

## Assignment2A

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

// Bubble Sort for parent
void bubble_sort(int arr[], int n) {
    for (int i = 0; i < n - 1; i++)
        for (int j = 0; j < n - i - 1; j++)
            if (arr[j] > arr[j + 1]) {
                int t = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = t;
            }
}

// Merge Sort for child
void merge(int arr[], int left, int mid, int right) {
    int n1 = mid - left + 1, n2 = right - mid;
    int *L = malloc(n1 * sizeof(int));
    int *R = malloc(n2 * sizeof(int));

    for (int i = 0; i < n1; i++) L[i] = arr[left + i];
    for (int i = 0; i < n2; i++) R[i] = arr[mid + 1 + i];

    int i = 0, j = 0, k = left;
    while (i < n1 && j < n2)
        arr[k++] = (L[i] <= R[j]) ? L[i++] : R[j++];
    while (i < n1) arr[k++] = L[i++];
    while (j < n2) arr[k++] = R[j++];

    free(L); free(R);
}

void merge_sort(int arr[], int left, int right) {
    if (left < right) {
        int mid = left + (right - left) / 2;
        merge_sort(arr, left, mid);
        merge_sort(arr, mid + 1, right);
        merge(arr, left, mid, right);
    }
}

int main() {
    int choice, n;

    printf("Choose the process state to simulate:\n");
    printf("1. Normal execution (child sorts first, then parent)\n");
    printf("2. Zombie process\n");
    printf("3. Orphan process\n");
}
```

```

printf("Enter choice: ");
scanf("%d", &choice);

if (choice == 1) {
// Normal sorting with parent wait
printf("Enter number of elements: ");
scanf("%d", &n);
int arr[n], arr_copy[n];

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

pid_t pid = fork();

if (pid < 0) {
perror("fork failed");
exit(1);
} else if (pid == 0) {
// Child: merge sort first
merge_sort(arr_copy, 0, n - 1);
printf("\nChild process (Merge Sort): ");
for (int i = 0; i < n; i++) printf("%d ", arr_copy[i]);
printf("\n");
_exit(0);
} else {
wait(NULL); // Wait for child to finish
bubble_sort(arr, n);
printf("\nParent process (Bubble Sort): ");
for (int i = 0; i < n; i++) printf("%d ", arr[i]);
printf("\nParent: Child process finished.\n");
}
}

else if (choice == 2) {
// Zombie process demo
pid_t pid = fork();

if (pid < 0) {
perror("fork failed");
exit(1);
} else if (pid == 0) {
printf("Child (Zombie Demo) PID: %d exiting immediately\n", getpid());
_exit(0); // Child exits immediately -> becomes zombie until parent calls wait
} else {
printf("Parent PID: %d sleeping 10 seconds (child becomes zombie)\n", getpid());
sleep(10); // Parent sleeps, child is zombie now

// Now parent collects child exit status to clean zombie
wait(NULL);
printf("Parent waited and cleaned zombie child\n");
}
}
}

```

```

}

}

else if (choice == 3) {
// Orphan process demo
pid_t pid = fork();

if (pid < 0) {
perror("fork failed");
exit(1);
} else if (pid == 0) {
// Child sleeps so that parent exits early, child becomes orphan
printf("Child (Orphan Demo) PID: %d, parent PID: %d\n", getpid(), getppid());
sleep(10);
printf("Child after sleep, new parent PID: %d\n", getppid());
} else {
printf("Parent PID: %d exiting immediately, child will become orphan\n", getpid());
exit(0); // Parent exits immediately -> child orphaned
}
}
else {
printf("Invalid choice!\n");
}

return 0;
}

```

=====

Choose the process state to simulate:

1. Normal execution (child sorts first, then parent)

2. Zombie process

3. Orphan process

Enter choice: 1

Enter number of elements: 3

Enter 3 integers:

4

2

9

Child process (Merge Sort): 2 4 9

Parent process (Bubble Sort): 2 4 9

Parent: Child process finished.

Choose the process state to simulate:

1. Normal execution (child sorts first, then parent)

2. Zombie process

3. Orphan process

Enter choice: 2

Parent PID: 12588 sleeping 10 seconds (child becomes zombie)

Child (Zombie Demo) PID: 12631 exiting immediately

Parent waited and cleaned zombie child.

Choose the process state to simulate:

1. Normal execution (child sorts first, then parent)
2. Zombie process
3. Orphan process

Enter choice: 3

Parent PID: 12985 exiting immediately, child will become orphan

Child (Orphan Demo) PID: 13023, parent PID: 12985