Assignment 2(a)

Roll no: 33229

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

void swap(int *a, int *b) {
  int temp = *a;
  *a = *b;
  *b = temp;
}
void heapify(int arr[], int N, int i) {
  int largest = i;
  int l = 2 * i + 1;
  int r = 2 * i + 2;
  if (l < N && arr[l] > arr[largest])
    largest = l;

  if (r < N && arr[r] > arr[largest])
    largest = r;

  if (largest != i) {
    swap(&arr[i], &arr[largest]);
    heapify(arr, N, largest);
  }
}
void heapSort(int arr[], int N) {
  for (int i = N / 2 - 1; i >= 0; i--)
    heapify(arr, N, i);

  for (int i = N - 1; i > 0; i--) {
    swap(&arr[0], &arr[i]);
    heapify(arr, i, 0);
  }
}
void printArray(int arr[], int size) {
  for (int i = 0; i < size; i++) {
    printf("%d ", arr[i]);
  }
  printf("\n");
}
void Zombie() {
  pid_t pid = fork();

  if (pid < 0) {
    perror("fork");
    exit(1);
  } else if (pid == 0) {
    // Child process (zombie)
    printf("Child process %d started, parent process is %d.\n", getpid(), getppid());
    exit(0);
  } else {
    // Parent process
    printf("Parent process %d created child process %d.\n", getpid(), pid);
    sleep(3);
    system("ps -eo pid,ppid,state,cmd | grep defunct");
```

```
        printf("Parent process %d is terminating.\n", getpid());
    }
    sleep(5);
}


void Orphan() {
    pid_t pid = fork();

    if (pid < 0) {
        // Error occurred
        perror("fork");
        exit(1);
    } else if (pid == 0) {
        // Child process (orphan)
        sleep(3); // Ensure the parent terminates first
        printf("Child process %d started, parent process is %d.\n", getpid(), getppid());
        system("ps -eo pid,ppid,state,cmd | grep a.out");
        printf("Child process %d finished, now adopted by init (PID: %d).\n", getpid(), getppid());
    } else {
        printf("Parent process %d created child process %d.\n", getpid(), pid);
        system("ps -eo pid,ppid,state,cmd | grep a.out");
        printf("Parent process %d is terminating.\n", getpid());
        exit(0); // Parent exits, leaving the child orphaned
    }
    sleep(3);
}


void SortByWaitCall(int arr[], int n) {
    pid_t pid = fork();

    if (pid < 0) {
        perror("fork failed");
        exit(1);
    } else if (pid == 0) {
        // Child process
        printf("Child process sorting with Heap Sort...\n");
        heapSort(arr, n);
        printf("Child process sorted array: ");
        printArray(arr, n);
        printf("Child process (PID: %d) finished.\n", getpid());
        exit(0);
    } else {
        // Parent process
        printf("Parent process sorting with Heap Sort...\n");
        heapSort(arr, n);
        printf("Array sorted and wait called\n");

        // Wait for the child process to finish
        int status;
        pid_t child_pid = wait(&status);

        if (child_pid < 0) {
            perror("wait failed");
        } else {
            printf("Parent process (PID: %d) waited for child process (PID: %d)\n", getpid(), child_pid);
        }
    }
```

```
}

int main() {
   int n, i, c;

   printf("Enter number of integers to sort: ");
   scanf("%d", &n);

   int arr[n];
   printf("Enter the integers:\n");
   for (i = 0; i < n; i++) {
      scanf("%d", &arr[i]);
   }

   printf("Enter choice: 1. Zombie 2. Orphan 3. Using wait and sort\n");
   scanf("%d", &c);

   switch (c) {
      case 1:
         Zombie();
         break;

      case 2:
         Orphan();
         break;

      case 3:
         SortByWaitCall(arr, n);
         break;

      default:
         printf("Invalid choice\n");
         break;
   }

   return 0;
}
```

Output:

```
Enter number of integers to sort: 4
Enter the integers:
56 23 56 9
Enter choice: 1. Zombie 2. Orphan 3. Using wait and sort
1
Parent process 3985 created child process 3992.
Child process 3992 started, parent process is 3985.
  3992   3985 Z [assi2] <defunct>
  3993   3985 S sh -c ps -eo pid,ppid,state,cmd | grep defunct
  3995   3993 S grep defunct
Parent process 3985 is terminating.

sakshi711@sakshi711:~$ ./assi2
Enter number of integers to sort: 4
Enter the integers:
7 3 6 8
Enter choice: 1. Zombie 2. Orphan 3. Using wait and sort
2
Parent process 3996 created child process 3997.
```

```
  3998   3996 S sh -c ps -eo pid,ppid,state,cmd | grep a.out
  4000   3998 R grep.out
  4000   3998 R grep a.out
Parent process 3996 is terminating.
sakshi711@sakshi711:~$ Child process 3997 started, parent process is 1315.
  4001   3997 S sh -c ps -eo pid,ppid,state,cmd | grep a.out
  4003   4001 R grep a.out
Child process 3997 finished, now adopted by init (PID: 1315).

sakshi711@sakshi711:~$ ./assi2
Enter number of integers to sort: 5
Enter the integers:
34 78 23 76 29
Enter choice: 1. Zombie 2. Orphan 3. Using wait and sort
3
Parent process sorting with Heap Sort...
Array sorted and wait called
Child process sorting with Heap Sort...
Child process sorted array: 23 29 34 76 78
Child process (PID: 4006) finished.
Parent process (PID: 4005) waited for child process (PID: 4006)
```

Assignment 2(b)

main.c

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>
#include <sys/wait.h>
void swap(int *a, int *b) {
 int temp = *a;
 *a = *b;
 *b = temp;
}

void heapify(int arr[], int N, int i) {
 int largest = i;
 int l = 2 * i + 1;
 int r = 2 * i + 2;
 if (l < N && arr[l] > arr[largest]) largest = l;
 if (r < N && arr[r] > arr[largest]) largest = r;
 if (largest != i) {
   swap(&arr[i], &arr[largest]);

   heapify(arr, N, largest);
 }
}
void heapSort(int arr[], int N) {
 for (int i = N / 2 - 1; i >= 0; i--) heapify(arr, N, i);
 for (int i = N - 1; i > 0; i--) {
   swap(&arr[0], &arr[i]);
   heapify(arr, i, 0);
 }
}

int main(){
   printf("Fork initialising\n");
   int n;
   printf("Enter number of integers to sort: ");
   scanf("%d", &n);
   int arr[n];
   printf("Enter the integers:\n");
   for (int i = 0; i < n; i++) {
       scanf("%d", &arr[i]);
   }
   heapSort(arr, n);
   printf("Array sorted: ");
   for (int i = 0; i < n; i++) {
       printf("%d ", arr[i]);
   }
   printf("\n");
   pid_t pid = fork();
   if (pid < 0) {
       printf("Error occurred\n");
   } else if (pid == 0) {
       printf("Inside child process\n");

       char *argv[n + 2];
       argv[0] = (char *)"./hello";
       for (int i = 0; i < n; i++) {
           argv[i + 1] = (char *)malloc(20 * sizeof(char)); // allocate
memory for each argument
           snprintf(argv[i + 1], 20, "%d", arr[i]);
       }
       argv[n + 1] = NULL;
       execv(argv[0], argv);
   } else {
       sleep(10);
       printf("Inside parent process\n");
   }
   return 0;
}
```

Hello.c

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

int main(int argc, char *argv[]) {
    printf("exec executed\n");
    printf("The PID of this process is: %d\n", getpid();
    if (argc > 1) {
        printf("Arguments received in reverse order: ");
        for (int i = argc - 1; i > 0; --i) {
            printf("%s ", argv[i]);
        }
        printf("\n");
    } else {
        printf("No arguments received.\n");
    }
    return 0;
}
```

Output :

```
swikar@LAPTOP-3VLQDHIH:~$ g++ hello.c -o hello
swikar@LAPTOP-3VLQDHIH:~$ ./hello
exec executed
The PID of this process is: 2207
No arguments received.
swikar@LAPTOP-3VLQDHIH:~$ g++ a2b.c
swikar@LAPTOP-3VLQDHIH:~$ ./a.out
Fork initialising
Enter number of integers to sort: 5
Enter the integers:
2 1 4 5 7
Array sorted: 1 2 4 5 7
Inside child process
exec executed
The PID of this process is: 2336
Arguments received in reverse order: 7 5 4 2 1
Inside parent process
```