as NULL

  execvp(args[0], args); //calling another program using exevp() system call
  return 0;
}

/* Sequence is important here

gcc reverse.c -o reverse.out
gcc 'Assignment 2b.c' -o 'Assignment 2b.out'
./'Assignment 2b.out'

*/
```

reverse.c

```c
#include <stdio.h>
#include <stdlib.h>

//argc provides the argument count
int main(int argc, char *argv[]) {
  int n = argc - 1; //excluding the last NULL character
  int arr[n]; //preparing new array of elements
  for (int i = 1; i <= n; i++) { // i = 1 because, argv[0] is pointing to filename
    arr[i - 1] = atoi(argv[i]); //charater string to integer
  }

  //printing the provided array in reverse order
  printf("\nInside child process through execvp() command...\nArray in reverse
order: ");
  for(int i = n - 1; i >= 0; i--) {
    printf("%d ", arr[i]);
  }
  printf("\n");
  return 0;
}
```

Output

```
abhishek-jadhav@abhishek-jadhav-ubuntu:~/Codes/OS Assignments/33232$ ./a.out

No of elements
5

Elements
5 4 3 2 1

Sorted array
1 2 3 4 5
```